1x PDI Builder

Release 6.12.54

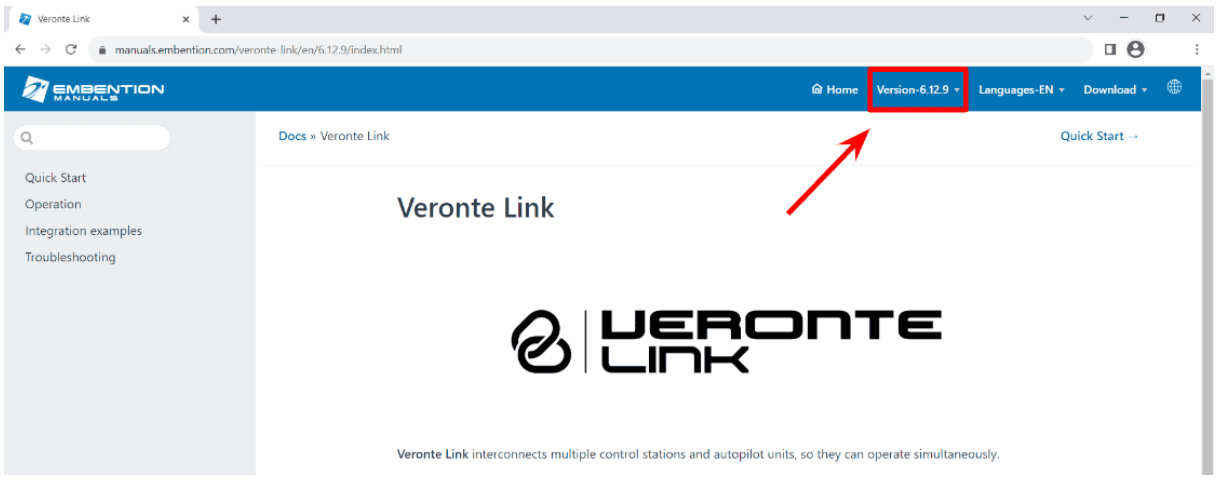
Embention

2024-04-25

CONTENTS

1x PDI Builder is an autopilot configuration tool (control laws, flight phases, operation modes, failsafes, etc.) to adapt it to a specific vehicle.

Warning: Select your version before reading any user manual for software. The following image shows where to select a version from any Embention user manual.



CHAPTER

ONE

QUICK START

1x PDI Builder is the main configuration tool to adapt a Veronte Autopilot 1x to a specific vehicle, including user-defined communication protocols. **1x PDI Builder** includes:

- **Telemetry**: real-time onboard UAV metrics, such as sensors, actuators and control states.
- **Configuration**: edit vehicle settings, such as servo trim, interface/port management and modes.
- **Automations**: actions that are automatically executed when a set of configured conditions are accomplished.
- **Block Programs**: Veronte Autopilot 1x can be programmed (control laws) with a friendly-user programming language.

Once Autopilot 1x has been detected on **Veronte Link**, install **1x PDI Builder**.

1.1 Download

Once **Veronte Autopilot 1x** has been purchased, a GitHub release should be created for the customer with the application.

To access to the release and download the software, read the Releases section of the **Joint Collaboration Framework** manual.

1.2 Installation

To install **1x PDI Builder** on Windows just execute the "1xPDIBuilder.exe" file and follow the indications of the *Setup Wizard*. Administrator rights are needed.

> **Warning:** If users have any problems with the installation, please disable the antivirus and the Windows firewall. Disabling the antivirus depends on the antivirus software.
>
> To disable the firewall:
>
> - Go to "Control Panel" → "System and Security" → "Windows Defender Firewall"
> - Then, click on "Turn windows Defender Firewall on or off".

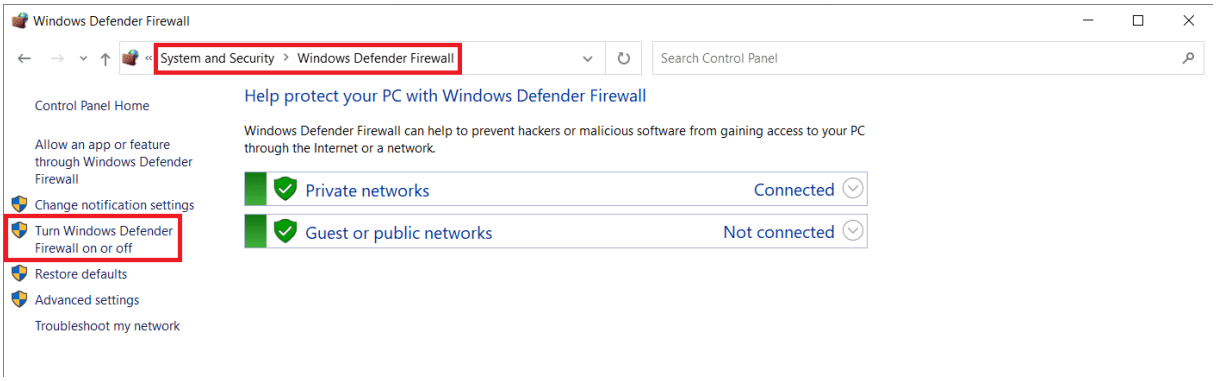


Fig. 1: **Windows Defender Firewall**

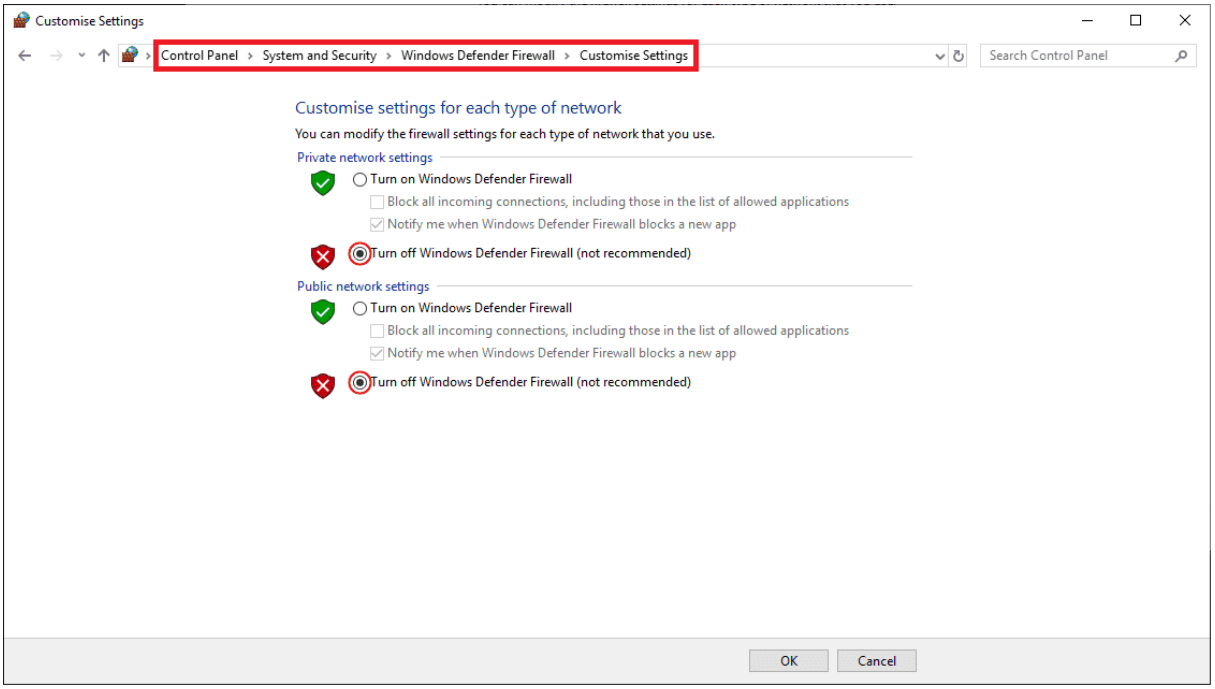


Fig. 2: **Windows Defender Firewall: Settings**

CHAPTER

TWO

CONFIGURATION

This section explains each option and parameter available in **1x PDI Builder**.

Once the installation is finished, open **1x PDI Builder** and select the unit.

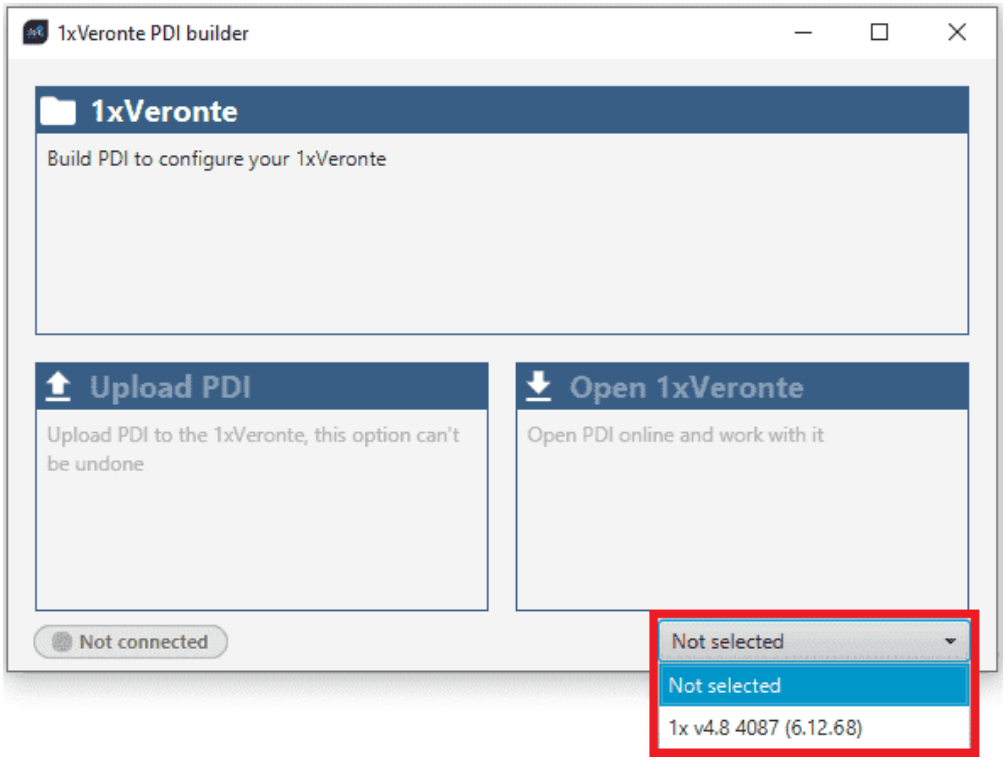


Fig. 1: **Autopilot 1x ID**

If it is correctly connected, **1x PDI Builder** will display the **mode** in which the connected unit is. In addition, a **PDI error button**  will appear:

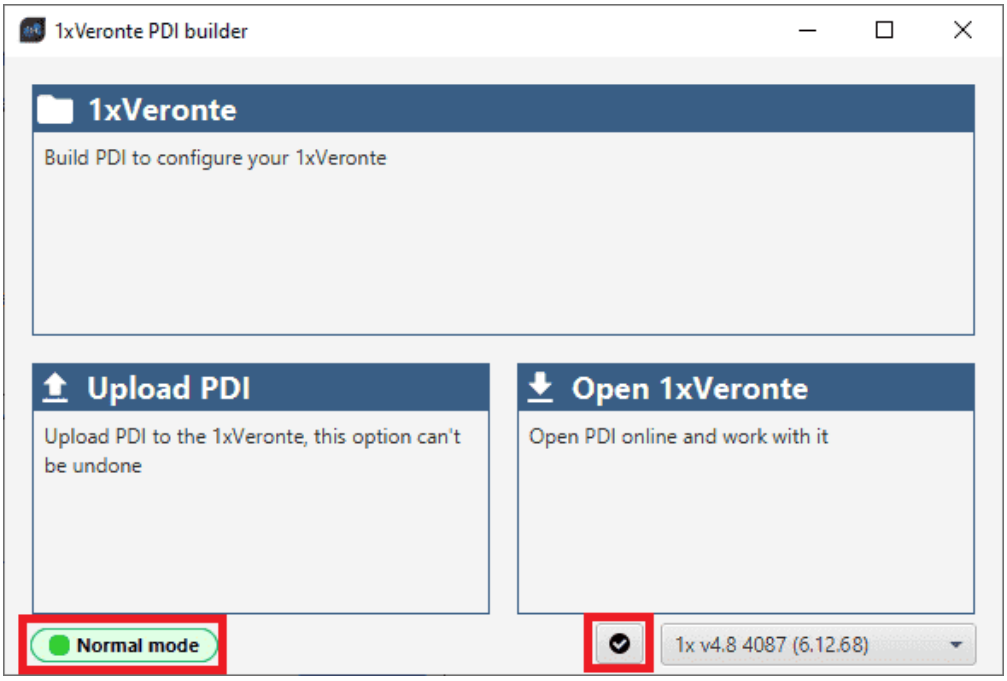


Fig. 2: **1x PDI Builder**

- **1x mode**: 1x unit should appear in **Normal mode**, as shown in the figure above, or **Maintenance mode**.

  It can also appear as **Maintenance mode (loaded with errors)** or **Normal mode - Disconnected**.

Note: **Maintenance mode (loaded with errors)** appears when something is wrong in the configuration. For more information, see *Maintenance mode (loaded with errors) - Troubleshooting* section of this manual.

-  **PDI Errors** button: The user can check if the connected unit has PDI Errors by simply clicking on it. If there are no errors, the following message appears:

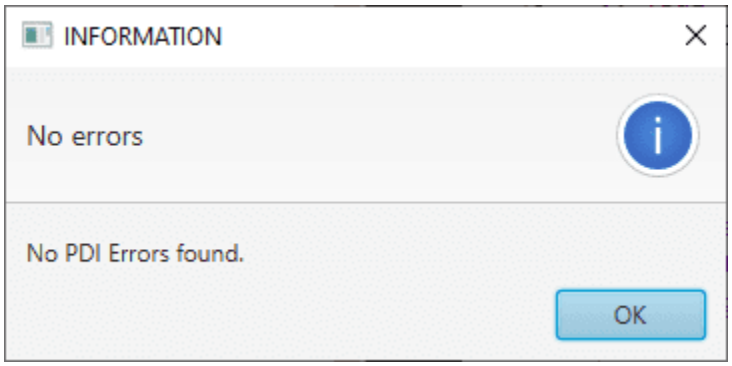


Fig. 3: **1x PDI Builder - PDI Errors message**

The user can access now to 3 configuration options:

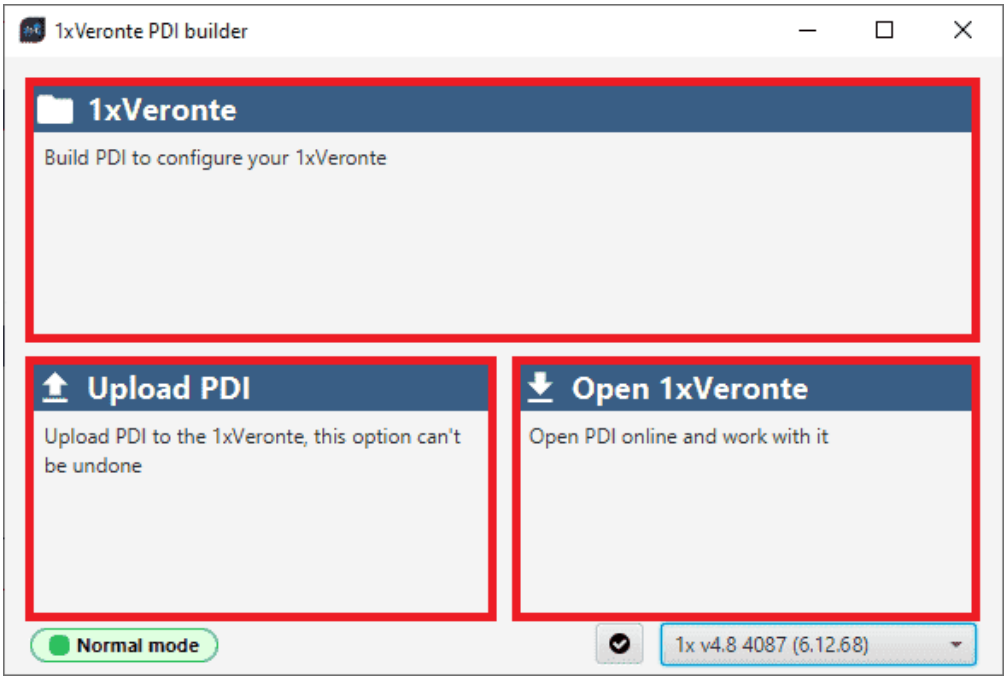


Fig. 4: **1x PDI Builder options**

- **1xVeronte**: It allows the user to work with **offline** configurations. A previously exported 1x PDI configuration can be opened and modified or it is possible to build a new one from the default configuration.

Note: When an offline configuration is opened, it is possible to select the hardware version the user wants to work with:

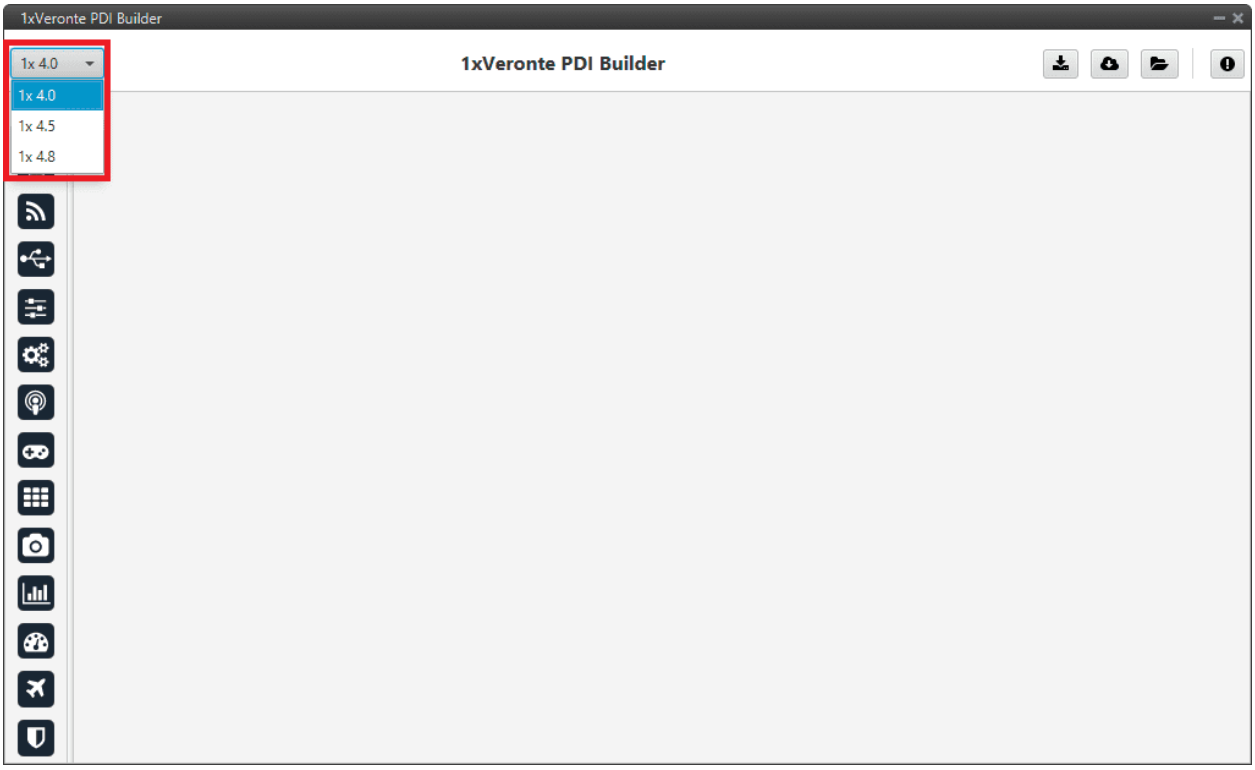


Fig. 5: **Hardware versions**

When switching between hardware versions, the following confirmation message appears:

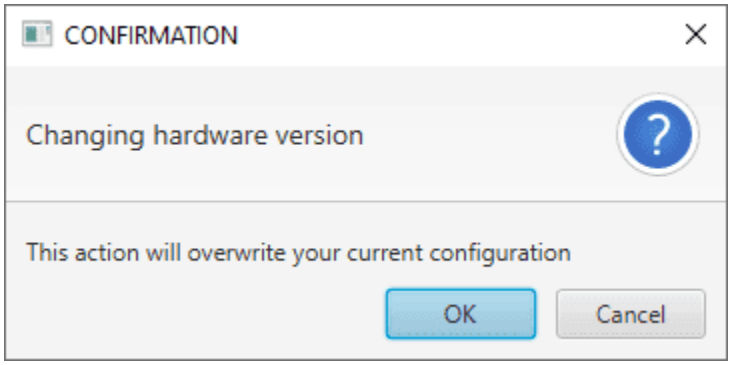


Fig. 6: **Hardware versions - Confirmation message**

- **Upload PDI**: A previously exported 1x PDI configuration can be imported to the linked 1x.

  This option is also intended to **upload** to Autopilot 1x a **calibration** previously exported with the **1x PDI Calibration** software.

> **Warning:** When a configuration is loaded into Veronte Autopilot 1x with a version older than the software version being used, an **automatic migration** from the configuration version to the software version being used will be performed.

For more information on this, see *Migrate configuration - Troubleshooting* section of this manual.

- **Open 1xVeronte**: By clicking on this option, **1x PDI Builder** configuration menu opens with the configuration (the PDI files) loaded in the connected **1x**. Then, the user can modify it **online**.

Note: PDI files are 1x configuration files. These files allow for modular control with improved version management. These PDI files are split in 2 folders. Each folder hold several .xml files:

- **operation**: This folder holds all the files related with the operations defined, such as waypoints, routes, operative parameters, runways, etc.
- **setup**: This contains the configuration of the vehicle. All the control loops and their parameters, the definition of the flight phases and guidance commands, and the automations defined are stored here.

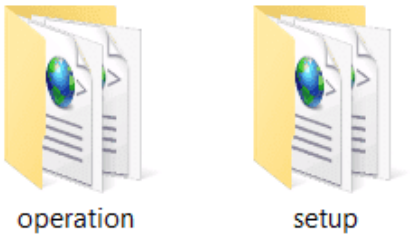


Fig. 7: **PDIs files**

Finally, click on '**1xVeronte**' to edit a configuration offline or '**Open 1xVeronte**' to open the current configuration and start editing it online. The different 'buttons' that can be seen in the initial menu of the **1x PDI Builder** are explained below.

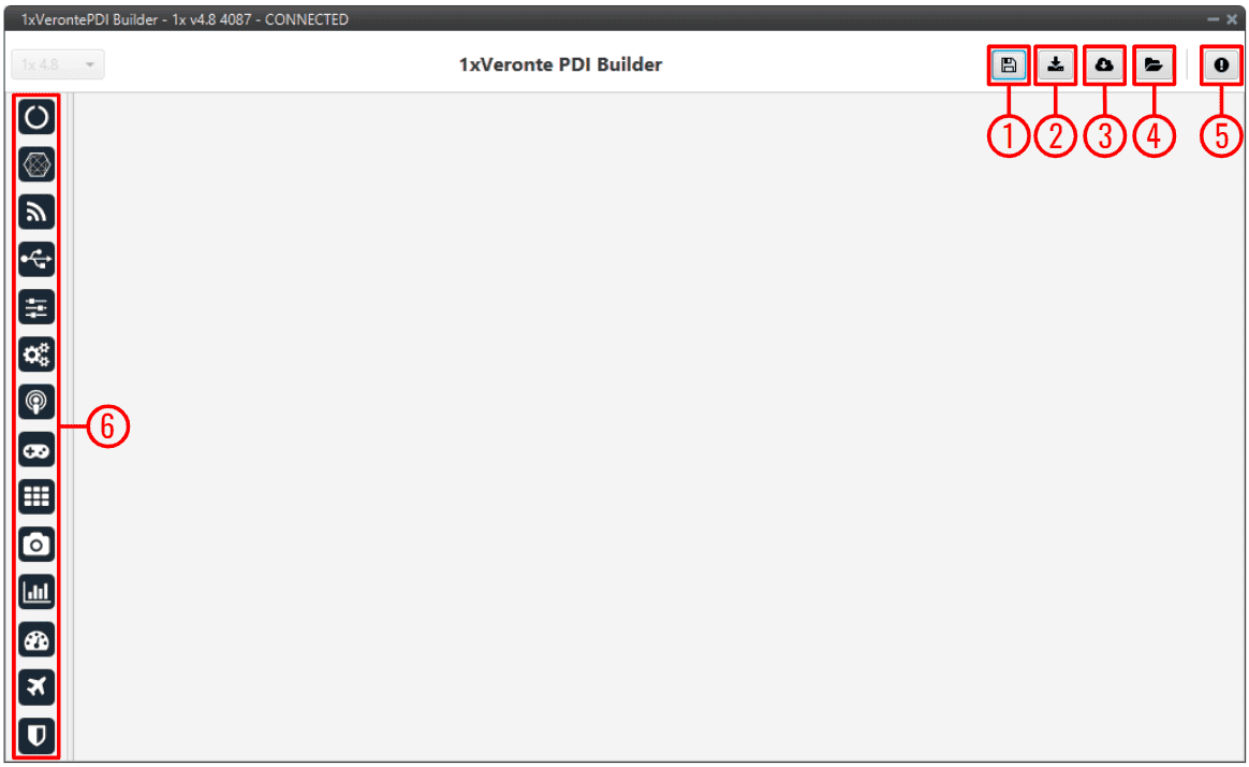


Fig. 8: **Initial menu**

1. **Save and close**: After changes are done, press on the save button to apply the changes.

   While saving, a percentage of saving process is displayed:

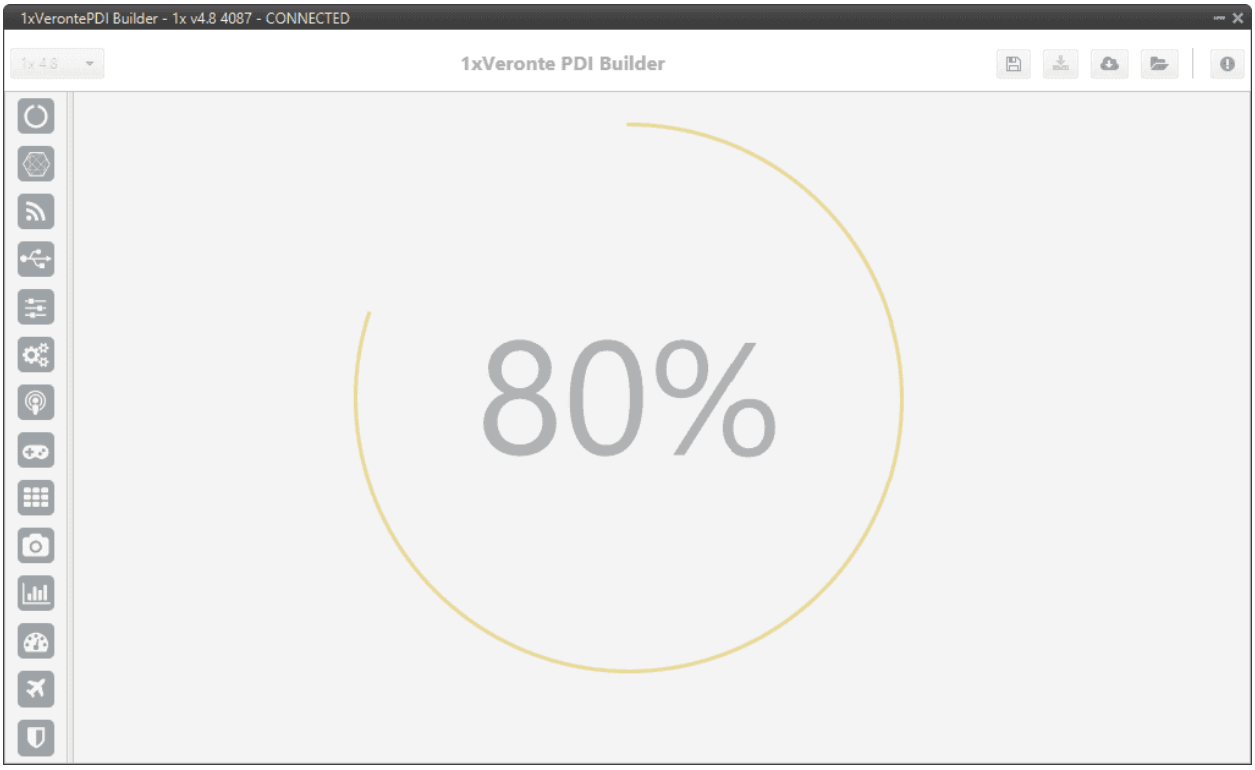


Fig. 9: **Save and close**

In order to save the configuration, Veronte Autopilot 1x must enter in **maintenance mode**. Then, after saving any changes, Autopilot 1x will **RESET** and **1x PDI Builder** software will consequently **close**. Therefore, users must accept the following confirmation panels for this to be possible:

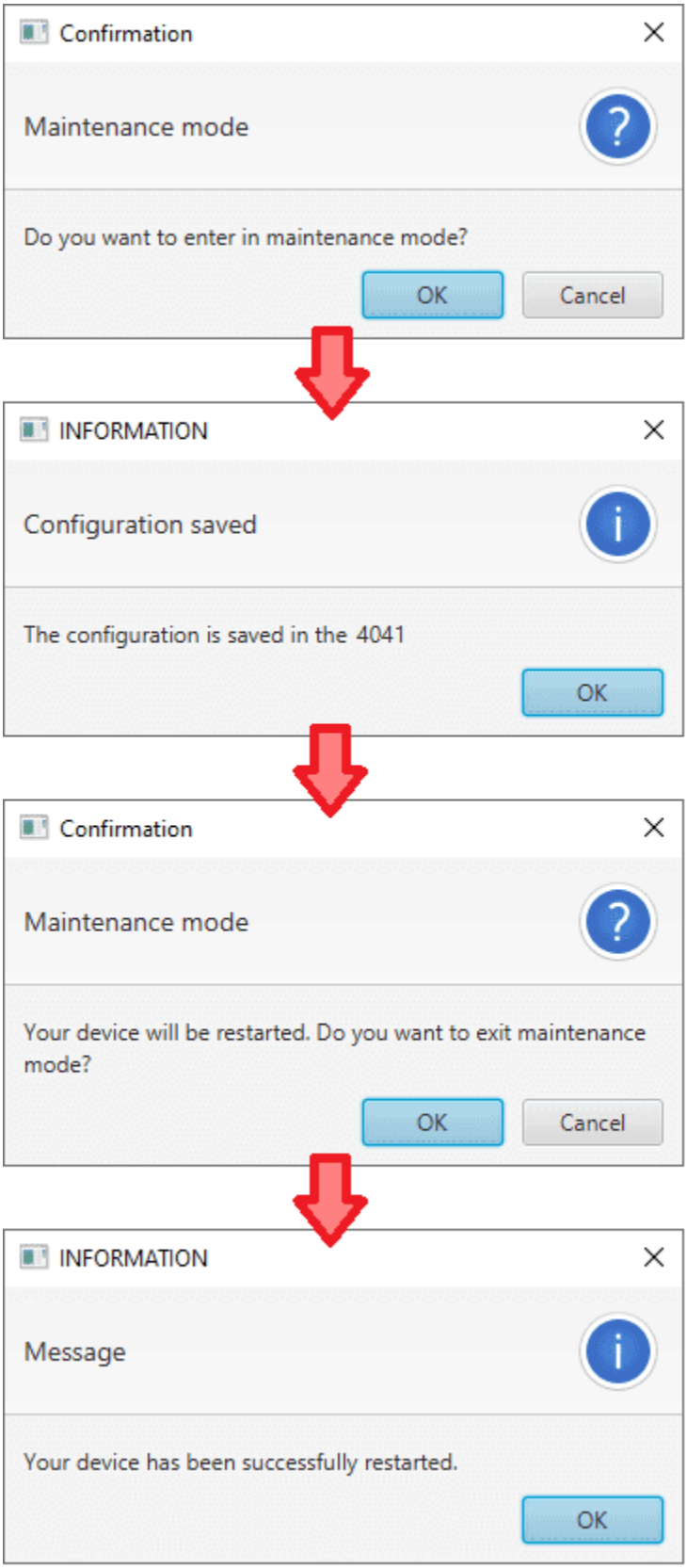


Fig. 10: **Save and close - Confirmation panels**

Danger: As Veronte Autopilot 1x is **reset**, it is **not advisable to save changes during flight tests**.

Note: This button will only appear if an Autopilot 1x is connected, i.e. when working offline this button will not be available.

2. **Export PDI**: After modifying a configuration, press the Export button to store the configuration in the local storage. Users can store this configuration in an empty folder or in the folder where the previously imported configuration is stored. With the latter option, the "original" configuration will be overwritten by the one with the new changes.

   The user can choose between:

   - **Download PDI**: With this option the 2 folders with the PDI files are downloaded.
   - **Download VER file**: Download a **.ver file** with the configuration in **binary**. This option is only available 'online', otherwise it will be disabled.

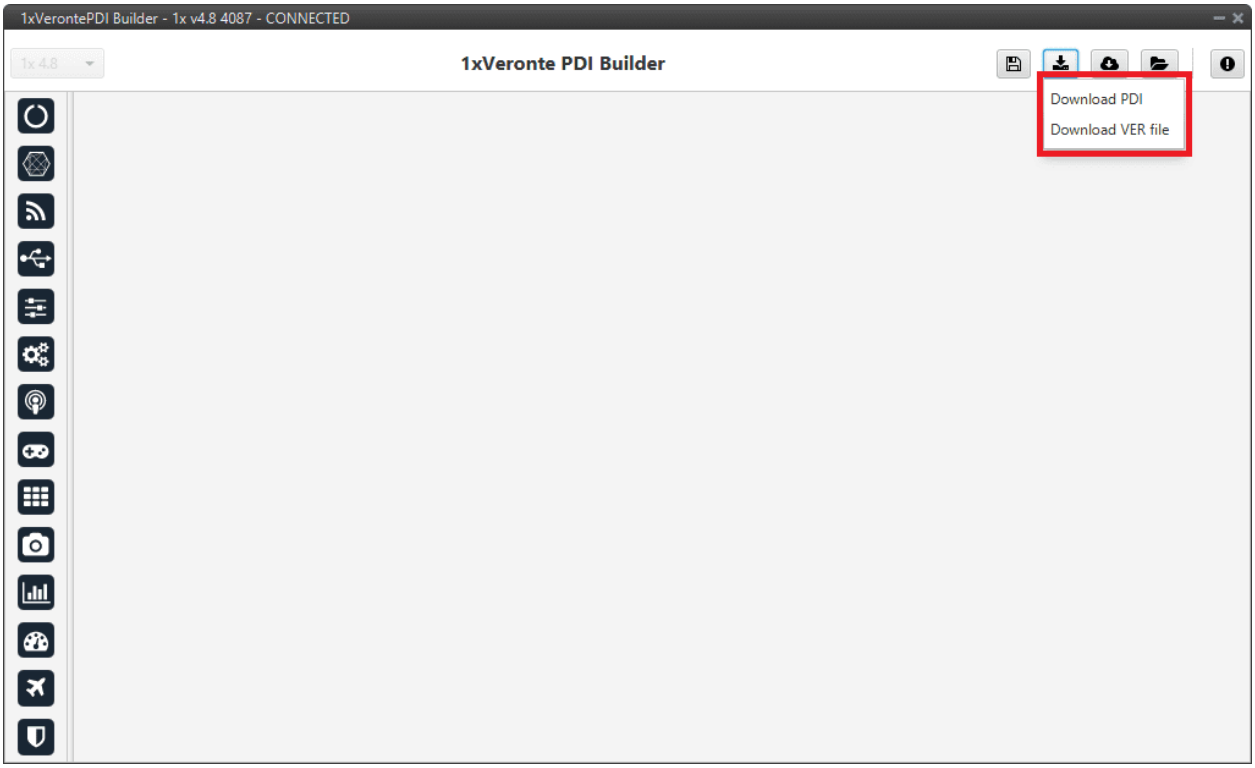


Fig. 11: **Download option**

3. **Import PDI from repo**: The user can import a configuration file from the GitHub repository and modify it. After that, if the **Save and close** button (1) is pressed, this configuration will be uploaded on the **1x**.
4. **Import PDI from local storage**: The user can import a configuration file from the local storage and modify it. After that, if the **Save and close** button (1) is pressed, this configuration will be loaded into the **1x**.

> **Warning:** If users want to upload a configuration in this way (either from the repo or from local storage), note that only the configuration 'setup' folder is uploaded, the 'operation' folder is not.
>
> This means that the operation of the new configuration uploaded to the **Autopilot 1x** will not be saved, but the previous operation will remain.

5. **Feedback**: Users can report a problem they have encountered by **creating an issue in their own 'Joint Collaboration Framework'**. The '**Download**' button downloads a zipped folder with the current **1x** configuration and more information needed for Embention to resolve the issue. It is advisable to attach this folder when creating the issue.

Note: The user's '**Joint Collaboration Framework**' is simply a **own Github repository for each customer**.

If the user has any questions about this Joint Collaboration Framework, please see Joint Collaboration Framework user manual or contact sales@embention.com.

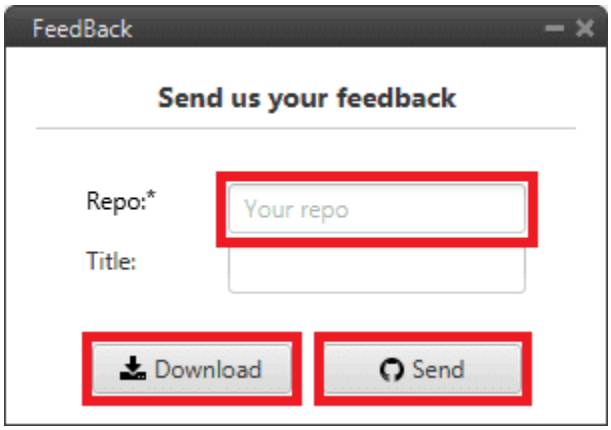


Fig. 12: **Feedback**

6. These are the different functions of Autopilot 1x. Each option will be explained in detail in the next sections.

Icon	Item	Description
	*Veronte*	Introduce Autopilot 1x information
	*Connections*	Configure I/O connections on Autopilot 1x
	*Sensors*	Configure parameter sensors
	*Input/Output*	Configure external sensors/devices and I/O signals
	*Control*	Introduce Phases, Modes and Arcade axis configuration
	*Automations*	Configure automatic actions on event detection (go home, change phase…)
	*Communications*	Configure alternative communication channels, statistics and routing
	*Stick*	Cusomize transmitter configuration
	*Block Programs*	Customize algorithms executed by Autopilot 1x
	*Devices*	Configure connected devices such as transponder/ADS-B, camera…
	*Telemetry*	Customize traffic such as log, telemetry…
	*UI*	Customize variable names
	*HIL*	Configure parameters for hardware in the loop simulations
	*Safety*	Customize checklist, block user control in PDI configuration and safety bits

2.1 Veronte

2.1.1 Unit name

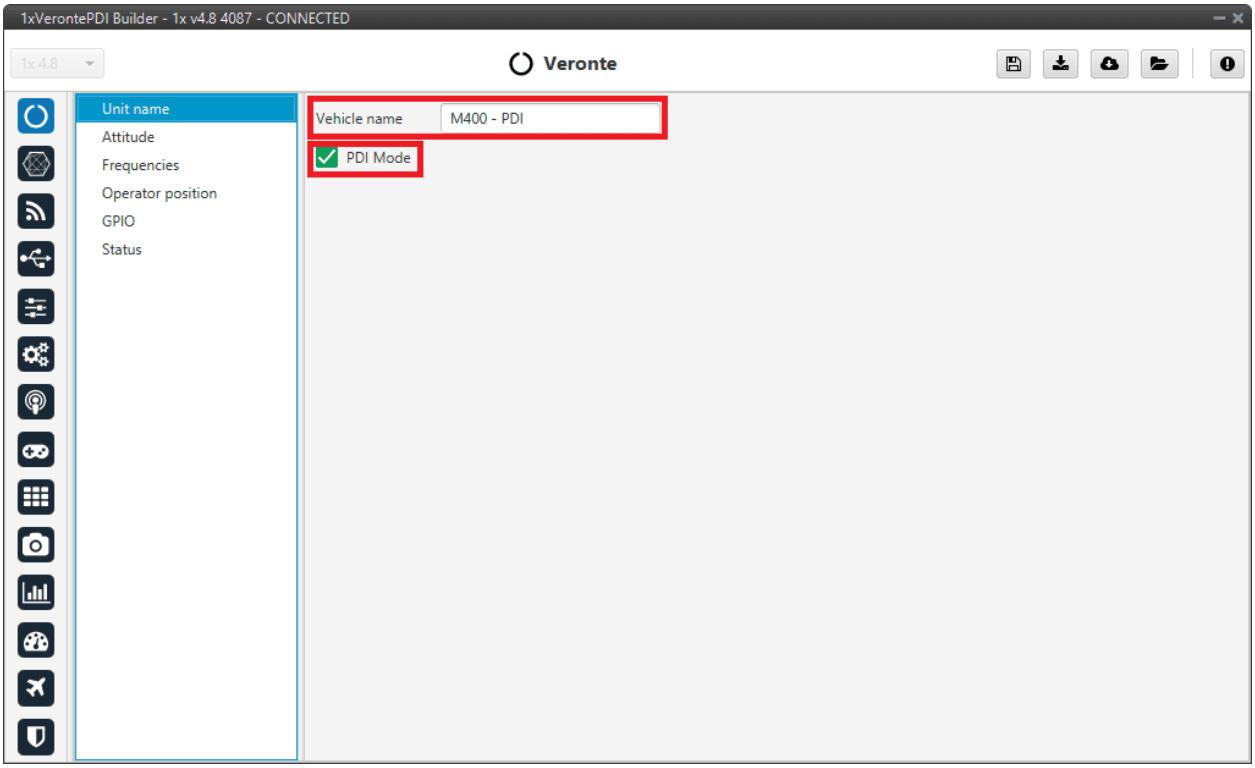


Fig. 13: **Unit name panel**

- **Vehicle name**: The user can define the name of the configuration.
- **PDI Mode**: It can be enabled or disabled. PDI mode allows the user to change the setup if Autopilot 1x is not in INI phase.

Warning:

- **Not** being in **PDI mode**, users cannot perform the following actions if they are not in INI phase:
  - Reboot Autopilot 1x
  - Change 1x setup (i.e. save to SD card)
  - Enter manually in maintenance mode
- The variable '**System error**' prevents operation in normal mode (**not PDI mode**). A list of all errors that can cause this bit to be set can be found in the Activation System Error bits section of the **1x Software Manual**.
- If Autopilot 1x has '**sensors errors**' and is in normal mode (**not PDI mode**), the user will not be able to switch to another flight phase, it will remain in **INI phase**.

Danger: PDI mode is intended for development purposes since, as detailed above, it **allows flight phase changes with system, sensor and PDI errors**.

It is highly recommended to limit its use to simulation and ground testing of peripherals during the development phase.

Therefore, as **it is not advisable to operate in PDI mode**, please disable it once the configuration is finished and intended to be used in flight.

2.1.2 Attitude

This menu allows the user to define the orientation of the autopilot with respect to the platform once it is installed. Aircraft axis are defined according to international aviation convention. Autopilot 1x axis are drawn on the autopilot's external case as defined in the Orientation - Hardware Installation section of the **1x Hardware Manual**.

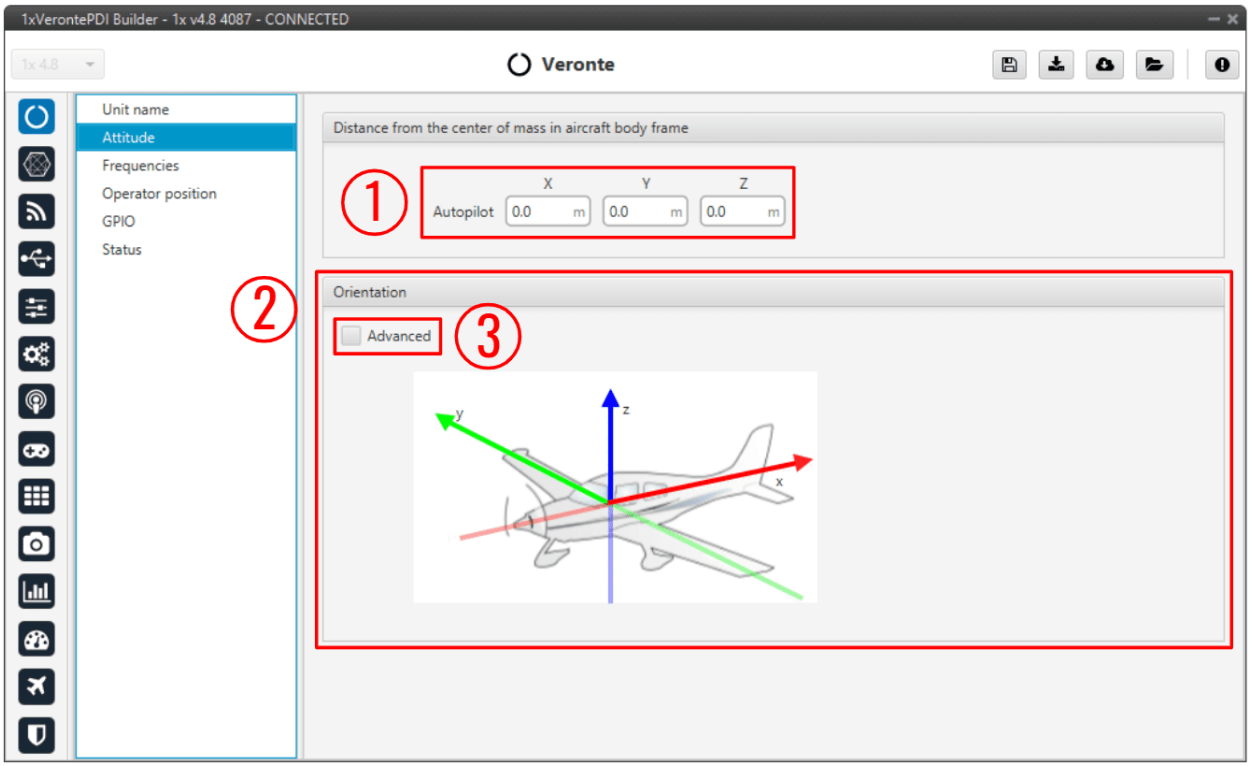


Fig. 14: **Attitude panel**

1. **Distance from the center of mass in aircraft body frame**: The autopilot's distance from the center of mass must be defined. This distance is entered in meters and accordingly to aircraft axis.

   If the autopilot is not located in the center of mass, it will measure a non-zero acceleration when turning. *Distance from the center of mass in aircraft body frame* is used to compensate this value. An example is presented below:

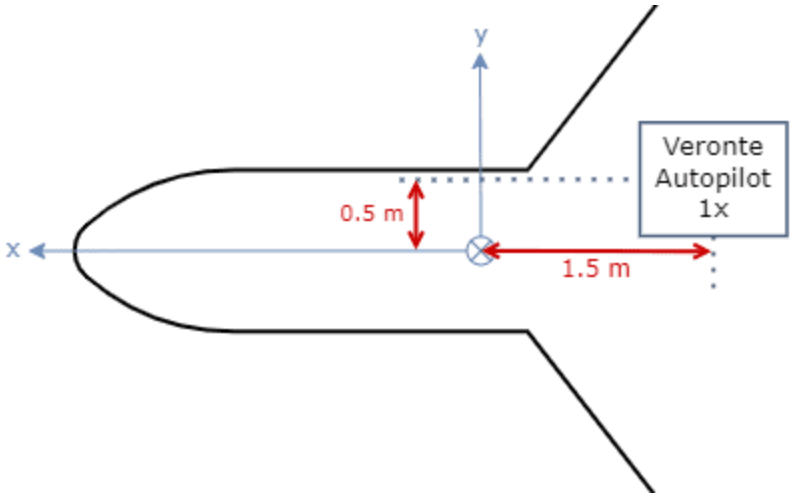


Fig. 15: **Distance from the center of mass in aircraft body frame - Example**

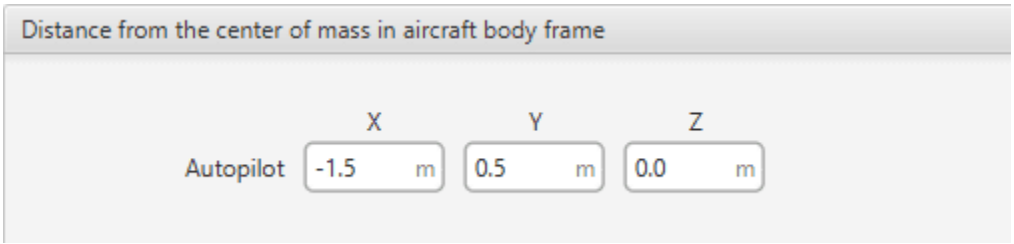


Fig. 16: **Distance from the center of mass in aircraft body frame - Example**

2. **Orientation**: It is not compulsory to install the autopilot aligned with the aircraft axis. In order to indicate the autopilot's relative position inside the platform, select the advanced option (**3**). A matrix relating vehicle axis and autopilot axis is needed to be filled in. The case of a non-orthogonal installation can be covered.

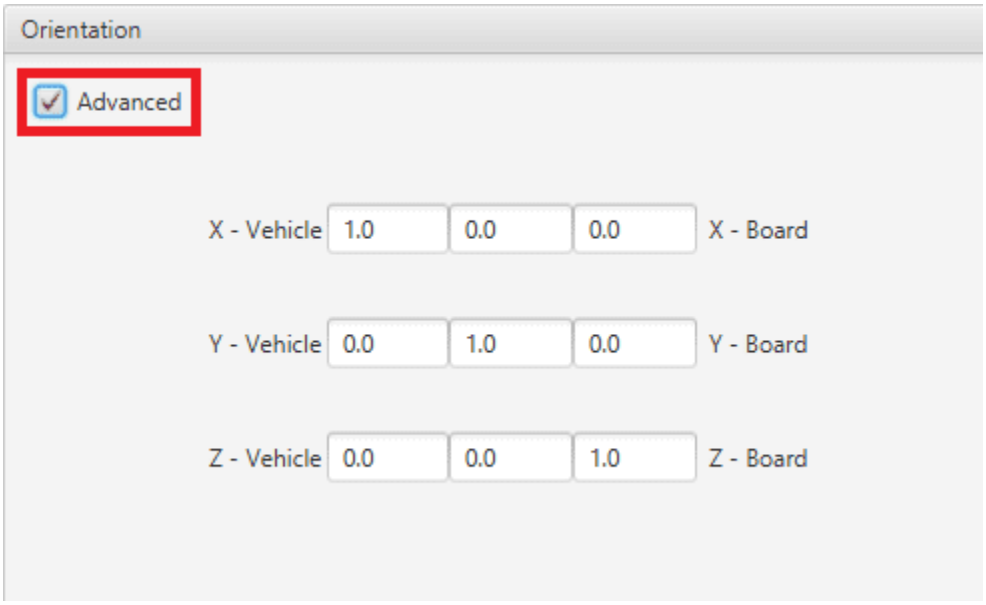


Fig. 17: **Advanced orientation**

Note: If only a simple rotation is required, for example, a -90º rotation in the Z axis, it is simpler to select the correct axis directly in the 'plane':

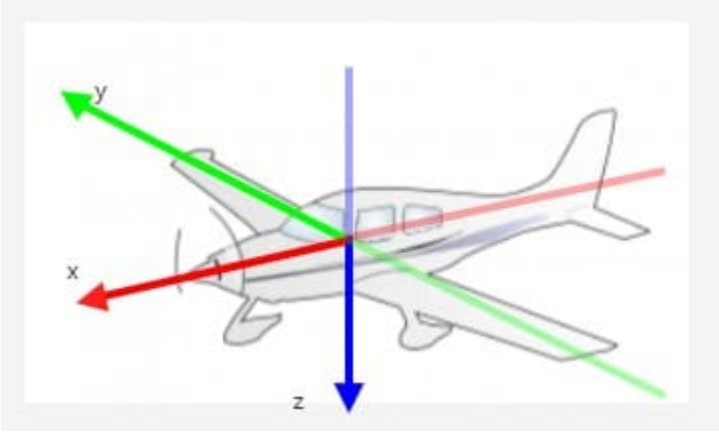


Fig. 18: **Change orientation**

2.1.3 Frequencies

The frequency of the GNC task refers to the maximum working frequency of the core. In this case, **400 Hz**, which is the **maximum possible**.

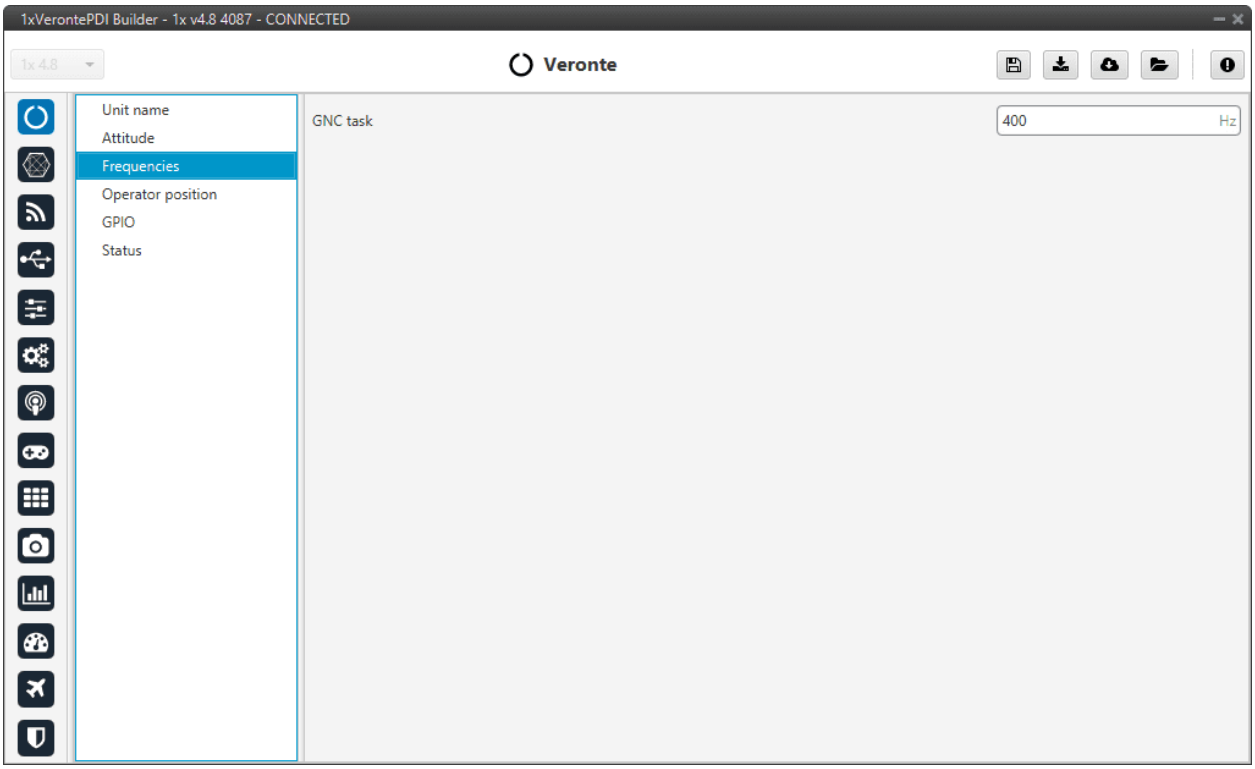


Fig. 19: **Frequencies panel**

Warning: 400 Hz can be used only for simple configurations, so it is often necessary to reduce the frequency to 250-300 Hz. To find out why the user should reduce the GNC Task frequency, see *Reduce GNC Task frequency - Troubleshooting* section of this manual.

2.1.4 Operator position

Veronte Autopilot 1x has a limited-operation firmware. The distance allowed by the license is calculated from the operator position.

For more information on this limitation, please refer to the Limited Operation Firmware - Quick Start section of the **1x Hardware Manual**.

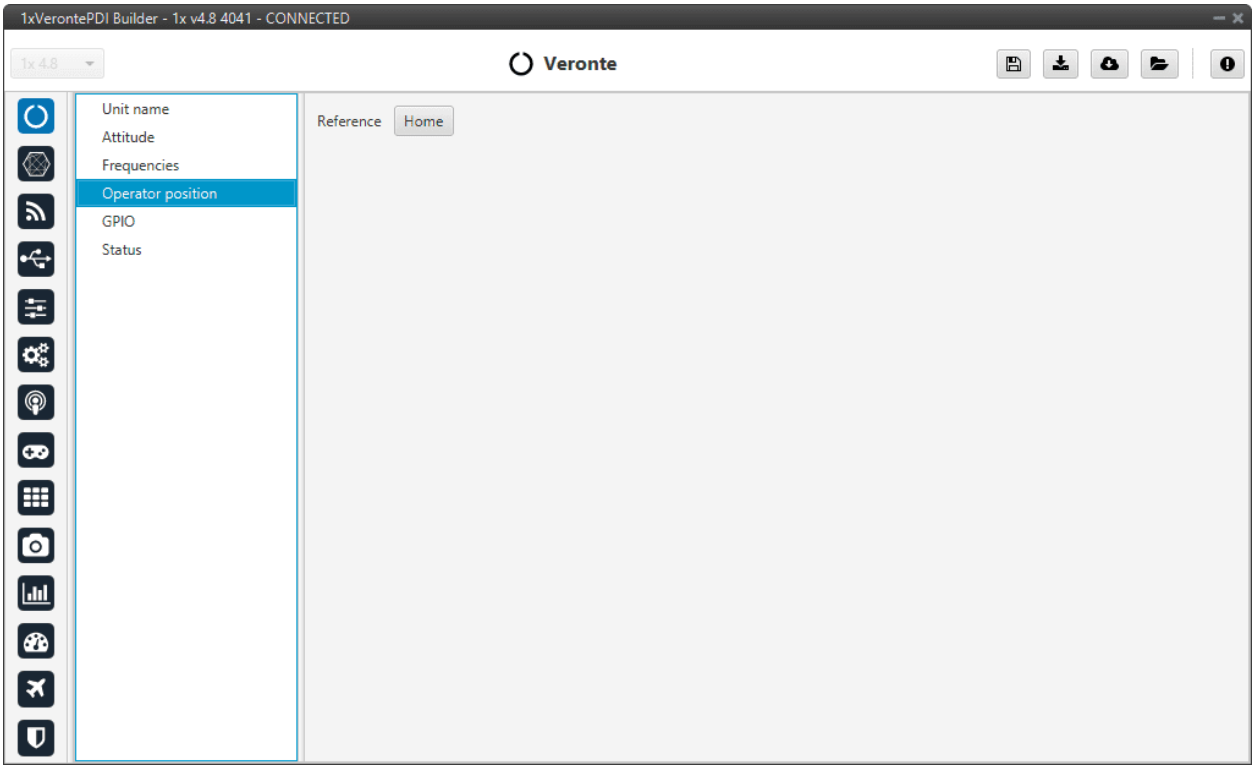


Fig. 20: **Operator position panel**

Tip: In **operation**, if the **air unit does not have the license activated**, it is recommended to set the position of the ground unit as the operator position in the air unit.

2.1.5 GPIO

In this tab each individual GPIO behavior can be configured:

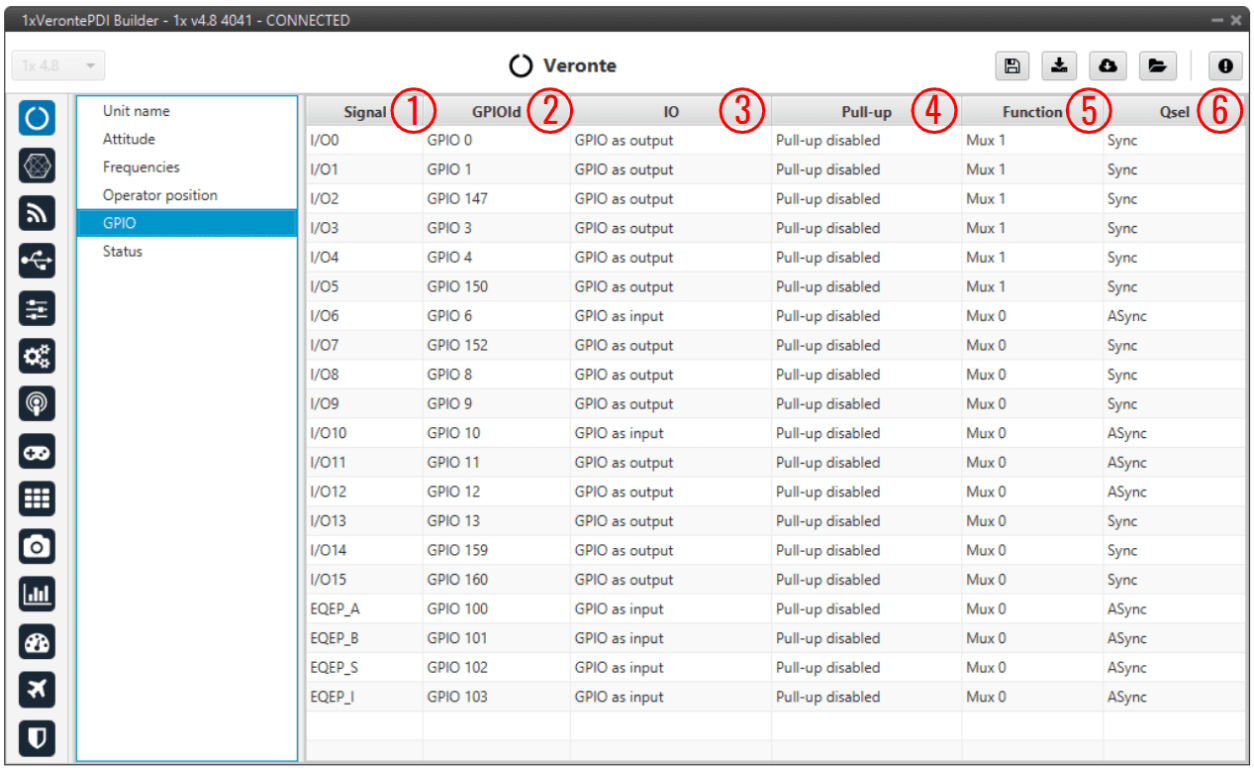


Fig. 21: **GPIO panel**

1. **Signal**: Pin ID as described in Pinout - Hardware Installation section of the **1x Hardware Manual**.
2. **GPIOId**: GPIO ID of the microcontroller.
3. **IO**: Define GPIO as an input or output.
4. **Pull-up**: Enable or disable the pull-up resistance.
5. **Function**: Mux 0: GPIO, Mux 1: PWM, Mux 2, Mux 3, etc. These are the different functionalities that the GPIO can have, this depends on the multiplexer.

Note: When users set **Function** to "Mux 1", it indicates that the corresponding pin is disabled as GPIO and enabled as PWM. Consequently, when **saving this change**, the corresponding PWM pin of the *PWM panel* should automatically be added as PWM and "disappear" as GPIO.

This behavior also happens the other way around, i.e. when switching from PWM (Mux 1) to GPIO (Mux 0).

Warning: **Once the change has been saved**, check that the corresponding PWM pin has been added to the *PWM panel* or that the corresponding GPIO pin has been added to the *GPIO panel*, depending on the change that has been made.

---

6. **Qsel**: This is the "input qualification", it is used to control how the value of a GPIO is evaluated. The available options are:
   - **Sync**: The value is taken as whatever is present at the time it is checked (synchronously). This is the default mode of all GPIO pins.
   - **3 Samples**: The value is checked 3 times and the value is only changed when the 3 times are the same.

- **6 Samples**: Same as before, but checking 6 times instead of 3.
- **ASync**: No checks are performed. It is used when it is not used as GPIO.

2.1.6 Status

This option enables the periodic sending of the status message that **Veronte Link** uses to recognize the Autopilot 1x.

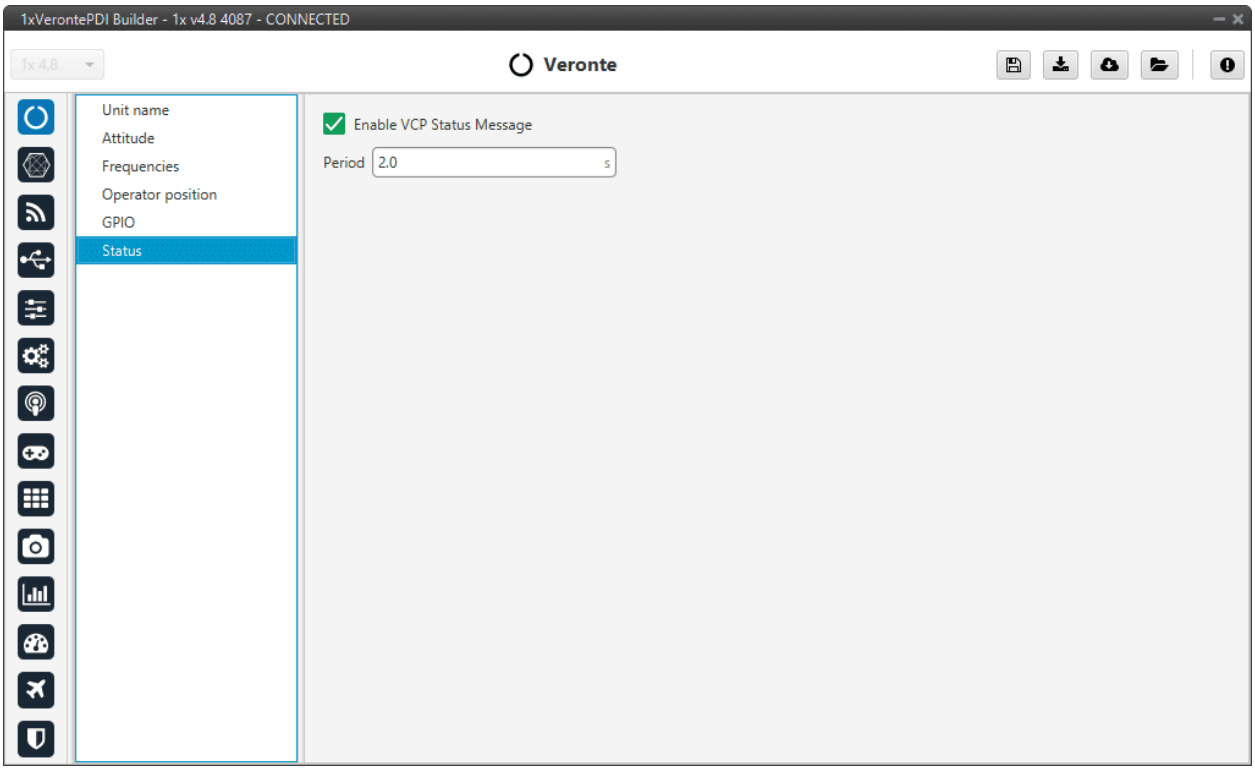


Fig. 22: **Status panel**

- **Period**: Enter a desired period to send repeatedly the status message.

Note: VCP is the Veronte Communication Protocol. To know more, read the VCP user manual.

2.2 Connections

Here the Input/Output ports of the autopilot can be configured. Depending on the configurable port selected, the user will need to provide different parameters.

Each connection is associated with a specific pin number. For more details see the Pinout - Hardware Installation section of the **1x Hardware Manual**.

2.2.1 ADC

ADC stands for Analog-to-Digital Converter. This connection is used by analog sensors. These sensors provide a voltage readout that needs to be converted into the actual measured variable, e.g. temperature, fuel volume, etc.

Autopilot 1x is equipped with 5 connections of this kind. Every ADC connection that is set requires an integer variable associated where the voltage readout will be stored. The **maximum voltage** of the ADC connection is **3.3 V**.

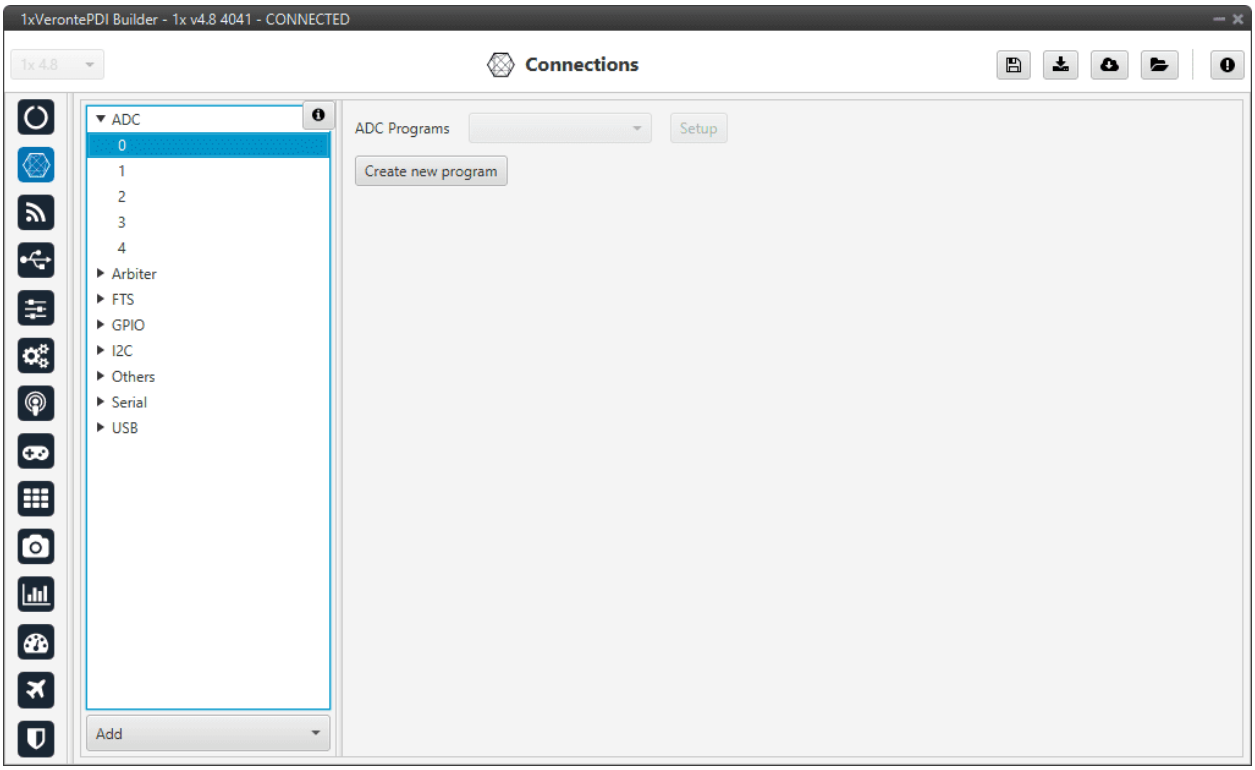


Fig. 23: **ADC panel**

To convert the input ADC value to the physical variable it represents, the user needs to create a new program. See more information about programs in the *Block Programs* section of this manual.

Application example

Let us consider a Fuel Level Sensor whose datasheet provides a direct relation of the voltage readout and the fuel volume (in L) through the polynomial $y = -0.0498x^4 + 0.3002x^3 - 0.3083x^2 + 1.2123x + 0.15$, where y is the fuel volume and x is the sensor voltage.

Creating a new program, the above equation can be reproduced. An example of how to do it is presented below.

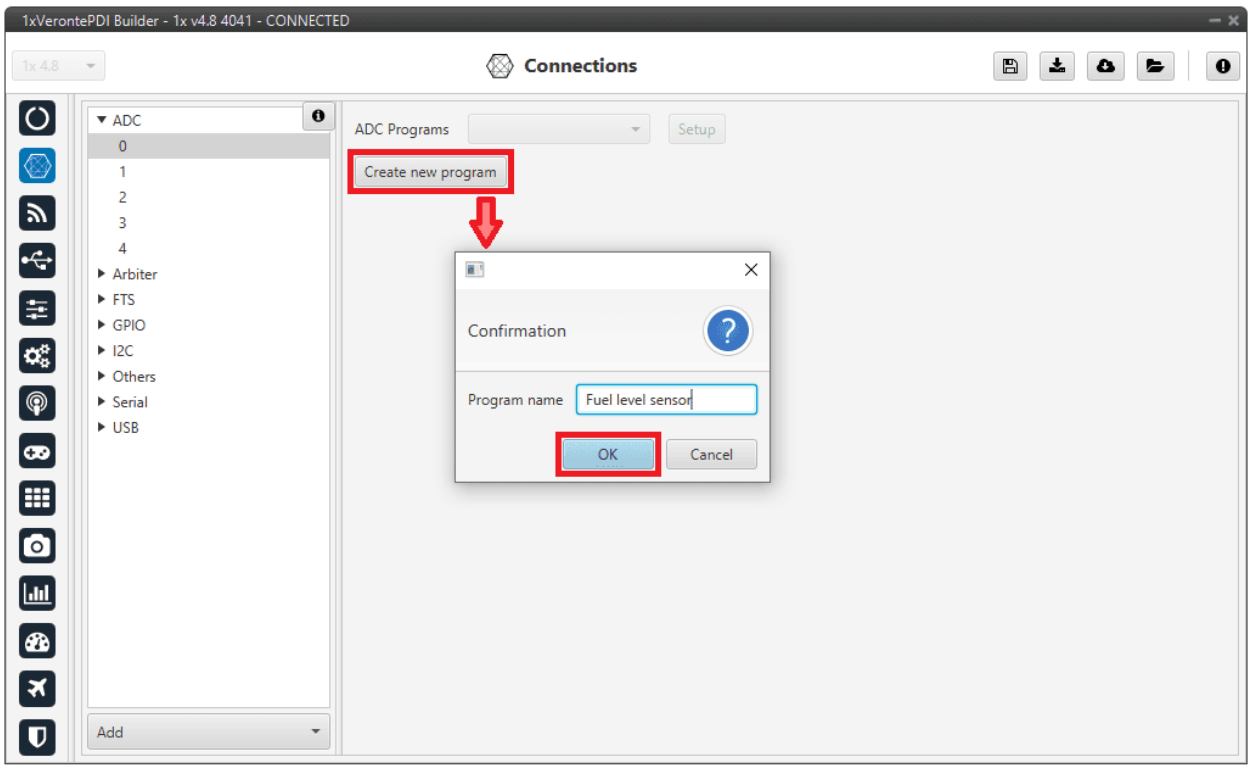


Fig. 24: **Create ADC program**

When the ADC program is created, a default block program is also created.

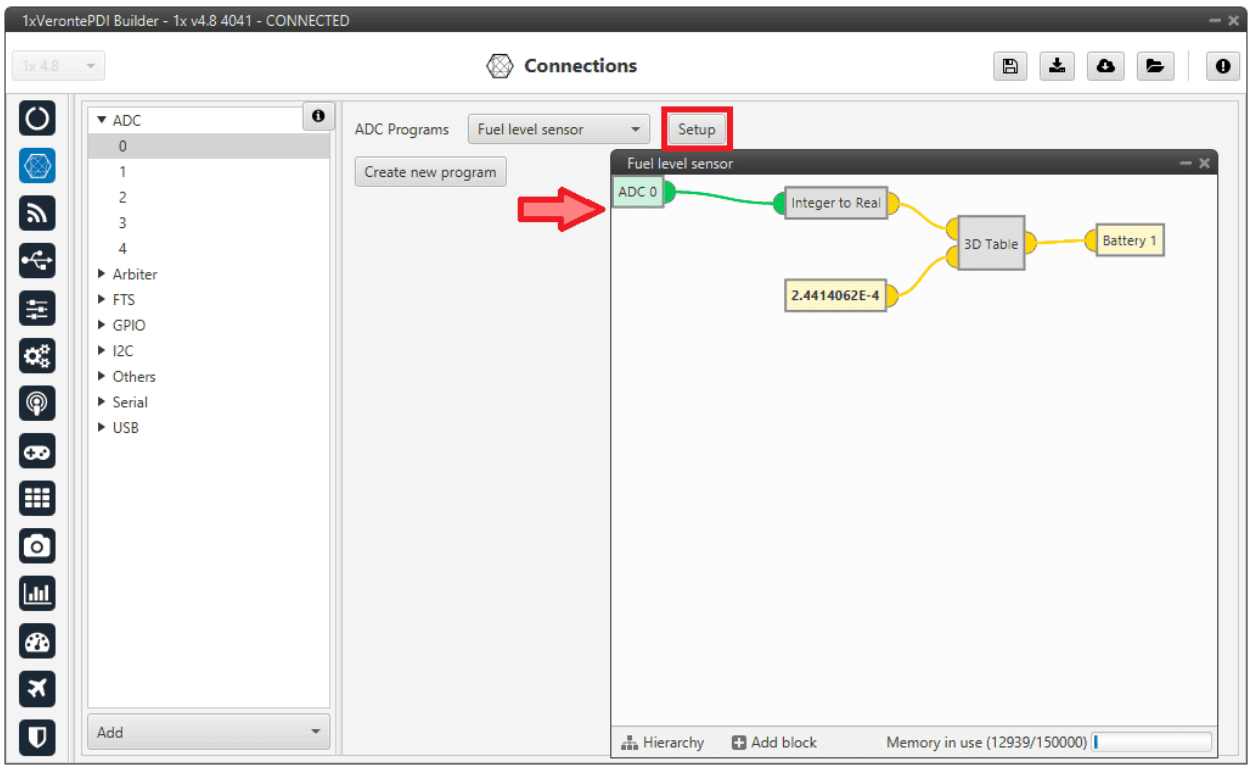


Fig. 25: **ADC default program**

Now, the program has to be customized for this application. See more information about programs in the *Block Programs* section of this manual.

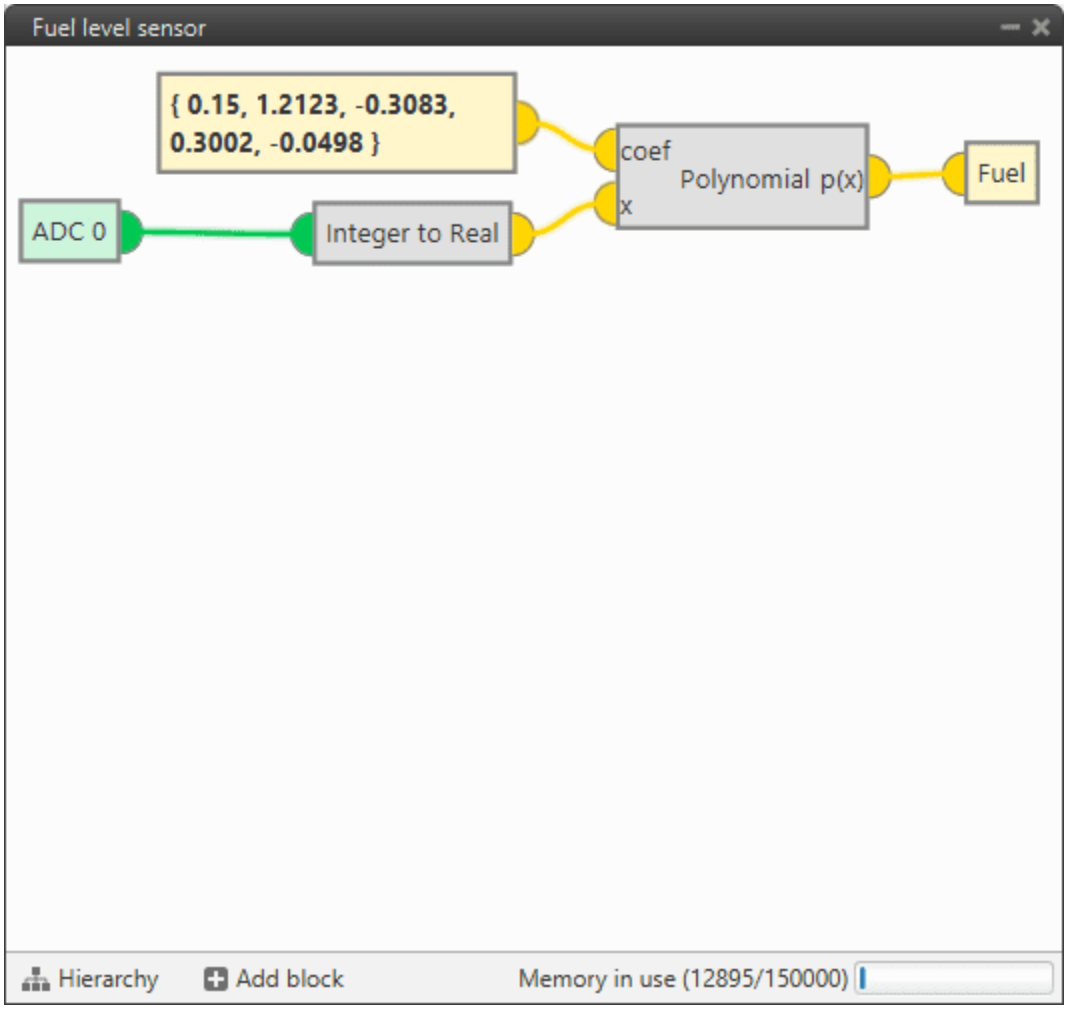


Fig. 26: **ADC conversion for a Fuel Level Sensor**

The fuel remaining in the tank is saved in a user variable, which can be used for displaying or warning purposes.

Note:

- The ADC variable is first converted from integer to real, and then the polynomial is applied.
- This program can now be modified by clicking in '**Setup**' in this menu or in the *Block Programs menu*.

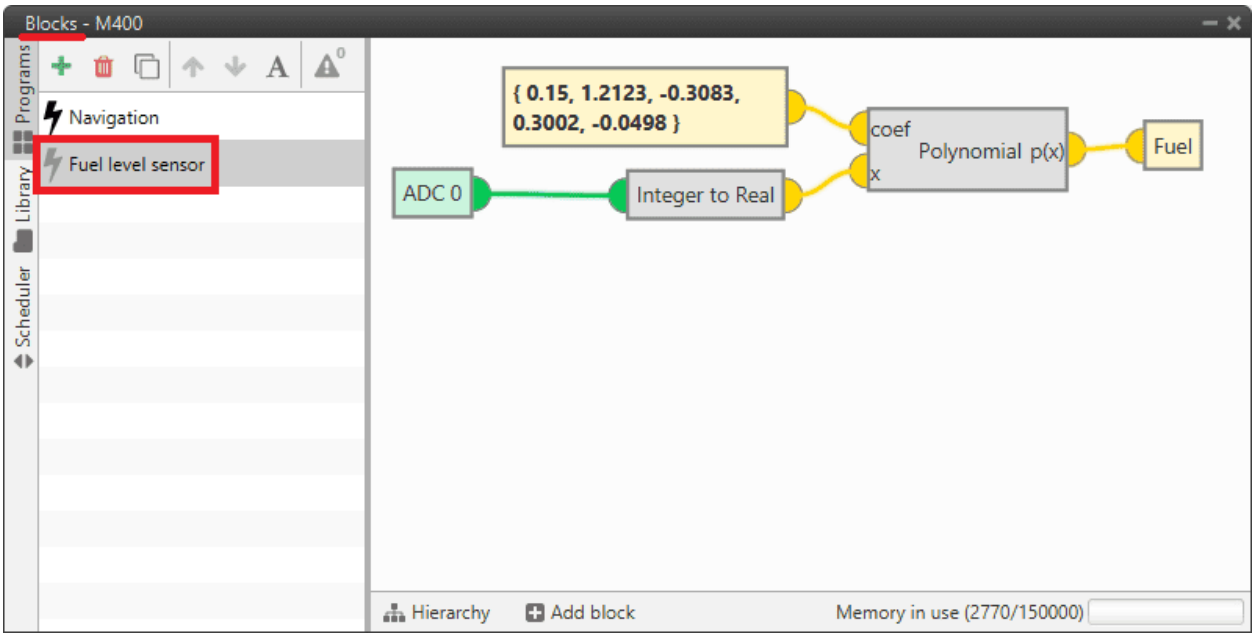


Fig. 27: ADC program in Block Programs

2.2.2 Arbiter

Pins 45 and 46 are dedicated pins to allow the UART communication with the Safety micro Controller (SuC). This microcontroller is in charge of monitoring the state of the main microcontroller and providing the Flight Termination Signals (FTS).

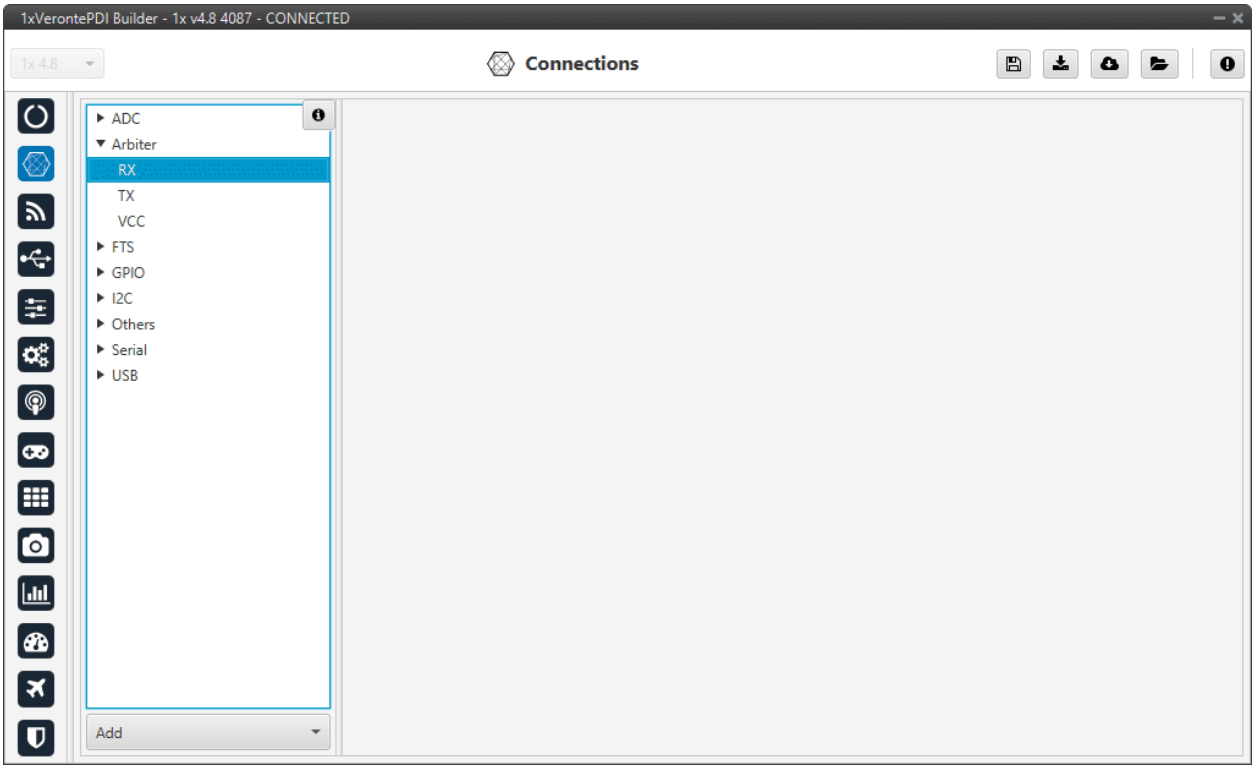


Fig. 28: **Arbiter panel**

2.2.3 FTS

The FTS (Flight Termination System) is a signal that is activated when a sytem error occurs (System Error bit is False). There is a group of bits that, when failed, cause the system error. To see more about the conditions that make the system error happen, go to Activation System Error bits section of the **1x Software Manual**.

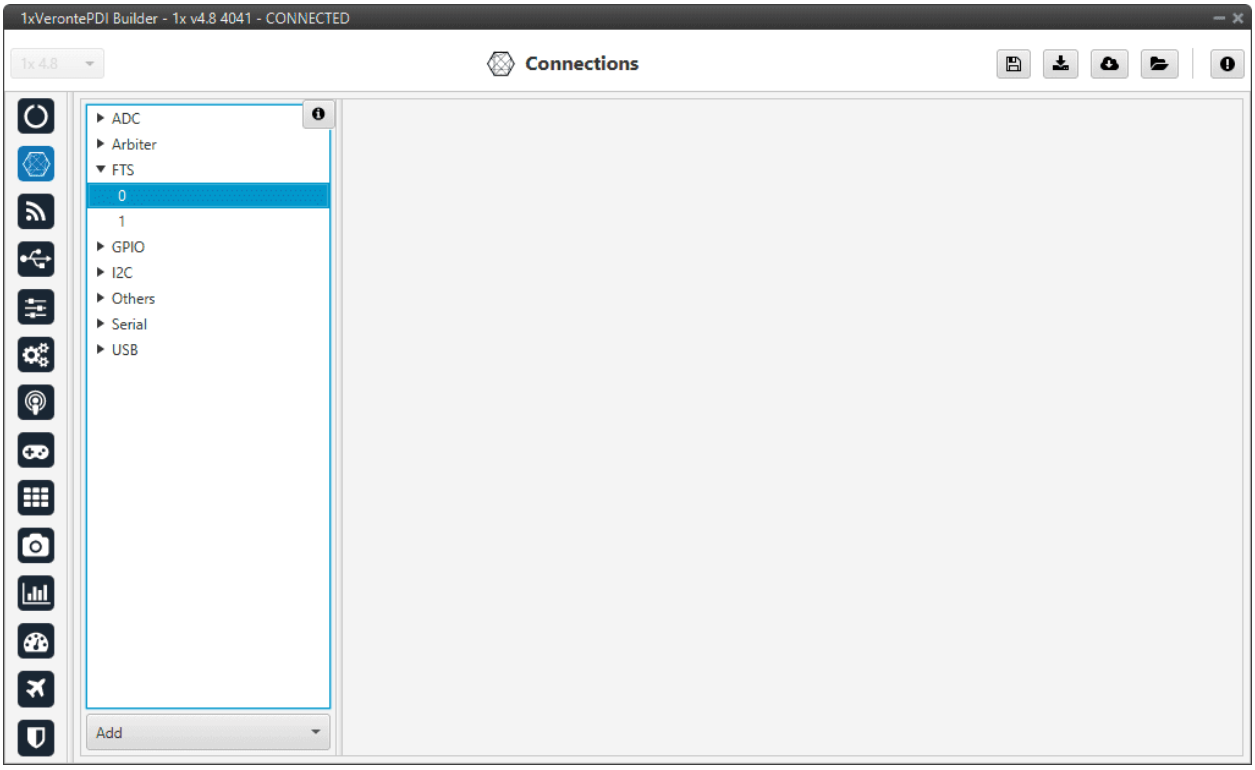


Fig. 29: **FTS panel**

Pins 42 and 43 are related to the FTS:

- **Pin 42**: Deadman signal from comicro, monitors main MPU encoding its product-level bit. This signal is a square wave at [100,125] Hz. It can be higher at rebooting (about 300-400Hz) but should never be less than 100Hz.
- **Pin 43**: !SystemOK Bit. 0 when Ok (no failure detected) and 1 (high, 3.3 V) when an error is detected. This pin goes high if the deadman signal sent by the MPU (main processor unit) is lower than 63Hz. That means there is a critical error.

2.2.4 GPIO

Output pins produce PWM or GPIO signals that are used to move the different servos and actuators of the platform.

A GPIO (General Purpose Input/Output) is a generic pin that can be configured as an input or output pin. When this option is configured as an output pin, the value sent will be different from the one sent if it was a PWM.

GPIO pins admit up to 4 different states:

- **ON**: A continuous signal of value 1, made by 3.3V.
- **OFF**: A continuous signal of value 0, made by Ground.
- **PULSE ON**: A single pulse of value 1, with a width specified in seconds.
- **PULSE OFF**: A single pulse of value 0, with a width specified in seconds.

The configuration of the pin output value is done with an *Output action* in the **Automations menu**.

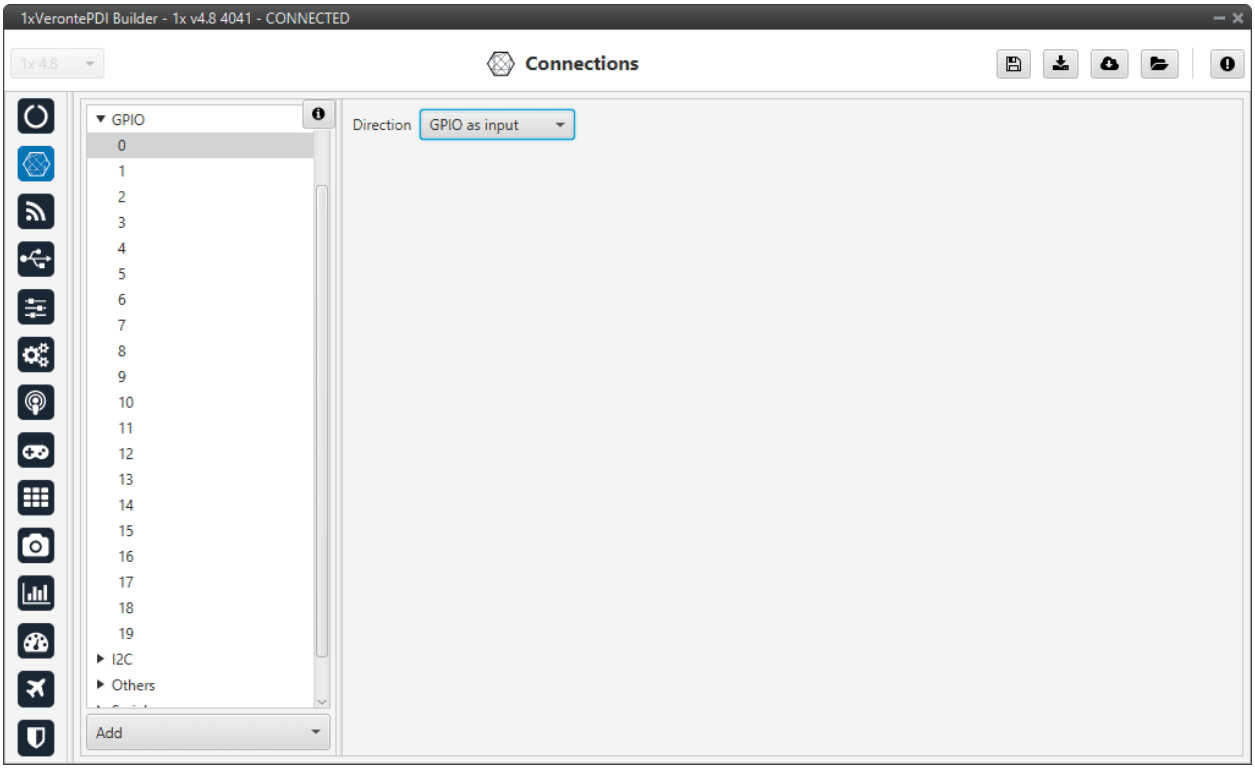
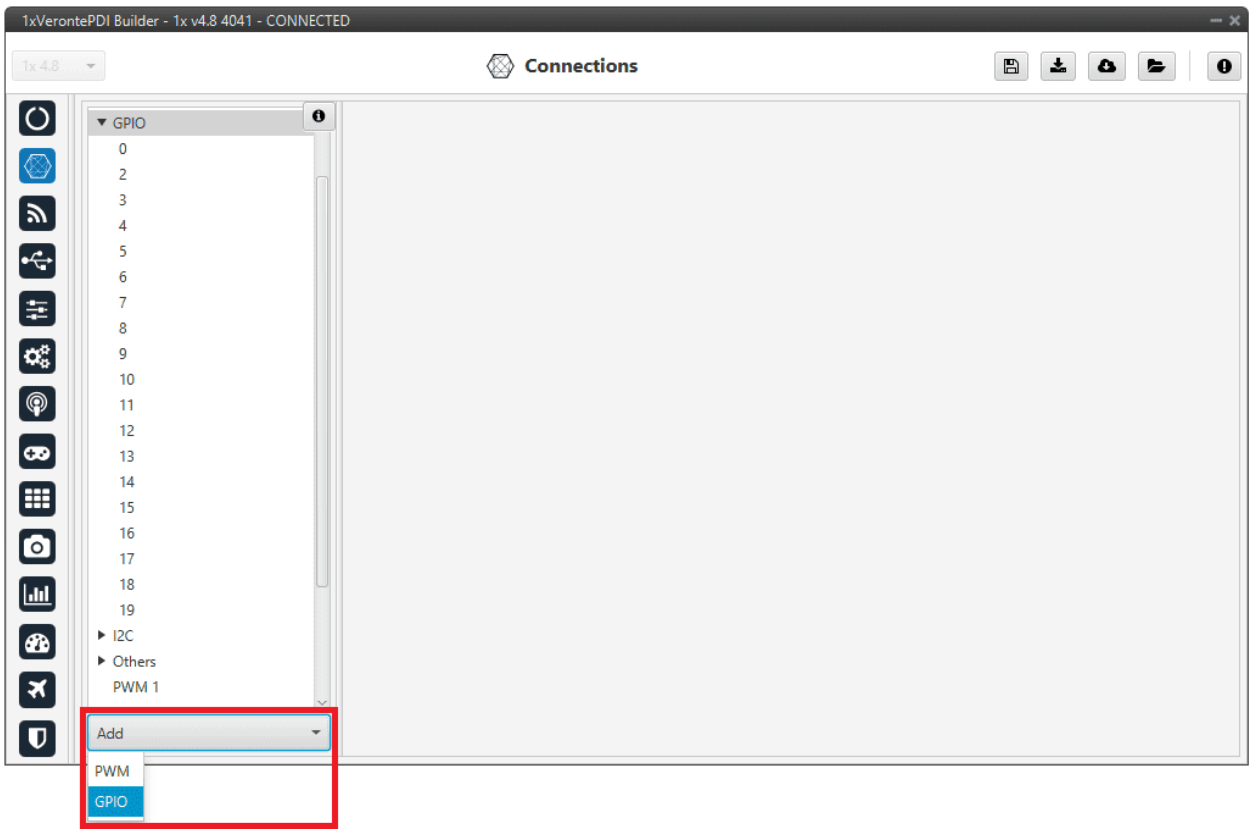


Fig. 30: **GPIO panel**

- **Direction**: Defines GPIO as an input or output.

Autopilot 1x admits up to 20 I/O PWM/GPIO signals. To configure a pin as GPIO after it has been changed to PWM, click *Add* and select GPIO:



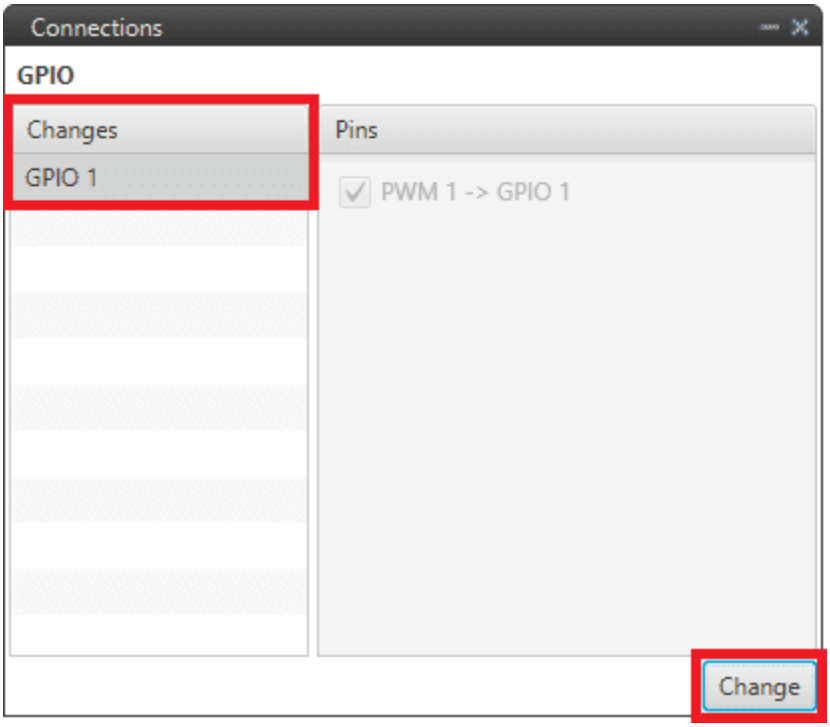


Fig. 31: **Add GPIO**

Note: When a pin is added as GPIO, it is disabled as PWM and enabled as GPIO. Hence, in the *GPIO panel* of **Veronte menu**, the "Function" parameter shall change to "**Mux 0**".

2.2.5 I2C

I2C stands for Inter-Integrated Circuit bus. It is a bus interface connection protocol incorporated into devices for serial communication. It operates in 2 modes: master and slave.

I2C uses only 2 bi-directional open-drain lines for data communication called **SDA** and **SCL**. Both these lines are pulled high.

- **Pin 31 - SCL**: Clock line for I2C bus (0.3V to 3.3V), it carries the clock signal.
- **Pin 32 - SDA**: Data line for I2C bus (0.3V to 3.3V), transfer of data takes place through this pin.

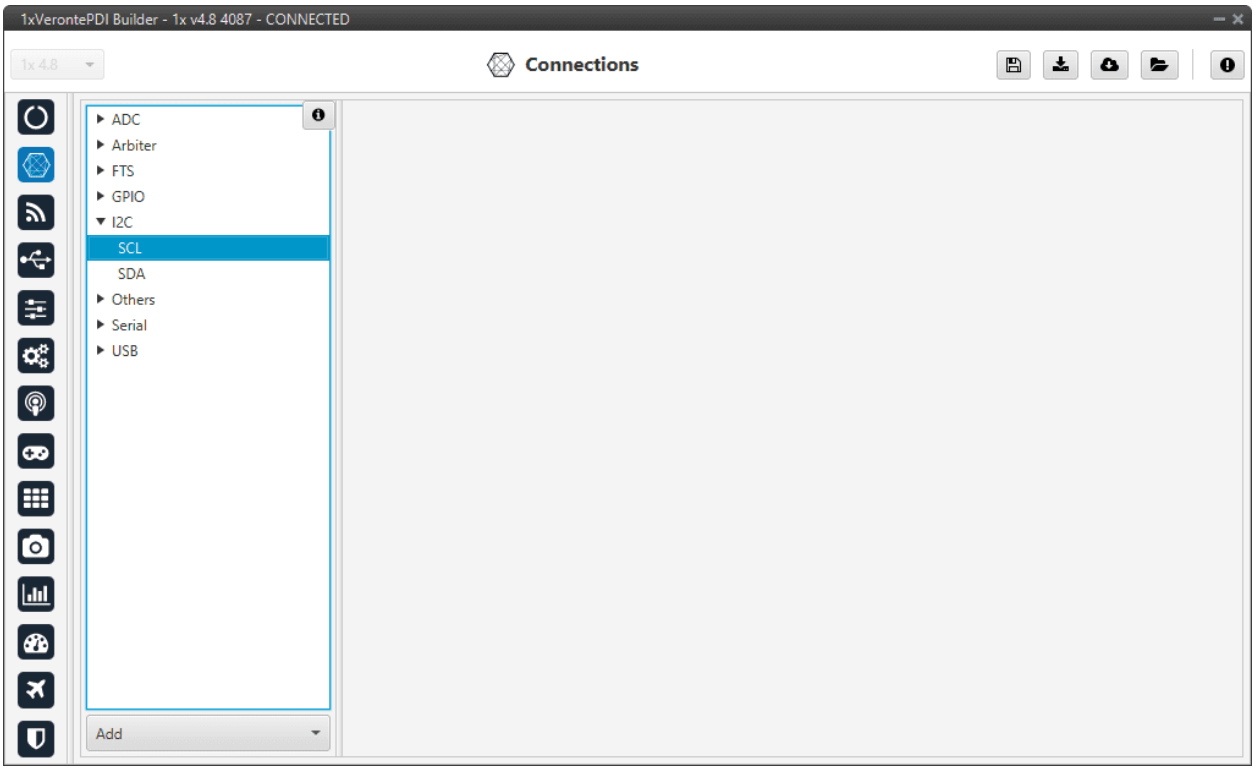


Fig. 32: **I2C panel**

Tip: These pins are very useful for **forcing maintenance mode**. For more information, refer to the Using the I2C pins to enter in maintenance mode - Troubleshooting section of the **1x Hardware Manual**.

2.2.6 Others

- GND: Ground.
- Power.

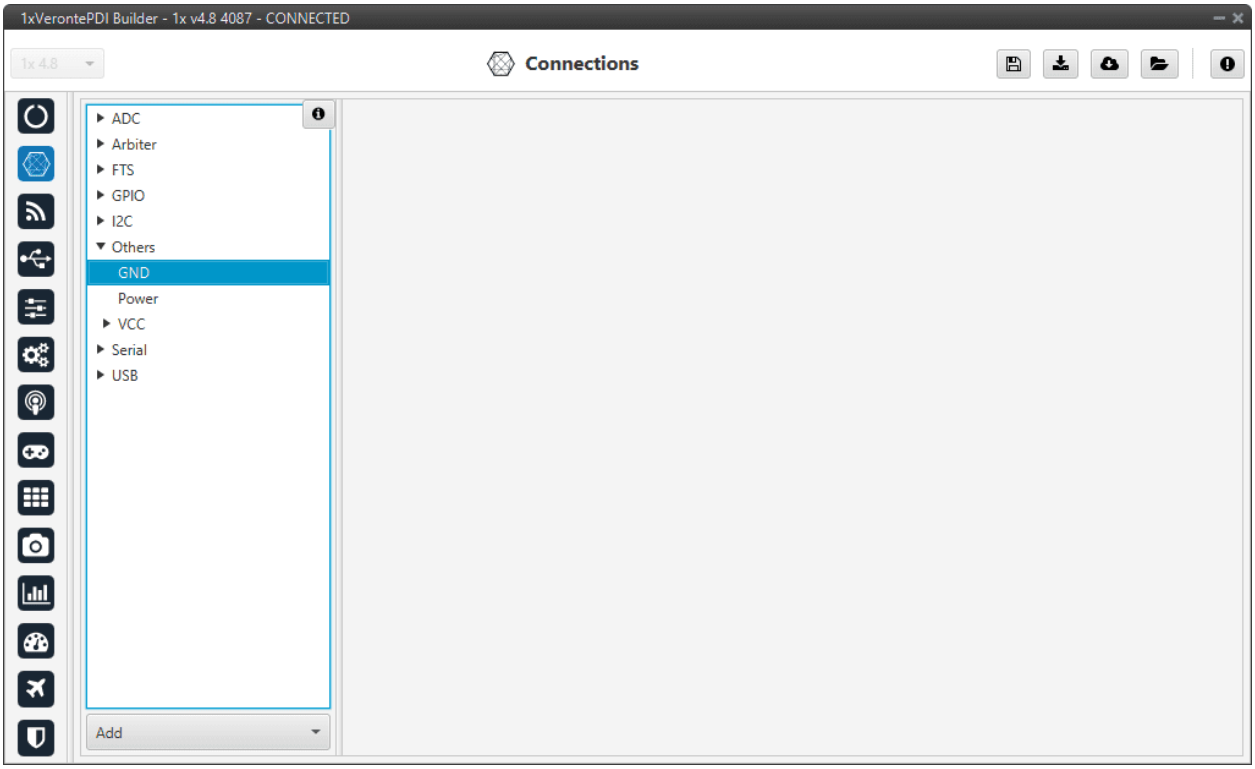


Fig. 33: **Others panel**

2.2.7 PWM

Output pins produce PWM or GPIO signals that are used to move the different servos and actuators of the platform.

The acronym PWM corresponds to Pulse Width Modulation. 1x sends a pulse with a certain width that is received by the servo/actuator, and according to the width of such pulse, it changes its behavior. A wide pulse will correspond to a big movement and a narrow one to a small movement.

By default, all PWM/GPIO pins are configured as GPIO output. So, to configure them as PWM, it is necessary to click *Add*:

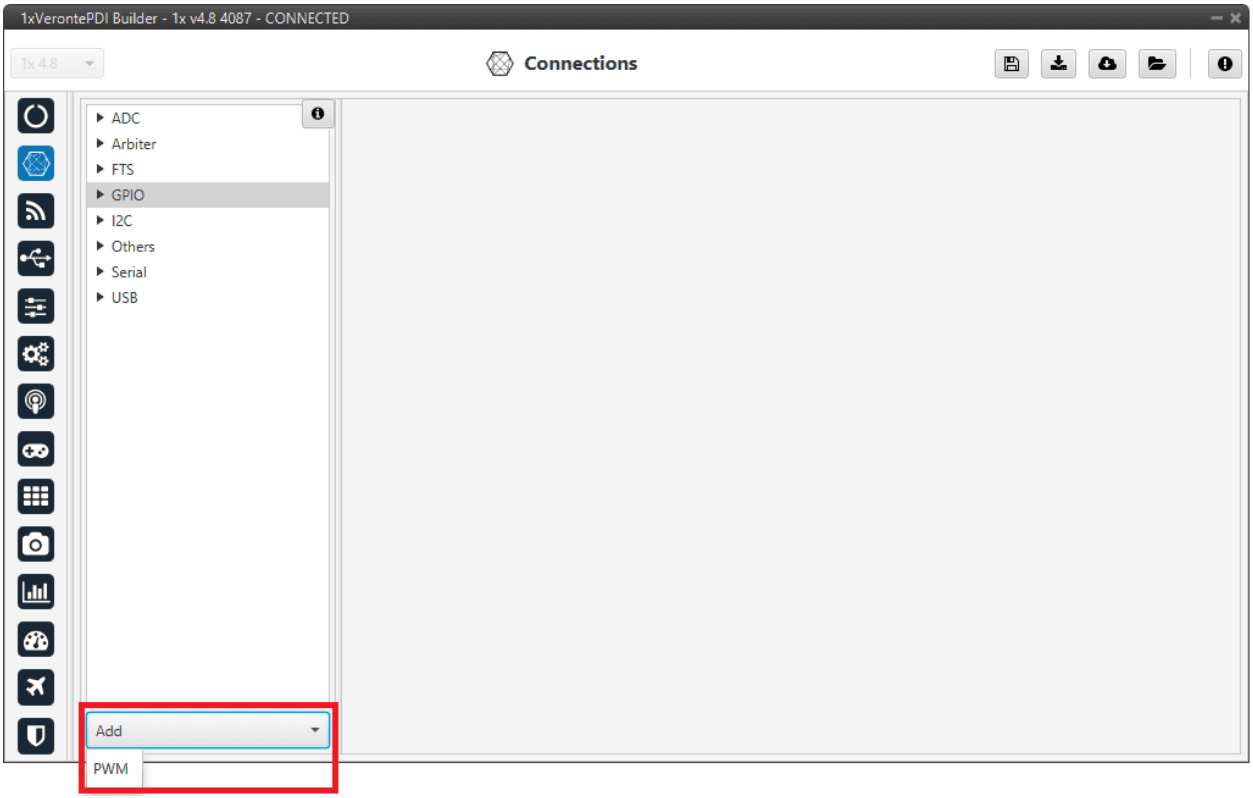


Fig. 34: **Add PWM**

Then, select the GPIO pin the user want to change to PWM. As can be seen, pins are interchangeable.

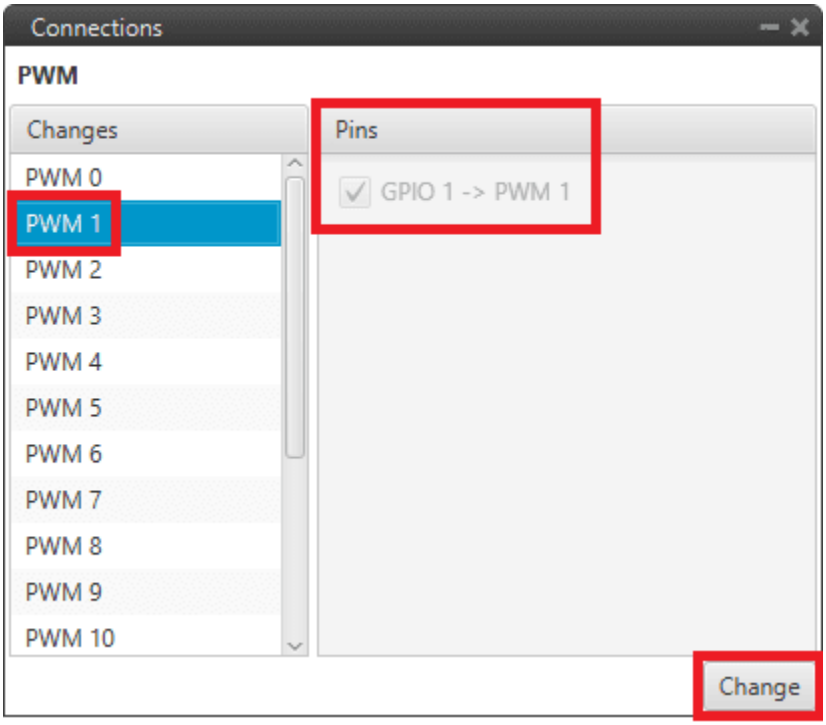


Fig. 35: **PWM change**

Note: When a pin is added as PWM, it is disabled as GPIO and enabled as PWM. Hence, in the *GPIO panel* of **Veronte menu**, the "Function" parameter shall change to "**Mux 1**".

As shown in the image below, the GPIO 1 output is now missing as it has been changed to a PWM output.

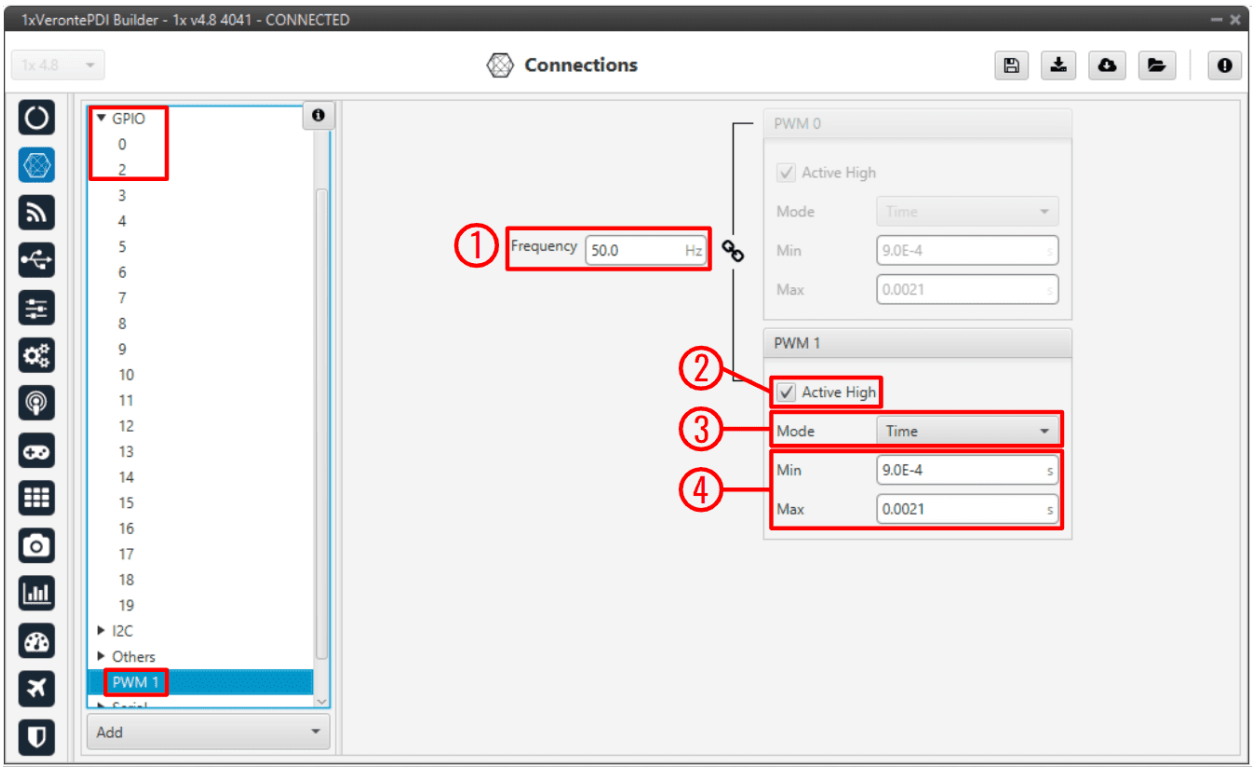


Fig. 36: **PWM panel**

In this menu the following parameters can be configured:

1. **Frequency**: PWM output frequency. This option determines the period of the pulses sent by the autopilot. The PWM is built in pairs inside the autopilot, and that is why the frequency is indicated in pairs, i.e when the frequency of PWM 0 is changed, the one of PWM 1 also changes. The following table shows the PMW pairs as configured in Autopilot 1x.

PWM 0	PWM 1
PWM 2	PWM 3
PWM 4	PWM 5
PWM 6	PWM 7
PWM 8	PWM 9
PWM 10	PWM 11
PWM 12	PWM 13
PWM 14	PWM 15
PWM 16	PWM 17
PWM 18	PWM 19

2. **Active High**: Polarity high or low (high if enabled).
3. **Mode**: The available options are **Time** and **Duty cycle**.
    - **Time**: The values indicated in **Min** and **Max** parameters are expressed in time units.

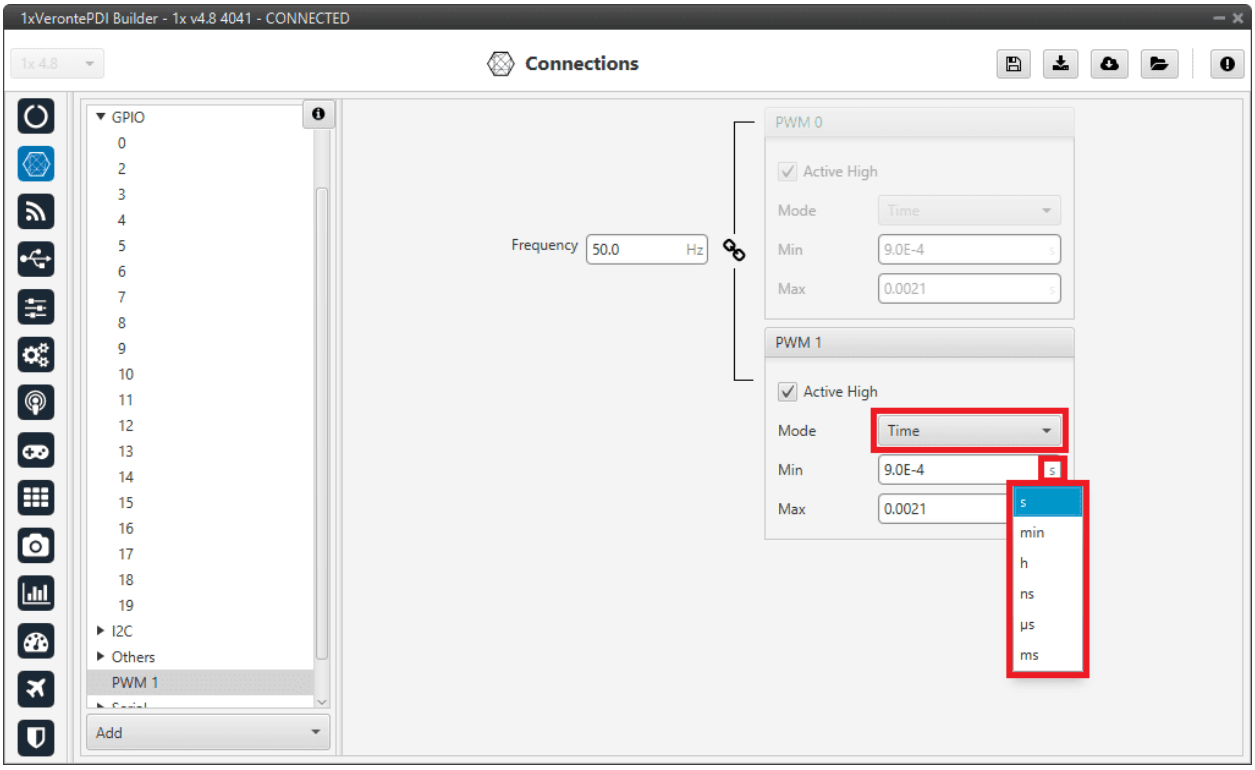


Fig. 37: **PWM panel - Time units**

- **Duty cycle**: This option is a different way of indicating the pulse width. Now the value indicated in **Min** and **Max** parameters is a percentage which corresponds to the relation between the pulse width over the total period of the sent signal.

  So a 100% duty cycle will correspond to a signal with a constant value of 1, while a 0% duty cycle implies a constant signal with value 0. Between this two extremes, the pulse width can vary as in the examples shown in the following figure.

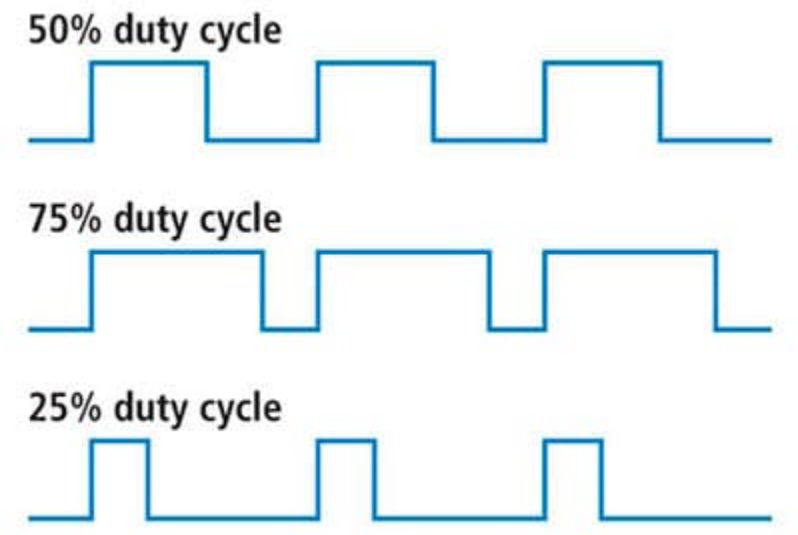


Fig. 38: **Duty cycle**

Note: Duty cycle percentages can be expressed in percent and per unit.

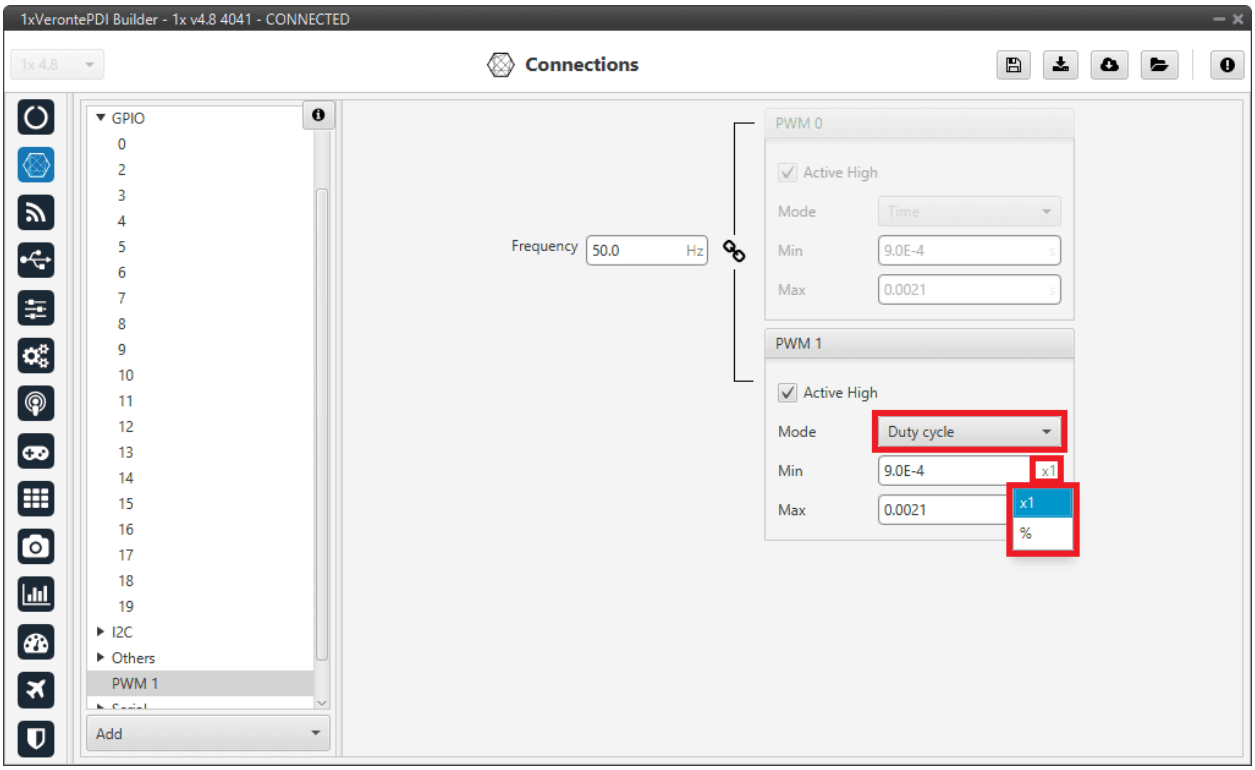


Fig. 39: **PWM panel - Duty cycle units**

4. **Min/Max**: These parameters are the pulse width values that will make the servo/actuator go to its **lowest/highest position**. It will be the output when the PWM message specifies **0/4095**.

   As an example let's consider the servo of an aircraft elevator, a pulse sent by Autopilot 1x of 0.9 ms will correspond with the lowest point of the servo range (-30 degrees for example). On the other hand, a pulse of 2.1 ms will make the servo go to its top position (for example 30 degrees).

Summary

A PWM is a signal which consists of a series of pulses having a width determined by a percentage over a range specified by the parameters Min and Max. On the other hand, the GPIO is a signal with a constant value (1,0) or with a single pulse (1,0).

2.2.8 Serial

Two serial interfaces are available with Autopilot 1x, 1 port RS-232 and 1 port RS-485, however more can be added by using a CEX or MEX. Each one of the serial interfaces is associated with a set of pins. To configure these serial ports, go to *Serial panel* in the **Input/Output menu**.

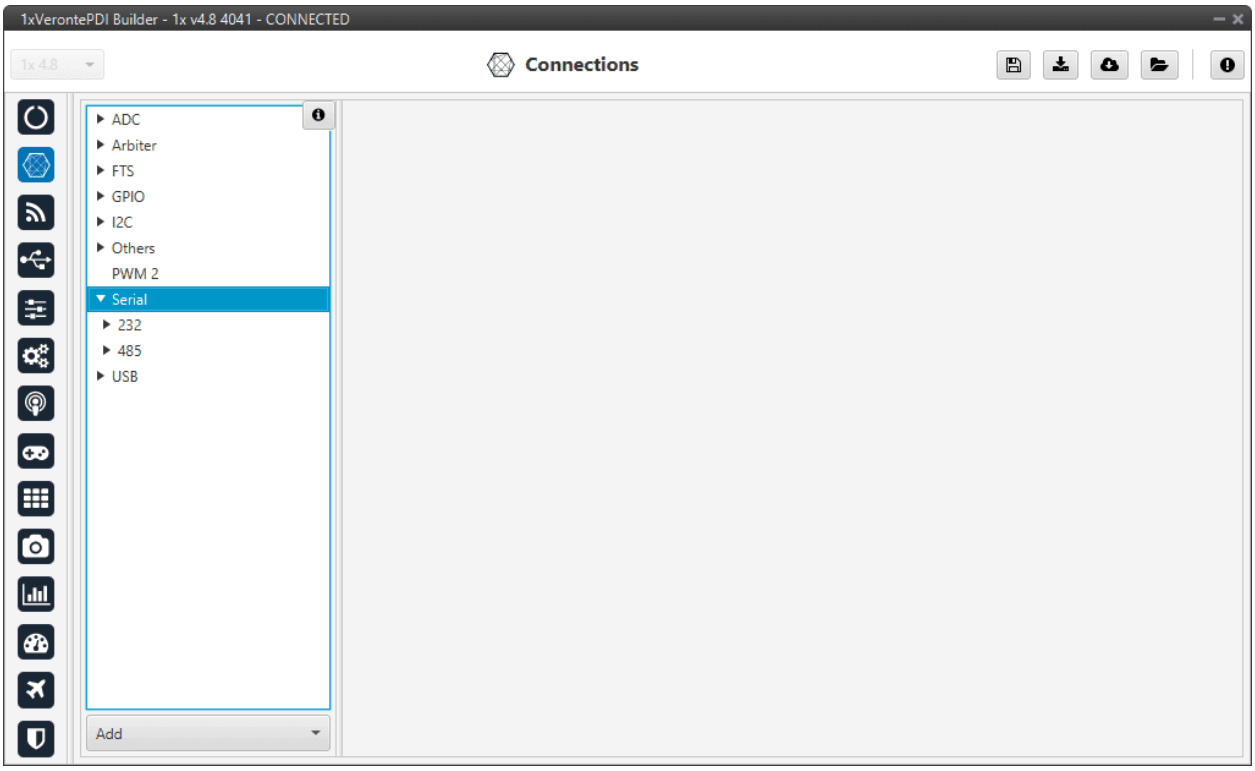


Fig. 40: **Serial panel**

2.2.9 USB

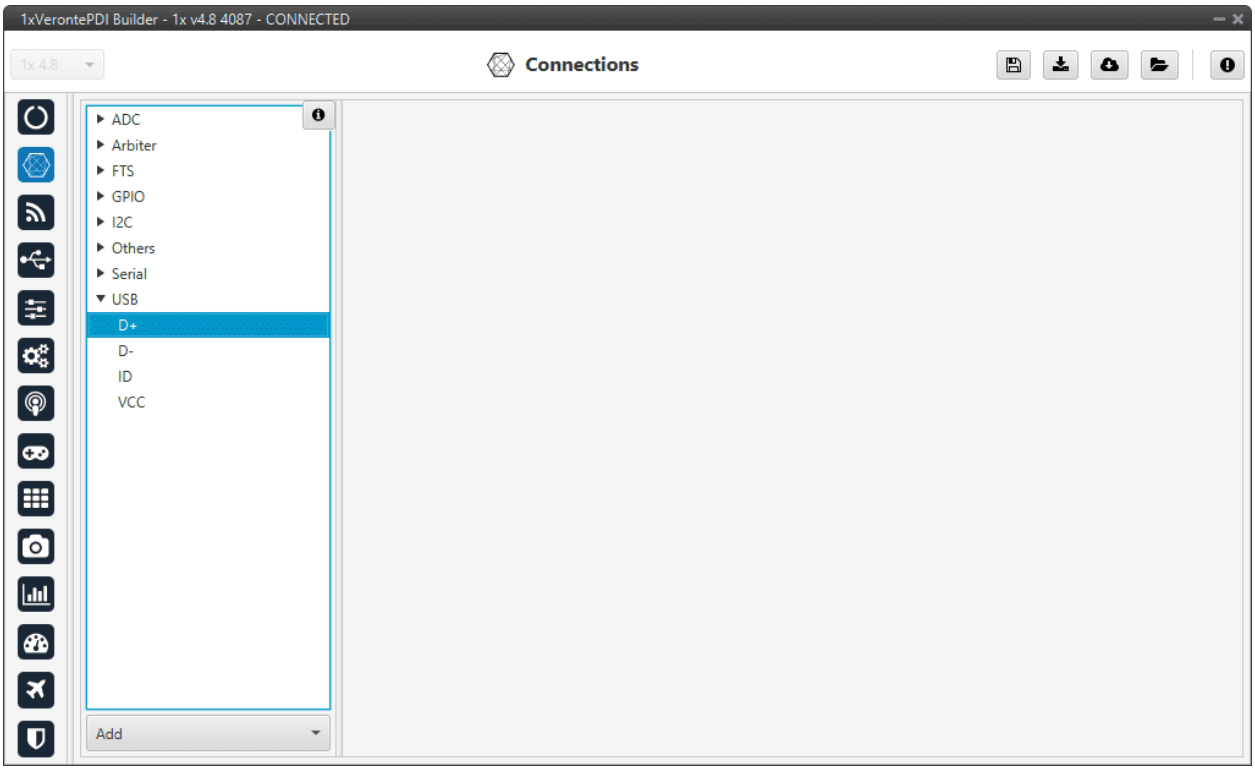


Fig. 41: **USB panel**

2.3 Sensors

Sensors menu allows to configure any external or internal sensor of Veronte Autopilot 1x.

2.3.1 Accelerometer

Veronte Autopilot 1x incorporates 3 Inertial Measurement Units (IMUs) that allow the 1x system to measure different variables and that are the main navigation data source. In addition, external IMUs can also be used for this purpose.

From the IMU, the user can configure the accelerometer and gyroscope. The first one is explained below.

The user can choose between 3 types of source for the accelerometer:

- **Internal (Secondary/BMI088/ADIS16505-3 Accelerometer)**: Autopilot 1x uses the internal sensor.

> **Warning:** If the user has an Autopilot 1x with `hardware version 4.5`, the **Main Accelerometer** is available **instead of the ADIS16505-3 Accelerometer**, which has the same configurable parameters as the Secondary Accelerometer.

- **External sensor 0-1**: Autopilot 1x uses values received from custom messages from a no-integrated external sensor.

- **Decimal var sensor 0-1**: Autopilot 1x uses a decimal value provided by an external sensor.

Hint: Depending on the hardware version, the following accelerometer is suggested:

- `4.5 version` **BMI088** Accelerometer
- `4.8 version` **ADIS16505-3** Accelerometer

It is **possible** to select **multiple** of these sensors for the navigation algorithm, so that Veronte Autopilot 1x performs a combination of all the measurements of the selected accelerometers. This combination consists of calculating the means and variances of each of these accelerometers in a given time (**time constant for mean and time constant for variance**) to obtain a mean weighted with the inverse of the variance. The lower the variance, the greater the weight of that sensor in the mean.

2.3.1.1 Common accelerometers configuration

First of all, the parameters to be configured in the accelerometer panel regardless of the selected accelerometer are presented:

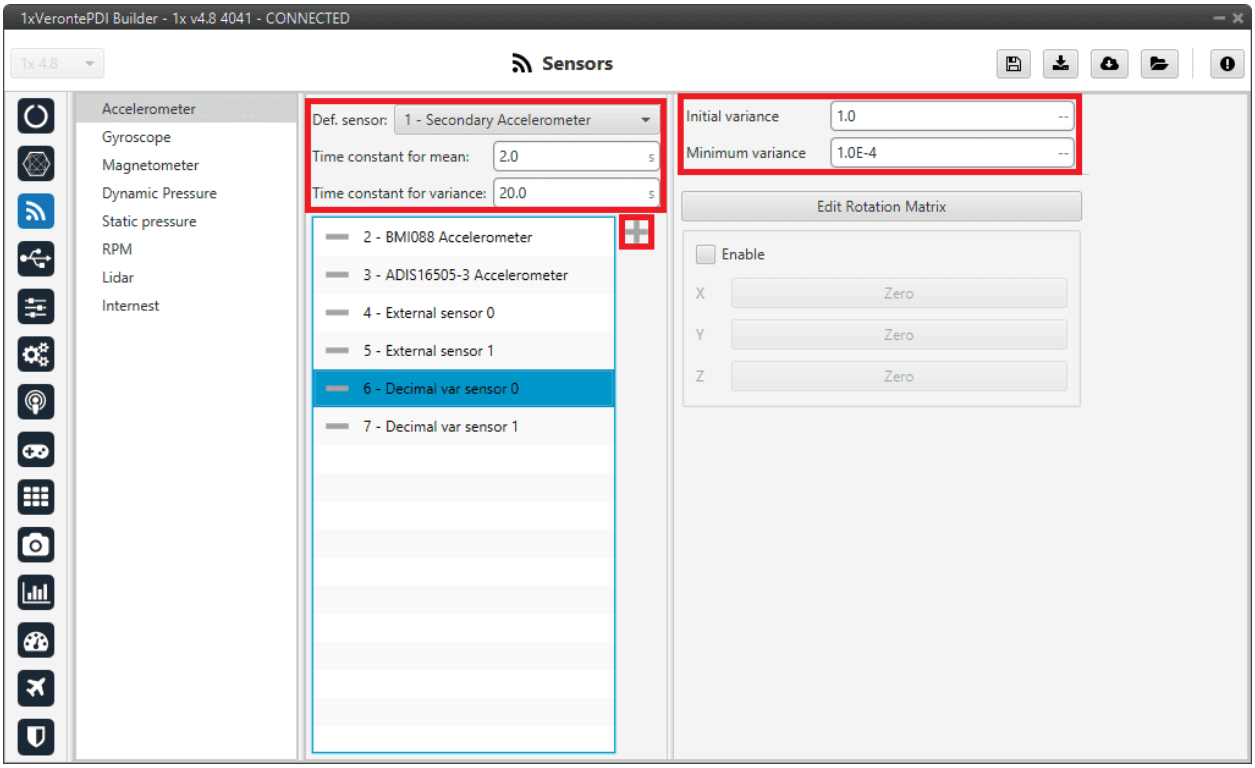


Fig. 42: **Accelerometer panel**

- **Def. sensor**: A default sensor must be chosen. If all selected sensors fail, the measurement value will be that of the default sensor.
- **Time constant for mean/variance**: Time taken by the Autopilot 1x system to estimate the mean/variance.
- **Initial variance**: Initial value of the variance from which the calculation of the combination of the measures is started.
- **Minimum variance**: The value estimated by the system for the variance cannot be less than the one set here.

To select the desired accelerometers, simply add them to the list. To do this, click on the + icon and select them in the panel displayed:

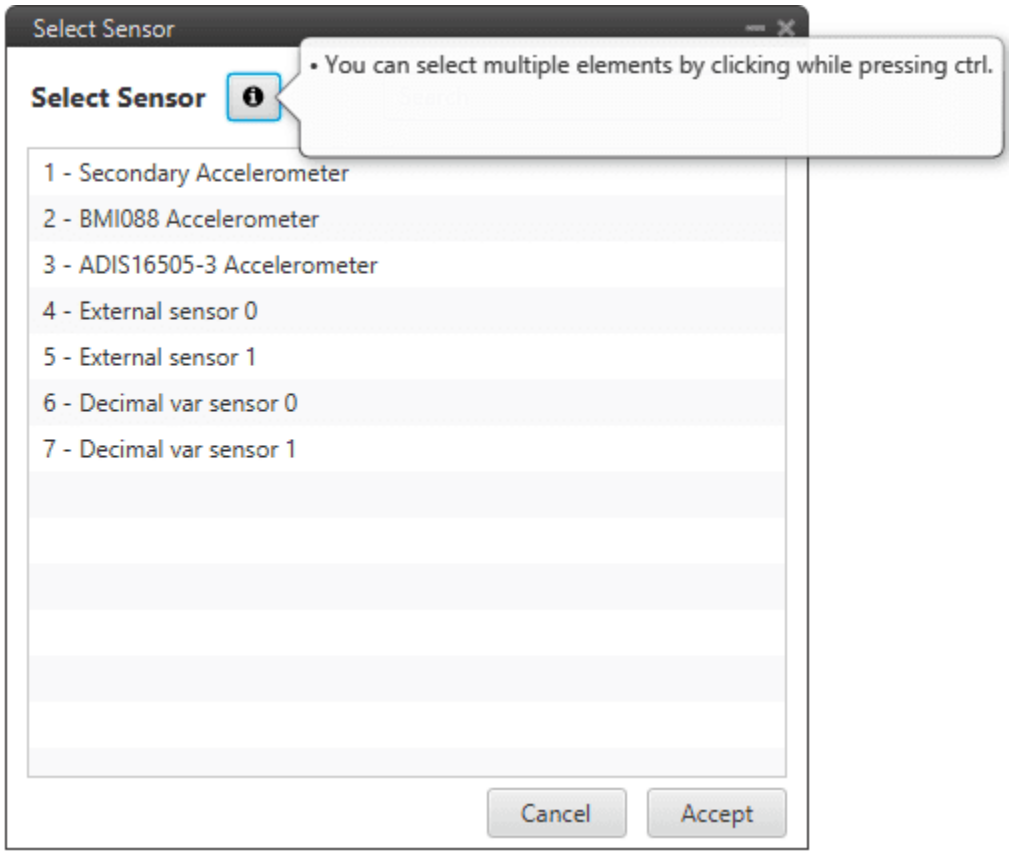


Fig. 43: **Accelerometer panel - Add accelerometers**

2.3.1.2 Common configuration of the internal accelerometers

Secondly, some configuration parameters that are common to all **Internal Accelerometers** (Secondary, BMI088 and ADIS16505-3) are explained:

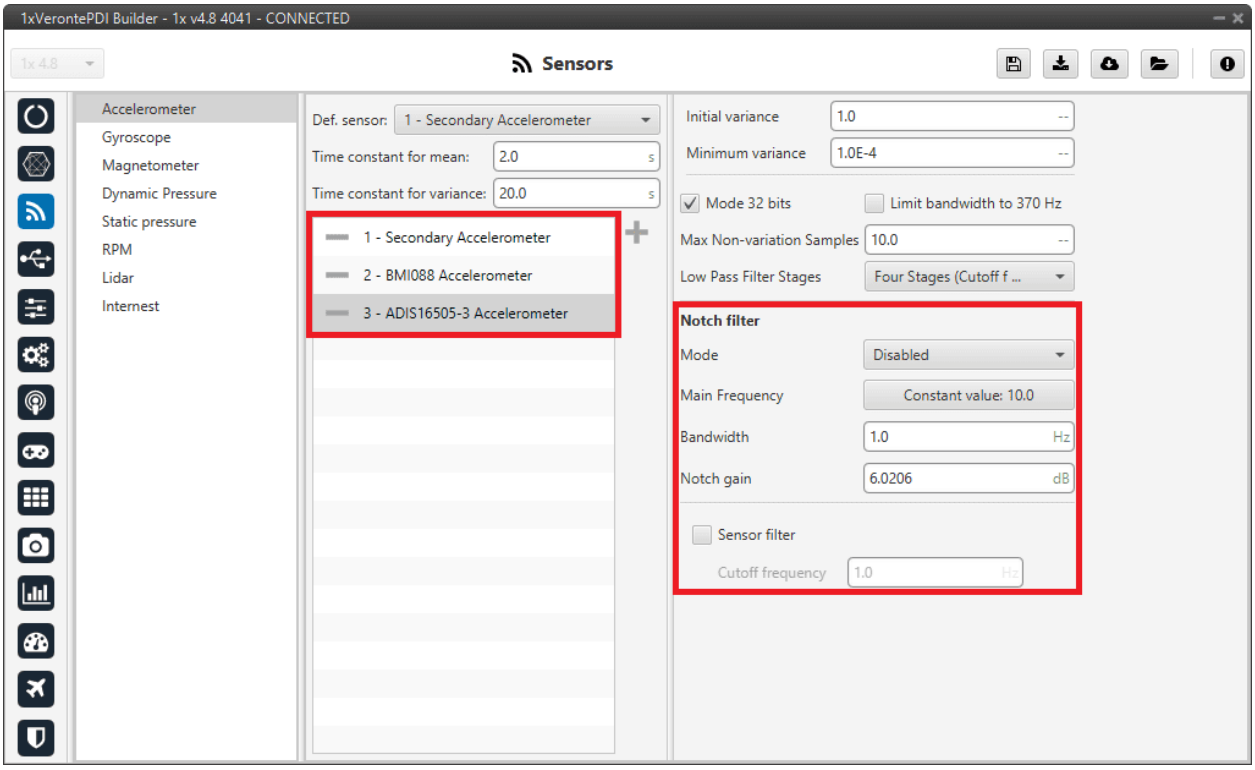


Fig. 44: **Accelerometer panel - Internal accelerometers configuration**

- **Notch filter**: It is a filter that dampens signals only at a specific frequency.
  - **Mode**: The available options are:
    - **Disabled**: Notch filter disabled.
    - **Main frequency**: **Only one filter** is created on the main frequency.
    - **Main frequency + 1st harmonic**: **Two filters** are created, one at the main frequency and one at the first harmonic, i.e. at the frequency which is twice the main frequency.
  - **Main Frequency**: Main frequency (Hz) at which the notch filter reaches its maximum damping.
  - **Bandwidth**: Design parameter. There is a damping of at least 3 dB within the bandwidth. The main frequency at which the maximum damping (notch gain) is reached, lies in the center of this spectrum.
  - **Notch gain**: Design parameter. This parameters sets the maximum damping (dB) for the main frequency.

    Note: Setting this parameter to zero disables the filter.

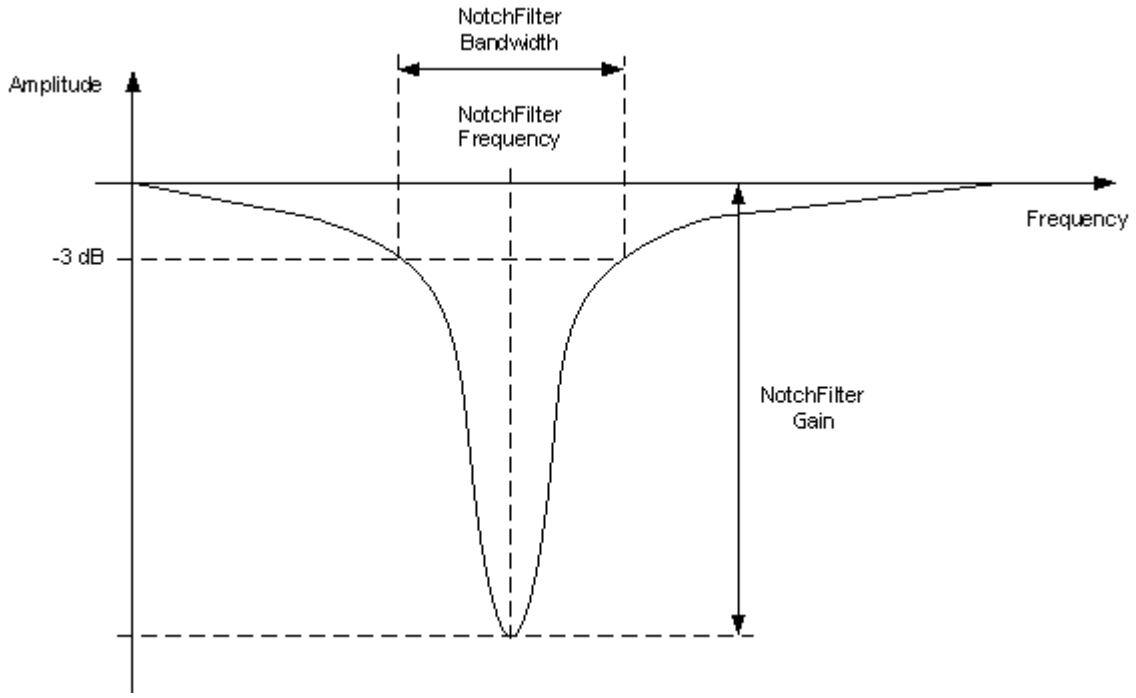


Fig. 45: **Notch filter**

- **Sensor filter**: Enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

2.3.1.3 Sensor

Secondary Accelerometer

This panel displays the possible parameters that can be configured only for the internal Secondary Accelerometer.

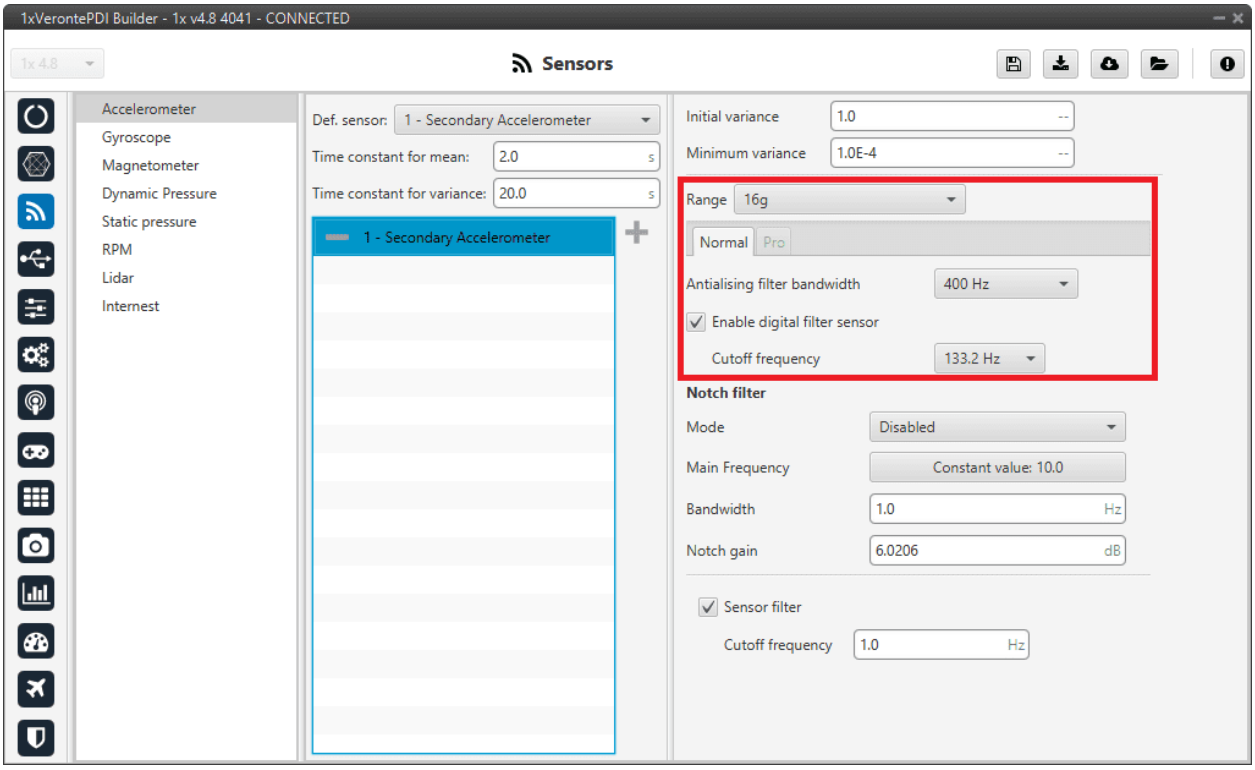


Fig. 46: **Accelerometer panel - Secondary Accelerometer configuration**

In this panel, it is possible to configure different options related to the range and filters of the accelerometer. The parameters that can be modified are:

Note: The configuration parameters *common to all accelerometers* and also the *common to all internal accelerometers*, are not explained again.

- **Range**: Selectable range of forces that the accelerometer can measure, high ranges implies less precision while small ranges might mean the system saturates. Values allowed are **2g**, **4g**, **8g** and **16g**.
- **Antialising filter bandwith**: It is the bandwidth of the antialiasing **low pass filter**. The options available are 50Hz, 100Hz, 200Hz and 400Hz, the greater the value selected the worse the filtering will be.
- **Enable digital filter sensor**: Enables a low pass filter which its **cutoff frequency** is configured from the options 16.65Hz, 66.6Hz, 133.2Hz and 740.0Hz. This is a **hardware filter**, included directly in the accelerometer.

BMI088 Accelerometer

This panel displays the possible parameters that can be configured for the internal BMI088 Accelerometer.

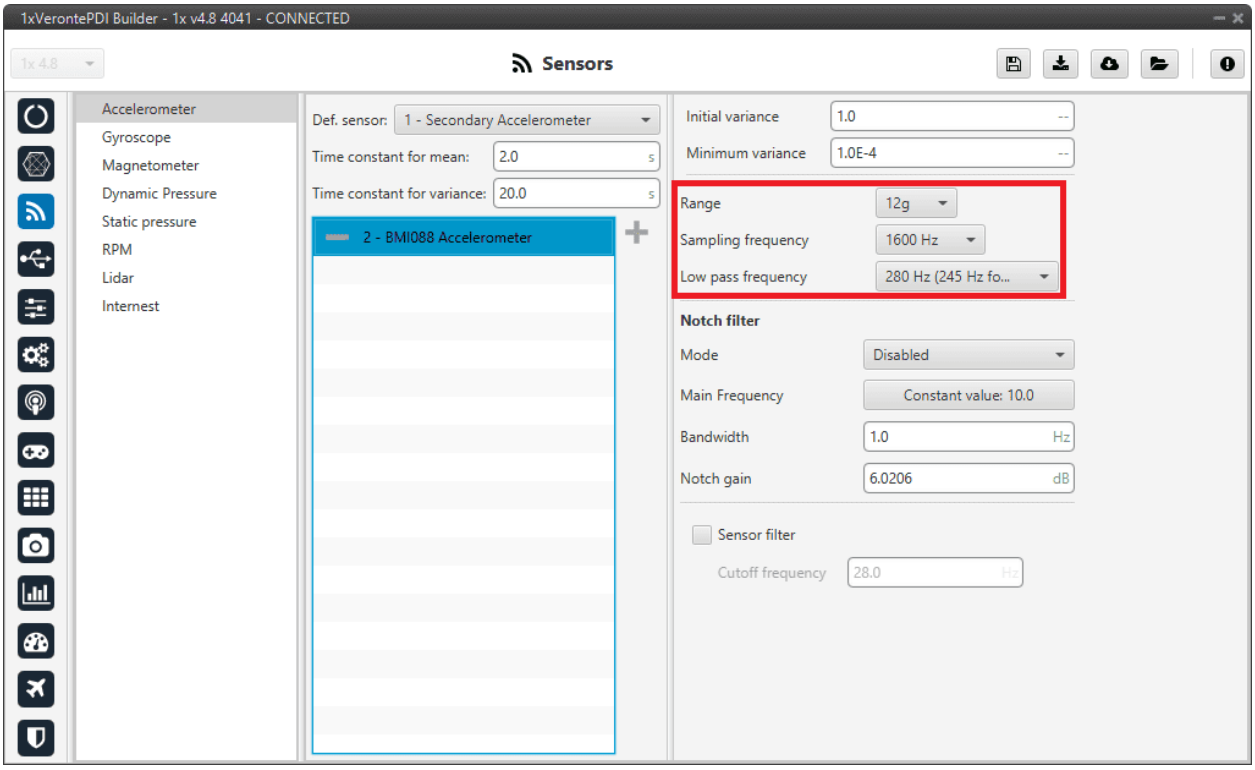


Fig. 47: **Accelerometer panel - BMI088 Accelerometer configuration**

In this panel, it is possible to configure different options related to the range and filters of the accelerometer. The parameters that can be modified are:

Note: The configuration parameters *common to all accelerometers* and also the *common to all internal accelerometers*, are not explained again.

- **Range**: Selectable range of forces that the accelerometer can measure, high ranges implies less precision while small ranges might mean the system saturates. Values allowed are **3g**, **6g**, **12g** and **24g**.
- **Sampling frequency**: That is the frequency at which the measurements are read out. Values allowed are 12.5Hz, 25Hz, 50Hz, 100Hz, 200Hz, 400Hz, 820Hz and 1600Hz. We recommend the **highest**.
- **Low pass frequency**: This is a **hardware filter**, included directly in the accelerometer, which its **cutoff frequency** is configured from the options 145Hz, 234Hz (215Hz for Z axis) and 280Hz (245Hz for Z axis).

ADIS16505-3 Accelerometer

This panel displays the possible parameters that can be configured for the internal ADIS16505-3 Accelerometer.

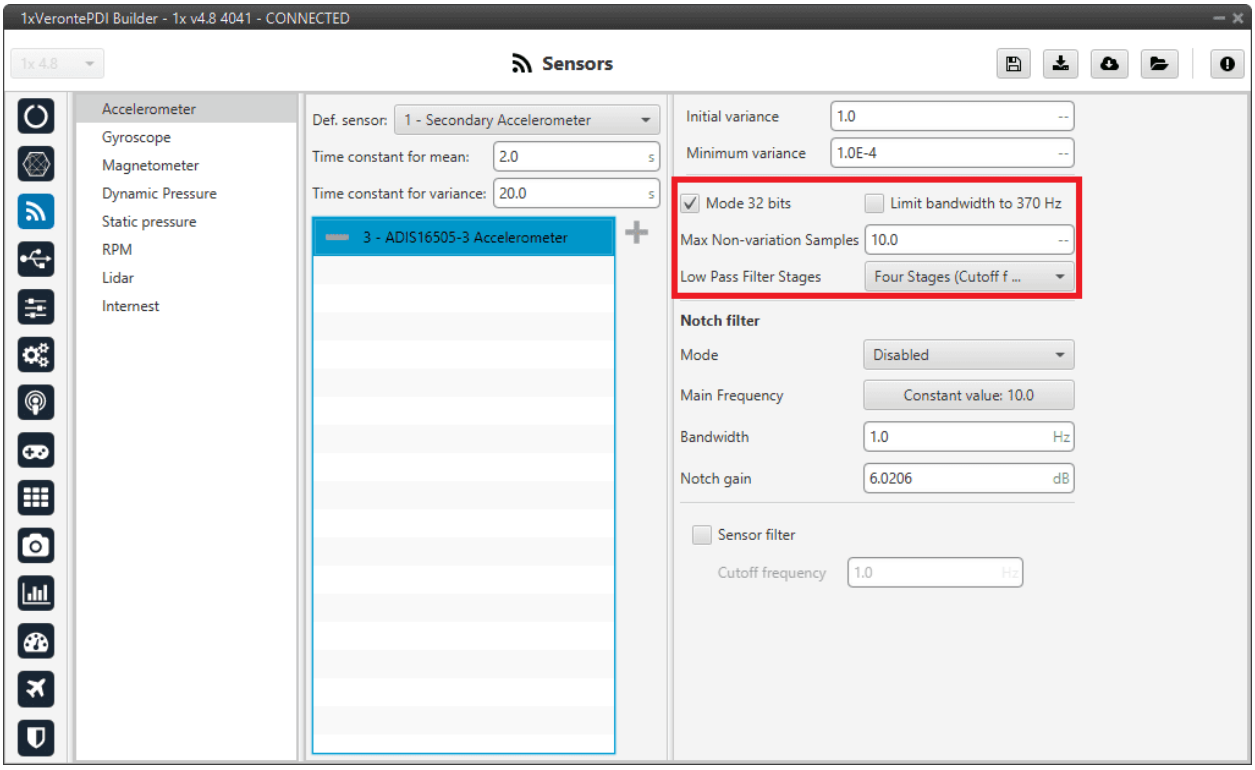


Fig. 48: **Accelerometer panel - ADIS16505-3 Accelerometer configuration**

In this panel it is possible to set different options regarding range and filters from the accelerometer. The parameters that can be modified are:

Note: The configuration parameters *common to all accelerometers* and also the *common to all internal accelerometers*, are not explained again.

- **Mode 32 bits**: Enable or disable. With 32 bits of precision. We recommend **enabling** it.
- **Limit bandwith to 370Hz**: Enable or disable. It can only be used **without** using a **Low Pass Filter Stages**. We recommend **disabling** it.
- **Max Non-variation Samples**: It is configured manually.
- **Low Pass Filter Stages**: IMU's **Hardware** filter. The options available are:
  - No filter
  - 1 stage (Cutoff f=364Hz)
  - 2 stages (Cutoff f=165Hz)
  - 3 stages (Cutoff f=80Hz)
  - 4 stages (Cutoff f=40Hz)
  - 5 stages (Cutoff f=20Hz)
  - 6 stages (Cutoff f=10Hz)

  We recommend **4 stages (Cutoff f=40Hz)** option.

Warning:

- It is **recommended** to choose the **hardware filter** (Low Pass Filter) except if a lower **cutoff frequency** is needed (< 10 Hz).
- It is **not recommended** flying without a filter.

External sensor 0-1

In this panel, the user must configure some parameters in order to correctly receive and manage the measurements from the external sensor.

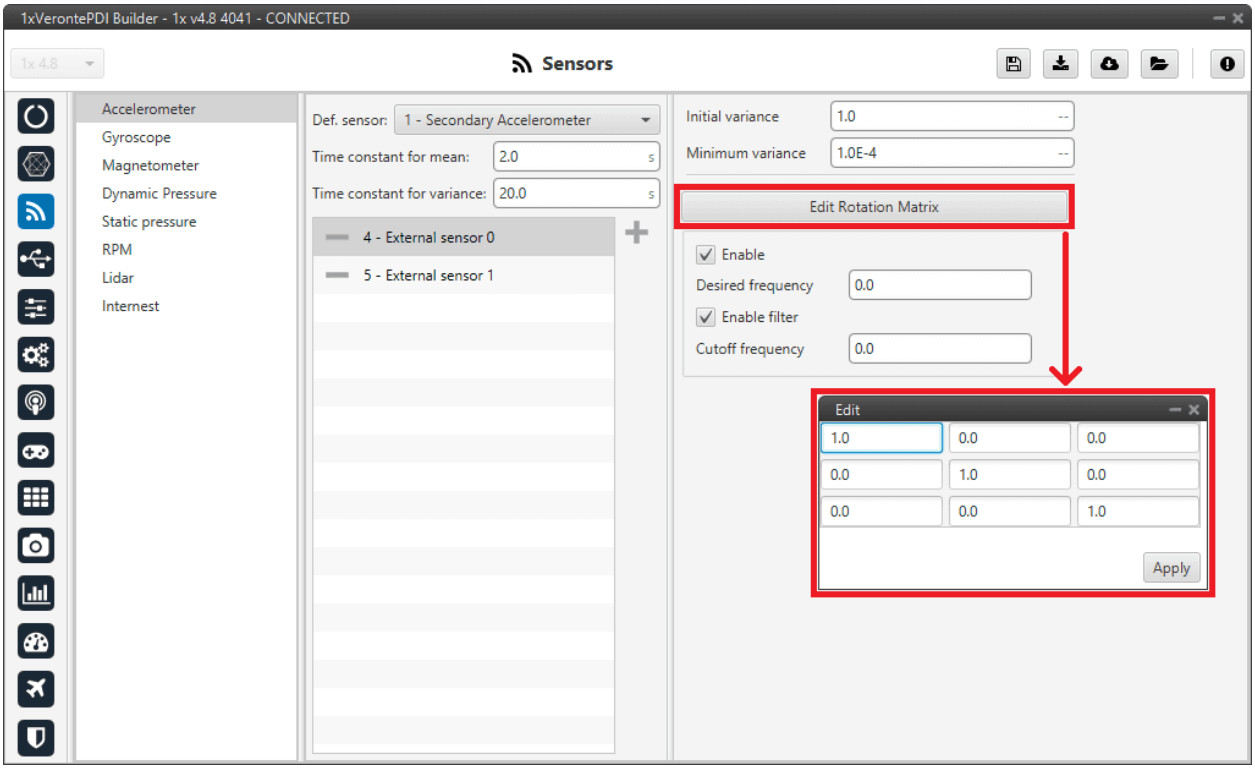


Fig. 49: **Accelerometer panel - External sensor 0-1 configuration**

Note: The configuration parameters *common to all accelerometers* are not explained again.

- **Edit Rotation Matrix**: Users must set the position of the external sensor with respect to the orientation of the Veronte Autopilot 1x.

  Error: The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

  For more information on Autopilot 1x orientation, please refer to the Orientation - Hardware Installation section of the **1x Hardware Manual**.

- **Enable**: Users must enable it if they want this sensor to be selected.
  - **Desired frequency**: The sensor measurement shall only be considered correct if the frequency at which the message with this measurement is received is $\geq 90\%$ of the **desired frequency** defined here by the user.

    This frequency is stored in the **RVars 1488-1489**. For more information on these variables, see the Real Variables (RVar) - 32 Bits section of the **1x Software Manual**.
  - **Enable filter**: Enables a butterworth second order low-pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz.

Decimal var sensor 0-1

In this panel it is possible to configure real variables provided by an external sensor.

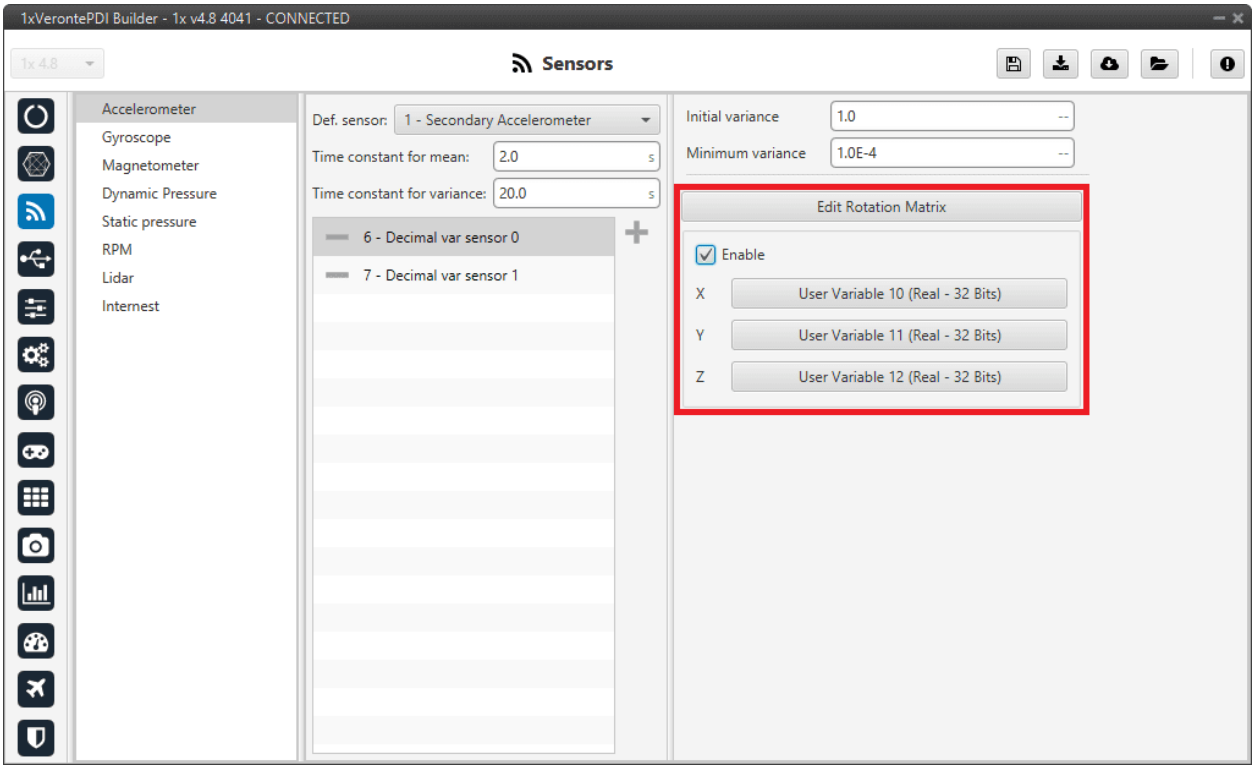


Fig. 50: **Accelerometer panel - Decimal var sensor 0-1 configuration**

Note: The configuration parameters *common to all accelerometers* are not explained again.

- **Edit Rotation Matrix**: Users must set the position of the external sensor with respect to the orientation of the Veronte Autopilot 1x.

Error: The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

For more information on Autopilot 1x orientation, please refer to the Orientation - Hardware Installation section of the **1x Hardware Manual**.

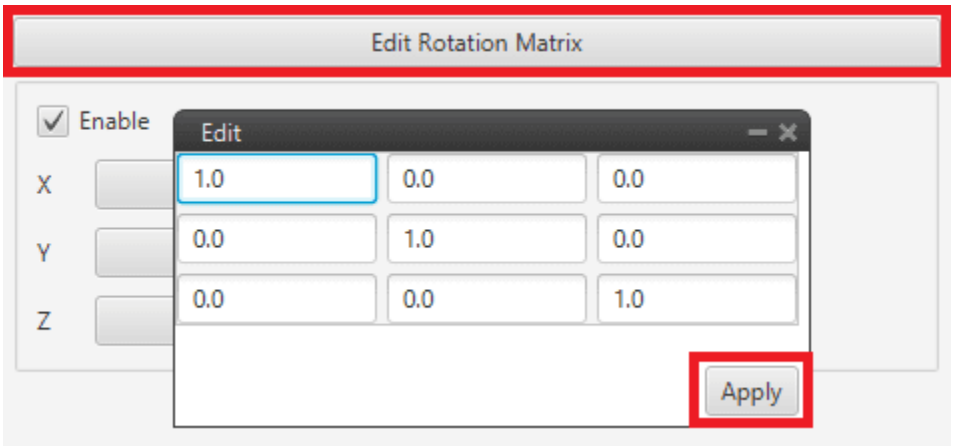


Fig. 51: **Accelerometer panel - Rotation matrix**

- **Enable**: When enabled, the user must select the real user variables for each axis (X,Y,Z) in which the values have been stored.

The configuration process must be done using custom messages. This is to be configured in the *Custom Messages panel* of the **Input/Output menu**. The configuration will depend on the device in use and its communication protocol.

2.3.2 Gyroscope

The gyroscope from the IMU can be configured as explained in the panels shown below.

The user can choose between 3 types of source for the gyroscope:

- **Internal (Secondary/BMI088/ADIS16505-3 Gyroscope)**: Autopilot 1x uses the internal sensor.

> **Warning:** If the user has an Autopilot 1x with `hardware version 4.5`, the **Main Gyroscope** is available instead of the **ADIS16505-3 Gyroscope**, which has the same configurable parameters as the Secondary Gyroscope.

- **External sensor 0-1**: Autopilot 1x uses values received from custom messages from a no-integrated external sensor.
- **Decimal var sensor 0-1**: Autopilot 1x uses a decimal value provided by an external sensor.

Hint: Depending on the hardware version, the following gyroscope is suggested:

- `4.5 version` **BMI088** Gyroscope
- `4.8 version` **ADIS16505-3** Gyroscope

It is **possible** to select **multiple** of these sensors for the navigation algorithm, so that Veronte Autopilot 1x performs a combination of all the measurements of the selected gyroscopes. This combination consists of calculating the means and variances of each of these gyroscopes in a given time (**time constant for mean and time constant for variance**) to obtain a mean weighted with the inverse of the variance. The lower the variance, the greater the weight of that sensor in the mean.

2.3.2.1 Common gyroscopes configuration

First of all, the parameters to be configured in the gyroscope panel regardless of the selected gyroscope are presented:

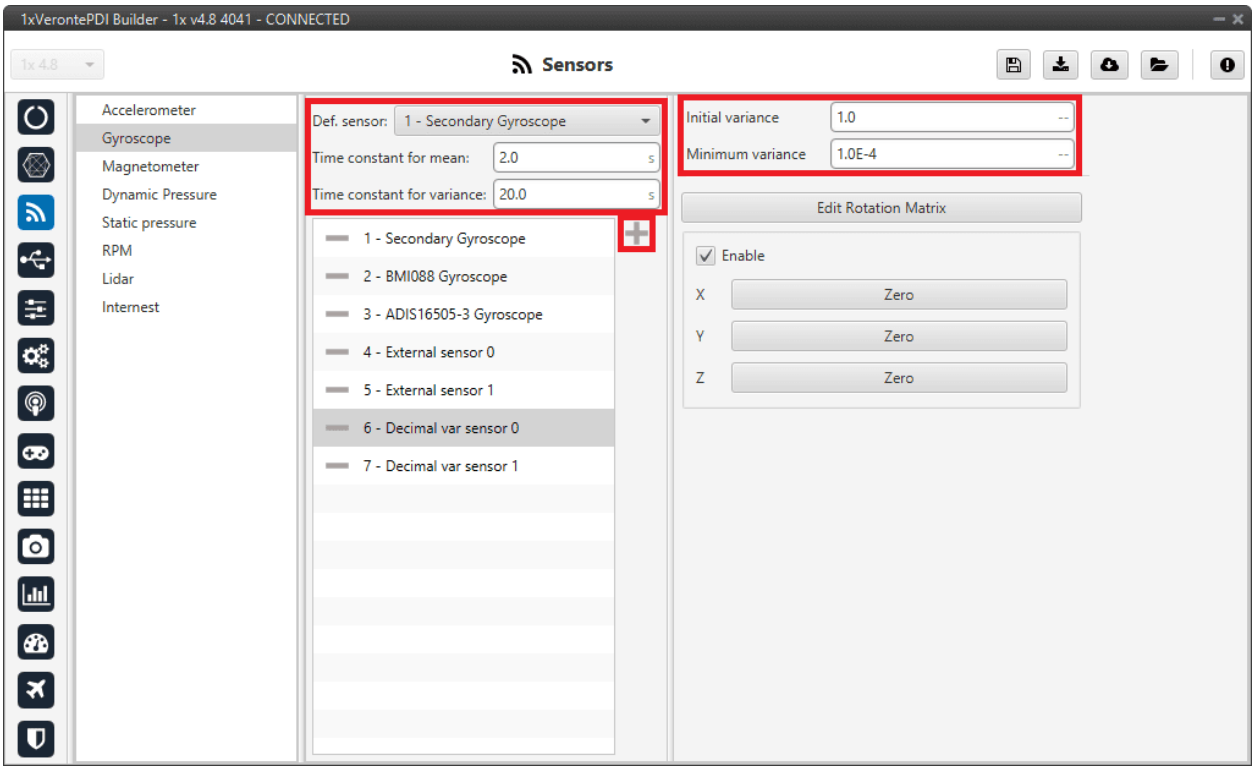


Fig. 52: **Gyroscope panel**

- **Def. sensor**: A default sensor must be chosen. If all selected sensors fail, the measurement value will be that of the default sensor. Default senor must be chosen.
- **Time constant for mean/variance**: Time taken by the Autopilot 1x system to estimate the mean/variance.
- **Initial variance**: Initial value of the variance from which the calculation of the combination of the measures is started.
- **Minimum variance**: The value estimated by the system for the variance cannot be less than the one set here.

To select the desired gyroscopes, simply add them to the list. To do this, click on the + icon and select them in the panel displayed:

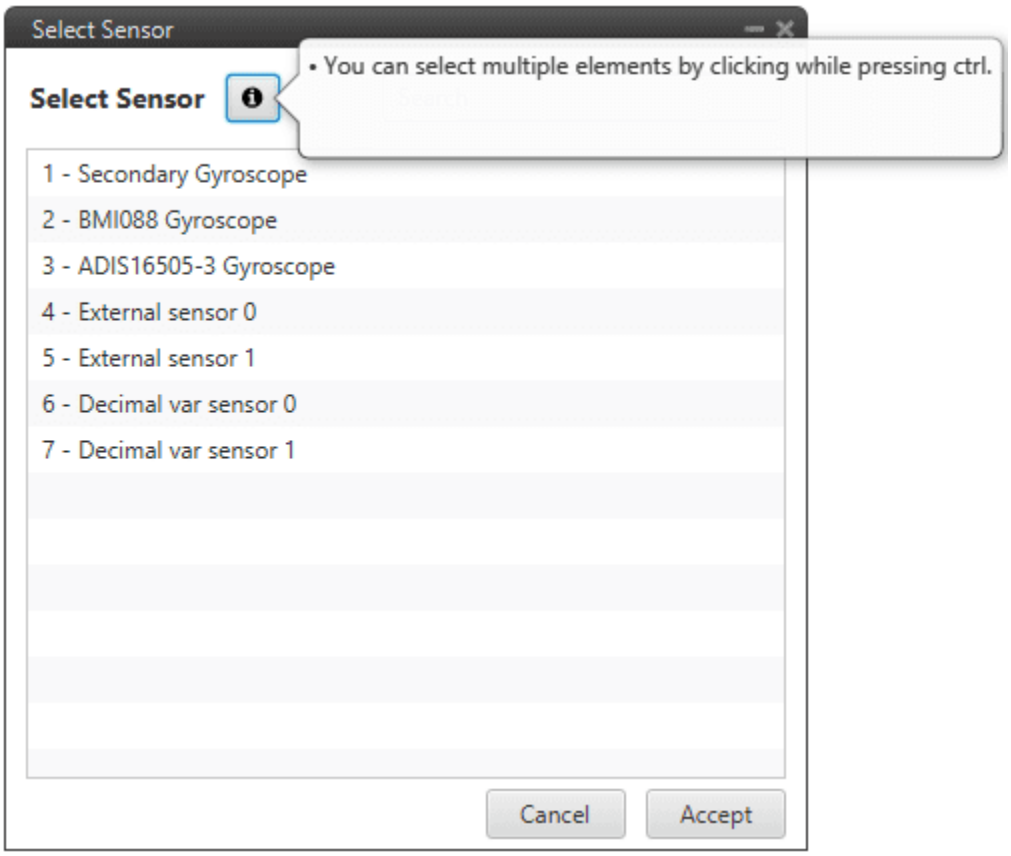


Fig. 53: **Gyroscope panel - Add gyroscopes**

2.3.2.2 Common configuration of the internal gyroscopes

Secondly, some configuration parameters that are common to all **Internal Gyroscopes** (Secondary, BMI088 and ADIS16505-3) are explained:

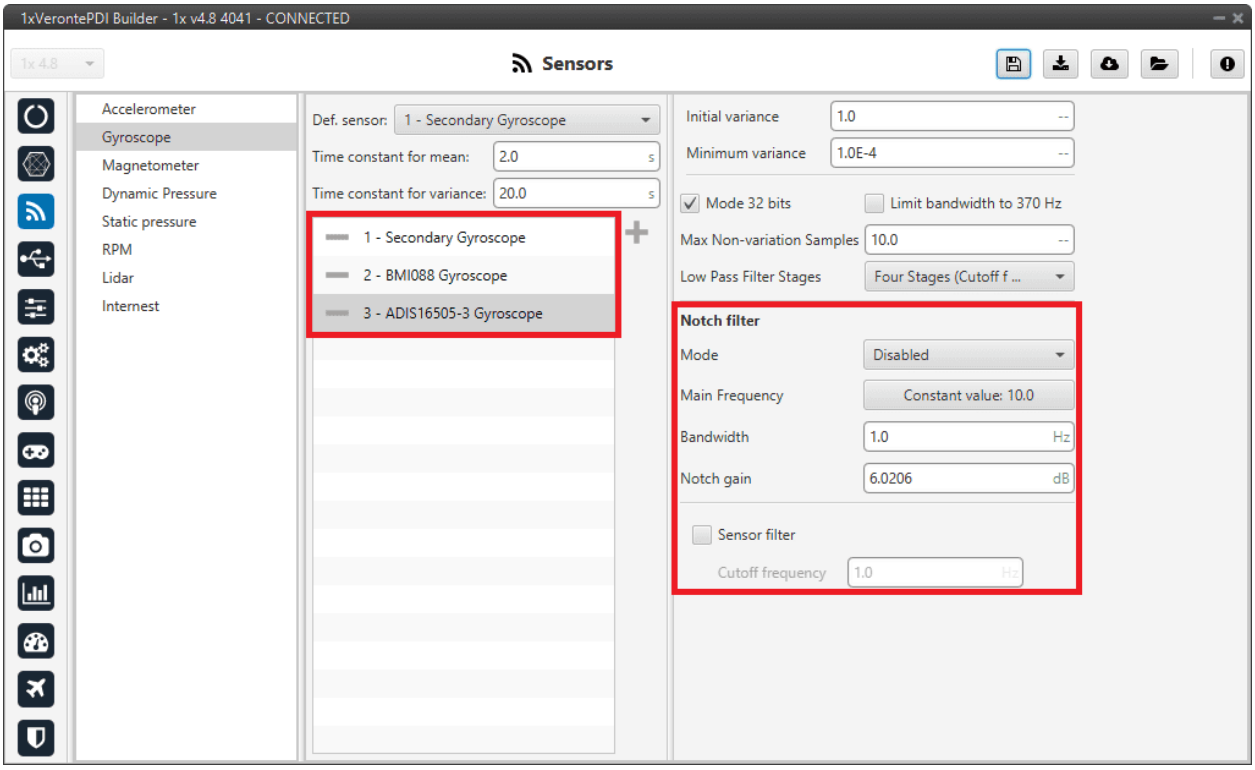


Fig. 54: **Gyroscope panel - Internal gyroscopes configuration**

- **Notch filter**: It is a filter that dampens signals only at a specific frequency.
  - **Mode**: The available options are:
    - **Disabled**: Notch filter disabled.
    - **Main frequency**: **Only one filter** is created on the main frequency.
    - **Main frequency + 1st harmonic**: **Two filters** are created, one at the main frequency and one at the first harmonic, i.e. at the frequency which is twice the main frequency.
  - **Main Frequency**: Main frequency (Hz) at which the notch filter reaches its maximum damping.
  - **Bandwidth**: Design parameter. There is a damping of at least 3 dB within the bandwidth. The main frequency at which the maximum damping (notch gain) is reached, lies in the center of this spectrum.
  - **Notch gain**: Design parameter. This parameters sets the maximum damping (dB) for the main frequency.

    Note: Setting this parameter to zero disables the filter.

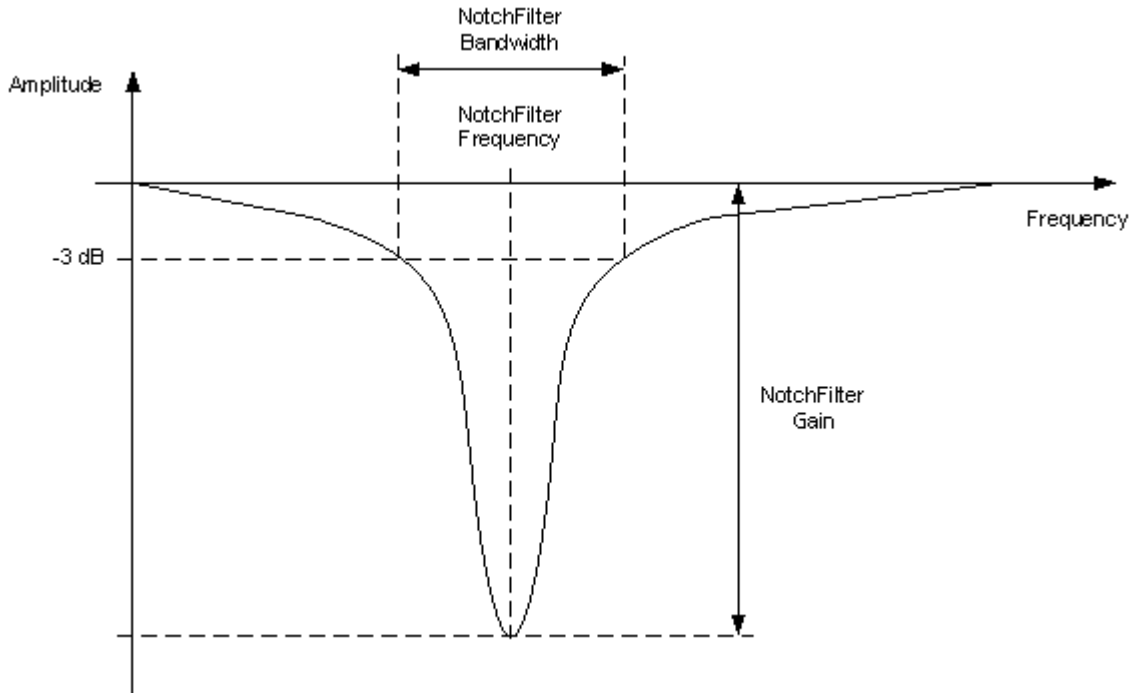


Fig. 55: **Notch filter**

- **Sensor filter**: Enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

2.3.2.3 Sensor

Secondary Gyroscope

This panel displays the possible parameters that can be configured for the internal Secondary Gyroscope.

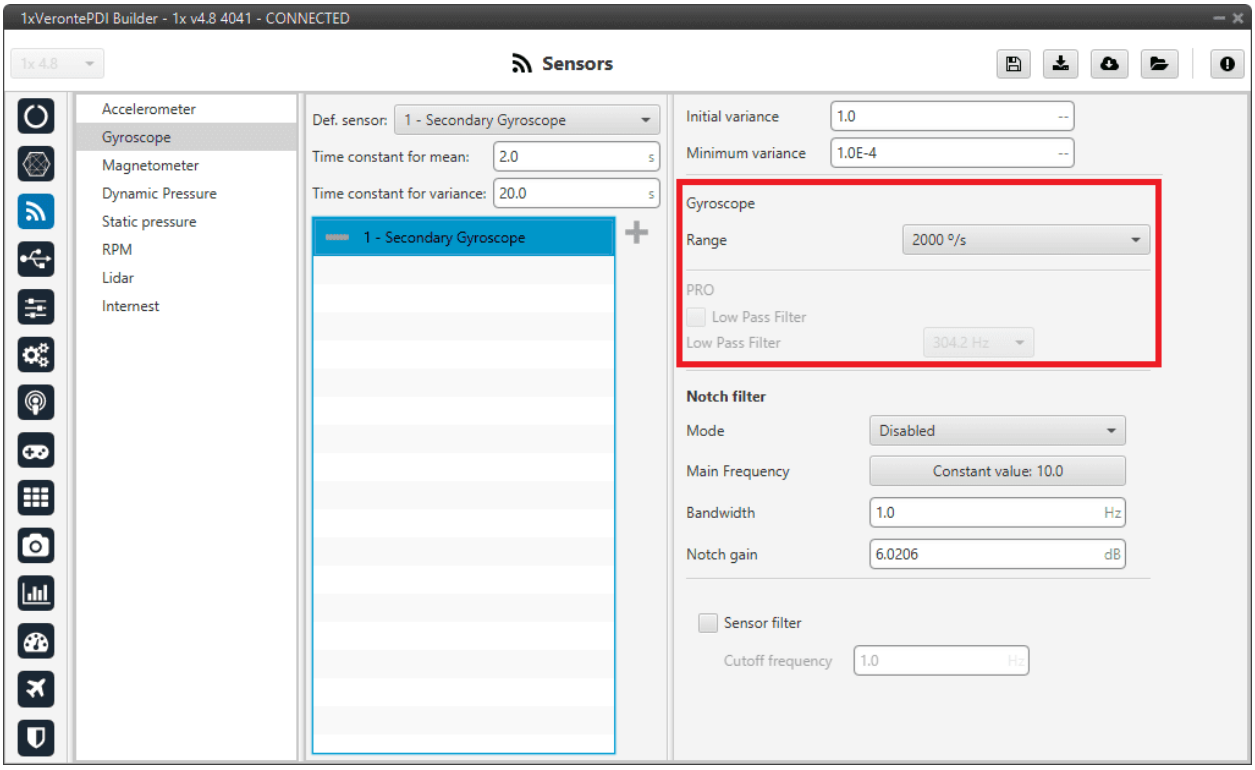


Fig. 56: **Gyroscope panel - Secondary Gyroscope configuration**

In this panel it is possible to set different options regarding range and filters from the gyroscope. The parameters that can be modified are:

Note: The configuration parameters *common to all gyroscopes* and also the *common to all internal gyroscopes*, are not explained again.

- **Range**: Sets the maximum range of performance, high ranges implies less precision while small ranges might mean the system saturates. Values allowed are **125°/s**, **250°/s**, **500°/s**, **1000°/s** and **2000°/s**.

BMI088 Gyroscope

This panel displays the possible parameters that can be configured for the internal BMI088 Gyroscope.

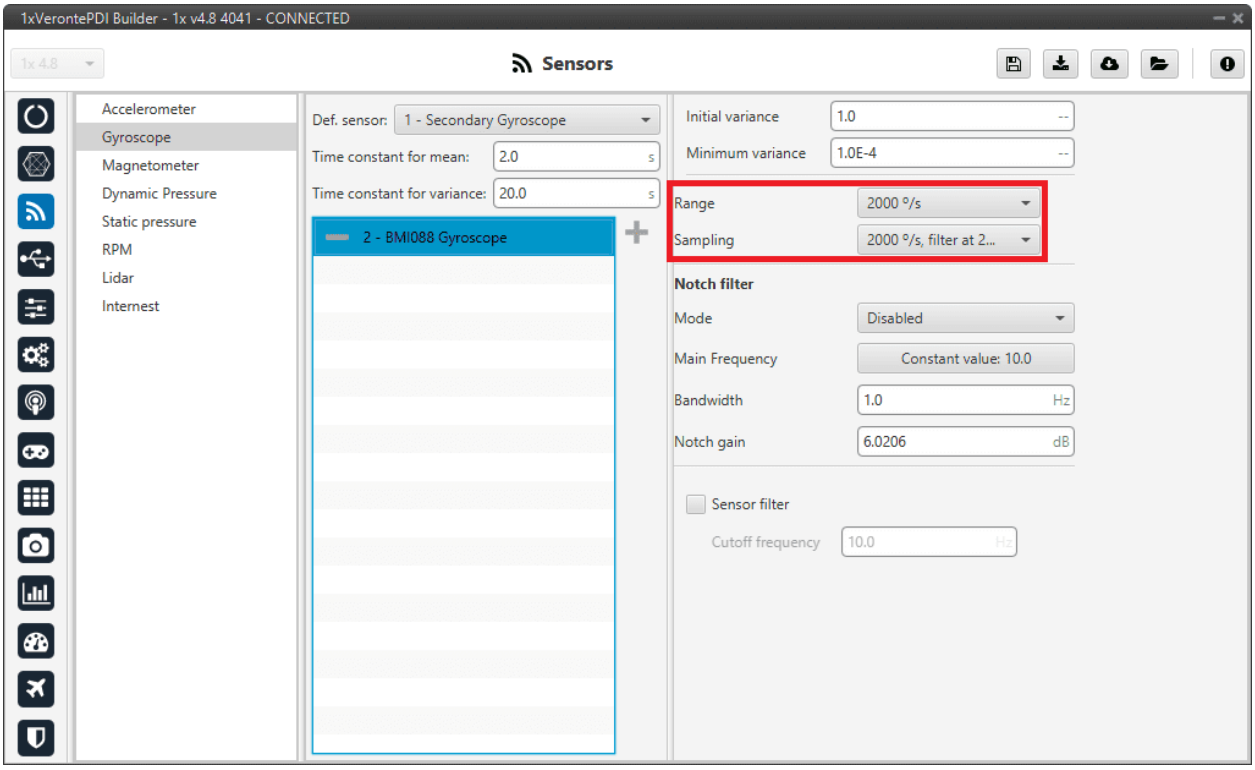


Fig. 57: **Gyroscope panel - BMI088 Gyroscope configuration**

In this panel it is possible to set different options regarding range and filters from the gyroscope. The parameters that can be modified are:

Note: The configuration parameters *common to all gyroscopes* and also the *common to all internal gyroscopes*, are not explained again.

- **Range**: Sets the maximum range of performance, high ranges implies less precision while small ranges might mean the system saturates. Values allowed are **125°/s**, **250°/s**, **500°/s**, **1000°/s** and **2000°/s**.
- **Sampling**: That is the angular velocity at which the measurements are read out. Values allowed are 100°/s filter at 32 Hz, 200°/s filter at 64 Hz, 100°/s filter at 12 Hz, 200°/s filter at 32 Hz, 400°/s filter at 47 Hz, 1000°/s filter at 116 Hz, 2000°/s filter at 230 Hz and 2000°/s filter at 532 Hz.

ADIS16505-3 Gyroscope

This panel displays the possible parameters that can be configured for the internal ADIS16505-3 Gyroscope.

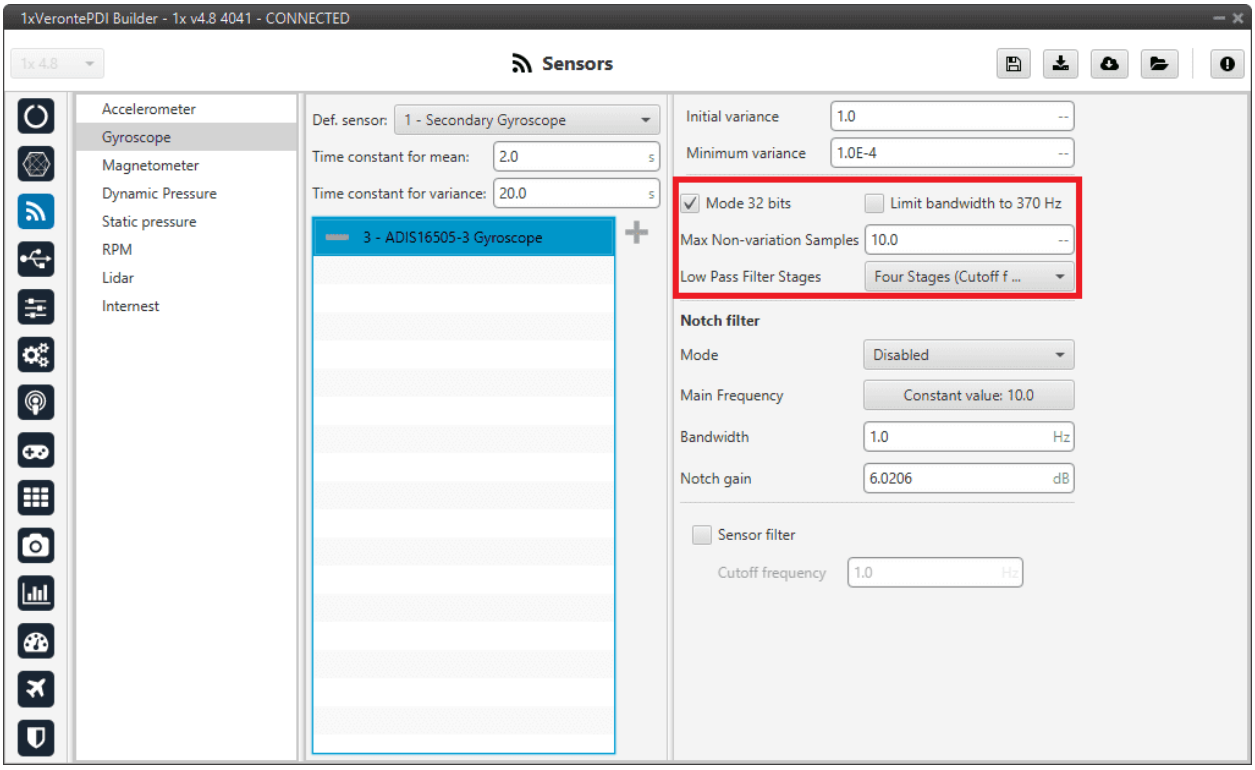


Fig. 58: **Gyroscope panel - ADIS16505-3 Gyroscope configuration**

In this panel it is possible to set different options regarding range and filters from the gyroscope. The parameters that can be modified are:

Note: The configuration parameters *common to all gyroscopes* and also the *common to all internal gyroscopes*, are not explained again.

- **Mode 32 bits**: Enable or disable. With 32 bits of precision. We recommend **enabling** it.
- **Limit bandwith to 370Hz**: Enable or disable. It can only be used **without** using a **Low Pass Filter Stages**. We recommend **disabling** it.
- **Max Non-variation Samples**: It is configured manually.
- **Low Pass Filter Stages**: IMU's **Hardware** filter. The options available are:
  - No filter
  - 1 stage (Cutoff f=364Hz)
  - 2 stages (Cutoff f=165Hz)
  - 3 stages (Cutoff f=80Hz)
  - 4 stages (Cutoff f=40Hz)
  - 5 stages (Cutoff f=20Hz)
  - 6 stages (Cutoff f=10Hz)

  We recommend **4 stages (Cutoff f=40Hz)** option.

Warning:

- It is **recommended** to choose the **hardware filter** (Low Pass Filter) except if a lower **cutoff frequency** is needed (< 10 Hz).
- It is **not recommended** flying without a filter.

External sensor 0-1

In this panel, the user must configure some parameters in order to correctly receive and manage the measurements from the external sensor.

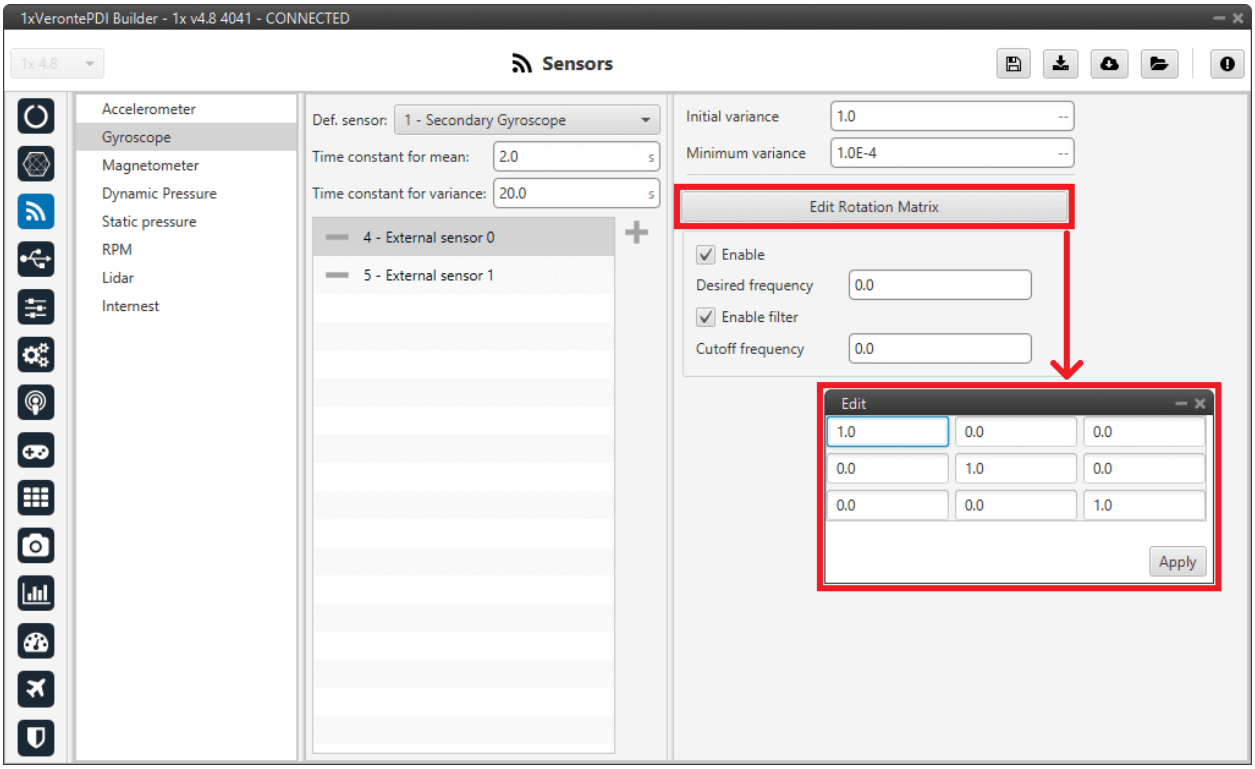


Fig. 59: **Gyroscope panel - External sensor 0-1 configuration**

Note: The configuration parameters *common to all gyroscopes* are not explained again.

- **Edit Rotation Matrix**: Users must set the position of the external sensor with respect to the orientation of the Veronte Autopilot 1x.

  Error: The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

  For more information on Autopilot 1x orientation, please refer to the Orientation - Hardware Installation section of the **1x Hardware Manual**.

- **Enable**: Users must enable it if they want this sensor to be selected.
  - **Desired frequency**: The sensor measurement shall only be considered correct if the frequency at which the message with this measurement is received is $\geq 90\%$ of the **desired frequency** defined here by the user.

    This frequency is stored in the **RVars 1488-1489**. For more information on these variables, see the Real Variables (RVar) - 32 Bits section of the **1x Software Manual**.
  - **Enable filter**: Enables a butterworth second order low-pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz.

Decimal var sensor 0-1

In this panel it is possible to configure real variables provided by an external sensor.

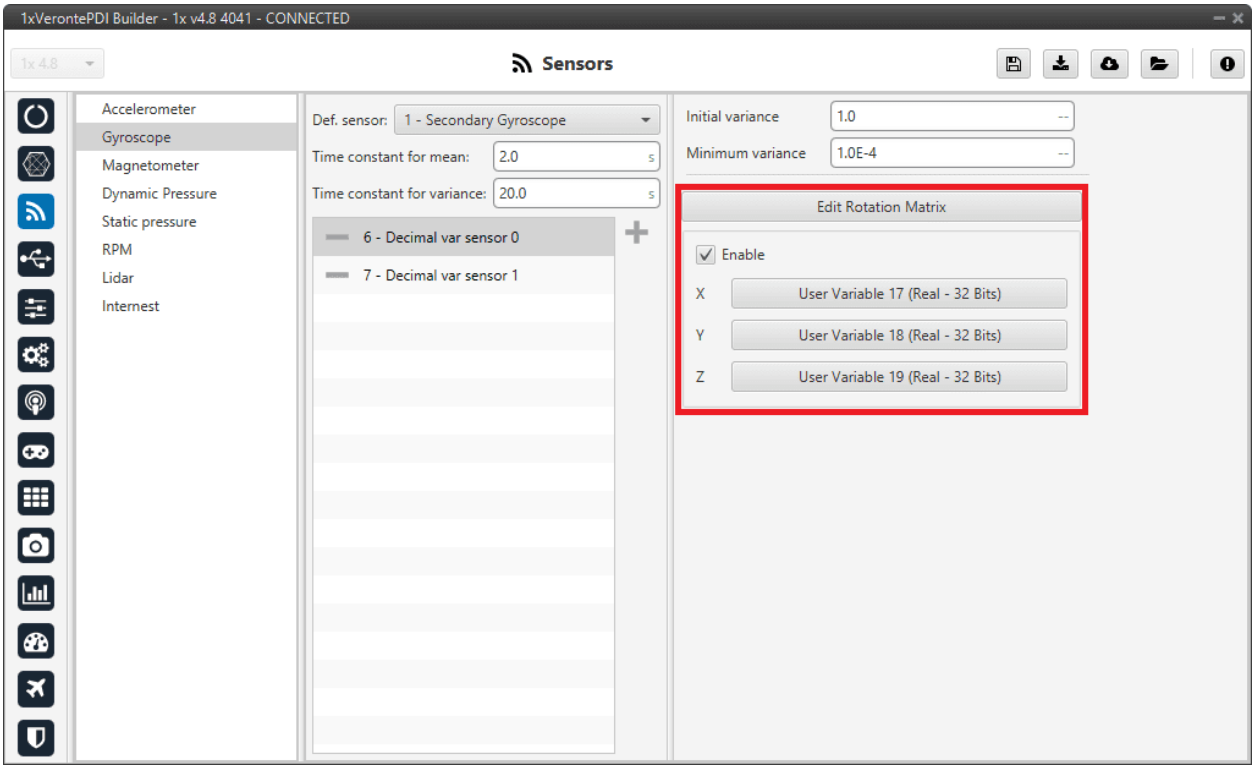


Fig. 60: **Gyroscope panel - Decimal var sensor 0-1 configuration**

Note: The configuration parameters *common to all gyroscopes* are not explained again.

- **Edit Rotation Matrix**: Users must set the position of the external sensor with respect to the orientation of the Veronte Autopilot 1x.

Error: The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

For more information on Autopilot 1x orientation, please refer to the Orientation - Hardware Installation section of the **1x Hardware Manual**.

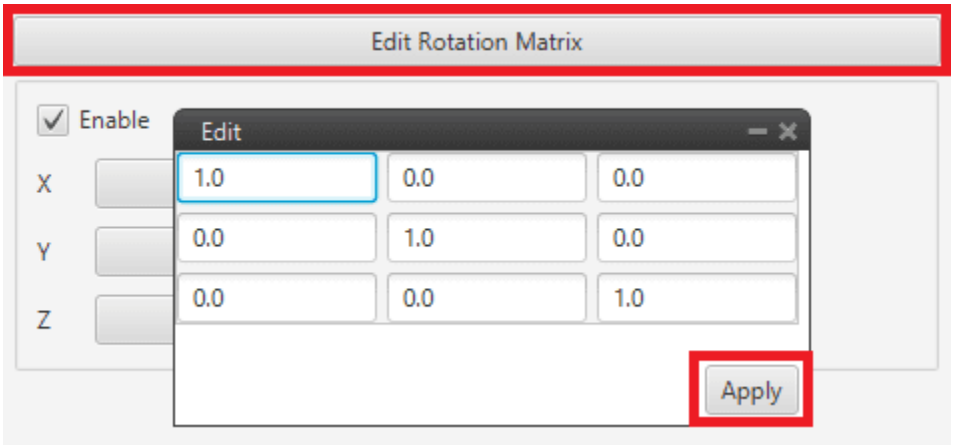


Fig. 61: **Gyroscope panel - Rotation matrix**

- **Enable**: When enabled, the user must select the real user variables for each axis (X,Y,Z) in which the values have been stored.

The configuration process must be done using custom messages. This is to be configured in the *Custom Messages panel* of the **Input/Output menu**. The configuration will depend on the device in use and its communication protocol.

2.3.3 Magnetometer

Veronte Autopilot 1x incorporates three internal magnetometers (only two are available on Autopilots 1x with `hardware version 4.5`) that allow the 1x system to measure the magnetic field. In addition, external magnetometers can also be used for this purpose.

The user can choose between 4 types of source for the magnetometer.

- **Internal (LIS3MDL/MMC5883MA/RM3100)**: Autopilot 1x uses the internal sensor.

> **Warning:** If the user has an Autopilot 1x with `hardware version 4.5`, **Internal RM3100** Magnetometer is **not available**.

- **External sensor 0-1**: Autopilot 1x uses values received from custom messages from a no-integrated external sensor.
- **Decimal var sensor 0-1**: Autopilot 1x uses a decimal value provided by a no-integrated external sensor.
- **External (HMR2300/LIS3MDL/HSCDTD008A/MMC5883MA/RM3100)**: Autopilot 1x uses the information from one of the compatible external magnetometers.

It is **possible** to select **multiple** of these sensors for the navigation algorithm, so the user will have **redundancy** in magnetometer sensor.

Hint: Depending on the hardware version, the following magnetometer is suggested:

- `4.5 version` **Internal MMC5883MA** Magnetometer
- `4.8 version` **Internal RM3100** Magnetometer

Note: **LIS3MDL** and **HSCDTD008A Magnetometer** are integrated in Autopilot 1x with the **I2C interface**. They are inside the 1x, but mounted externally to avoid the possible interferences from being close to electronic components.

- **Magnetometer LIS3MDL**: It is a three-axis magnetic sensor with a very small package.
- **Magnetometer HSCDTD008A**: It is a three-axis terrestrial magnetism sensor of the digital output.

It is very important to know that address cannot be chosen in the software and must be as follows:

- **Magnetometer LIS3MDL**: 0x1C.
- **Magnetometer HSCDTD008A**: 0x0C.

Important: In this panel, the user can only modify some configurable parameters, the selection of the magnetometer and the configuration of the **Variance** are done in the *Magnetometer - Sensors blocks* of the **Block Programs menu**.

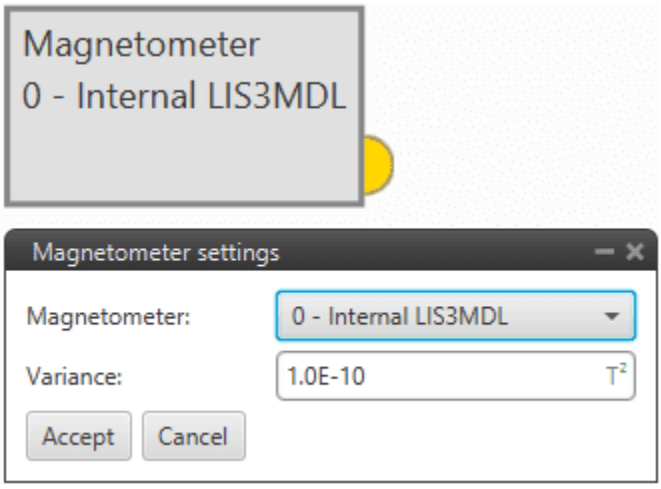


Fig. 62: **Magnetometer block**

2.3.3.1 Sensor

Internal LIS3MDL/MMC5883MA/RM3100

This panel displays the possible parameters that can be configured for the internal magnetometers.

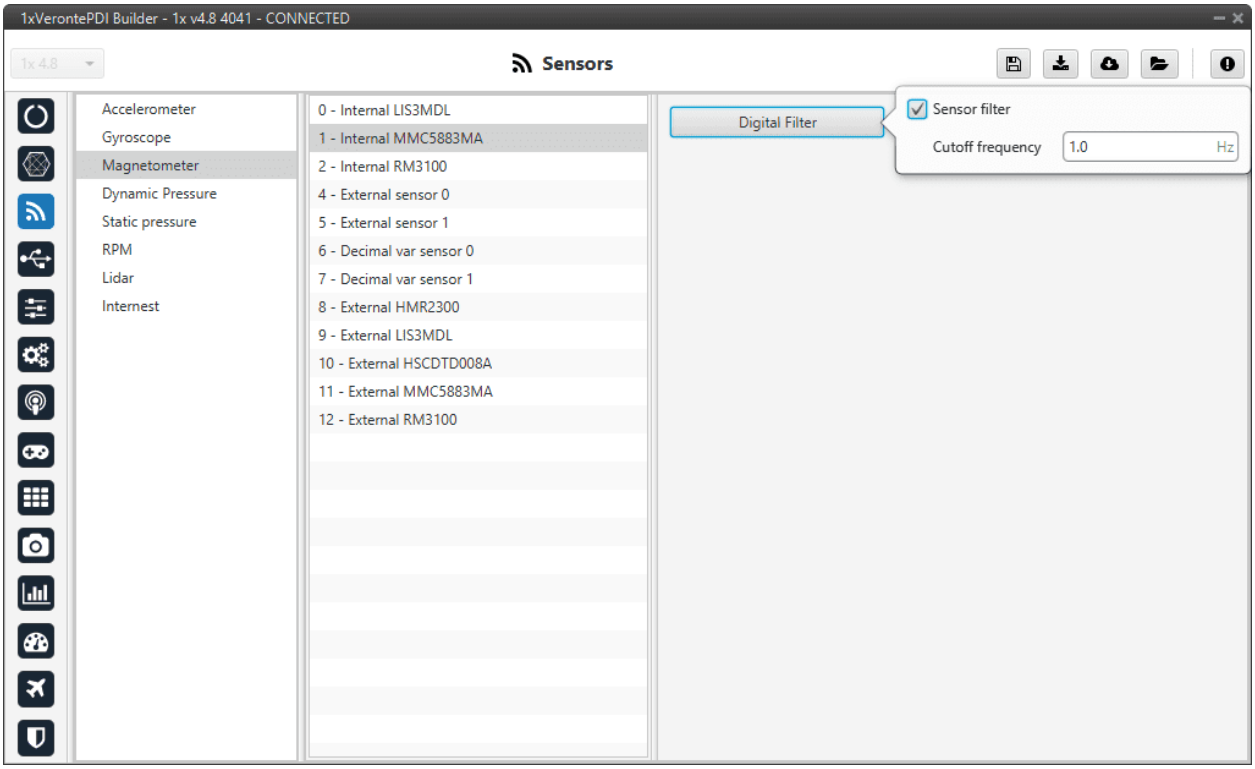


Fig. 63: **Magnetometer panel - Internal magnetometers configuration**

In this panel it is possible the following options regarding the digital filter:

- **Digital filter**: By enabling the **Sensor filter** checkbox, the user enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

External sensor 0-1

In this panel, the user must configure some parameters in order to correctly receive and manage the measurements from the external sensor.

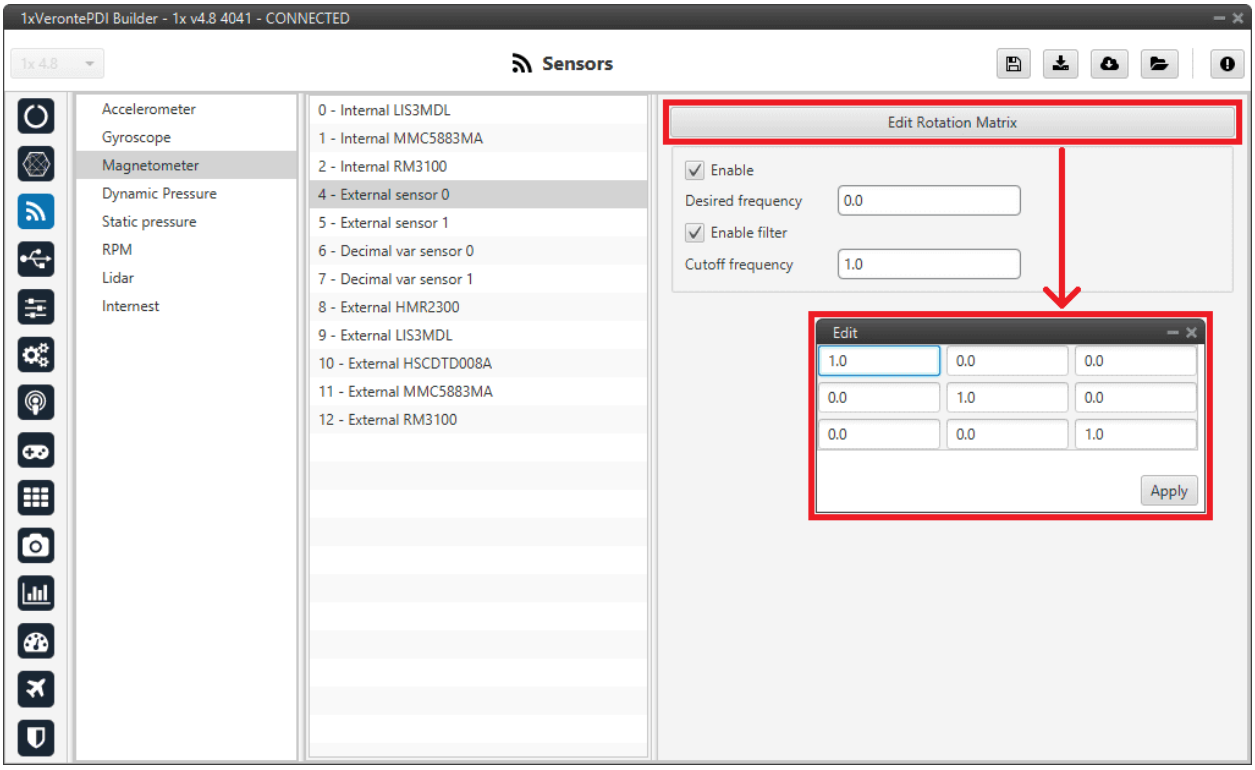


Fig. 64: **Magnetometer panel - External sensor 0-1 configuration**

- **Edit Rotation Matrix**: Users must set the position of the external sensor with respect to the orientation of the Veronte Autopilot 1x.

  Error: The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

  For more information on Autopilot 1x orientation, please refer to the Orientation - Hardware Installation section of the **1x Hardware Manual**.

- **Enable**: Users must enable it if they want this sensor to be selected.
  - **Desired frequency**: The sensor measurement shall only be considered correct if the frequency at which the message with this measurement is received is $\geq 90\%$ of the **desired frequency** defined here by the user.

    This frequency is stored in the **RVars 1488-1489**. For more information on these variables, see the Real Variables (RVar) - 32 Bits section of the **1x Software Manual**.
  - **Enable filter**: Enables a butterworth second order low-pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz.

Decimal var sensor 0-1

In this panel it is possible to configure real variables provided by an external sensor.

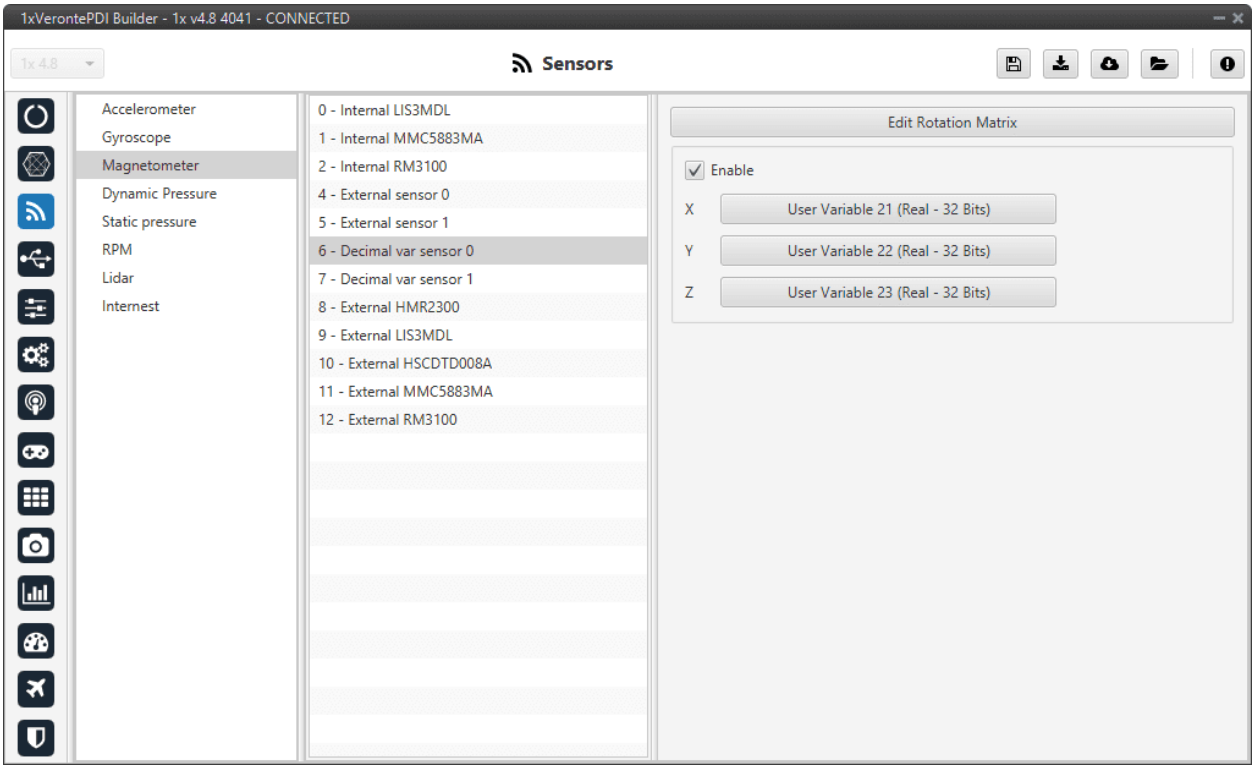


Fig. 65: **Magnetometer panel - Decimal var sensor 0-1 configuration**

- **Edit Rotation Matrix**: Users must set the position of the external sensor with respect to the orientation of the Veronte Autopilot 1x.

> **Error:** The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

For more information on Autopilot 1x orientation, please refer to the Orientation - Hardware Installation section of the **1x Hardware Manual**.

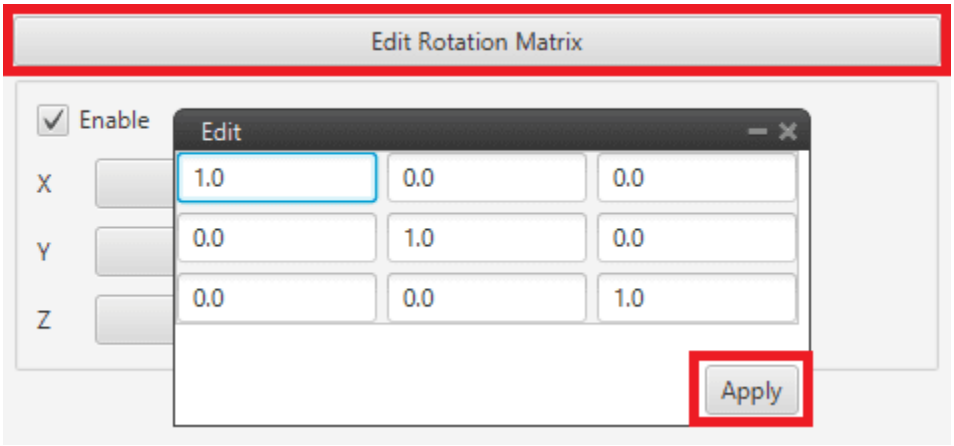


Fig. 66: **Magentometer panel - Rotation matrix**

- **Enable**: When enabled, the user must select the real user variables for each axis (X,Y,Z) in which the values have been stored.

The configuration process must be done using custom messages. This is to be configured in the *Custom Messages panel* of the **Input/Output menu**. The configuration will depend on the device in use and its communication protocol.

External HMR2300/LIS3MDL/HSCDTD008A/MMC5883MA/RM3100

Autopilot 1x has been designed to have compatibility with these external magnetometers.

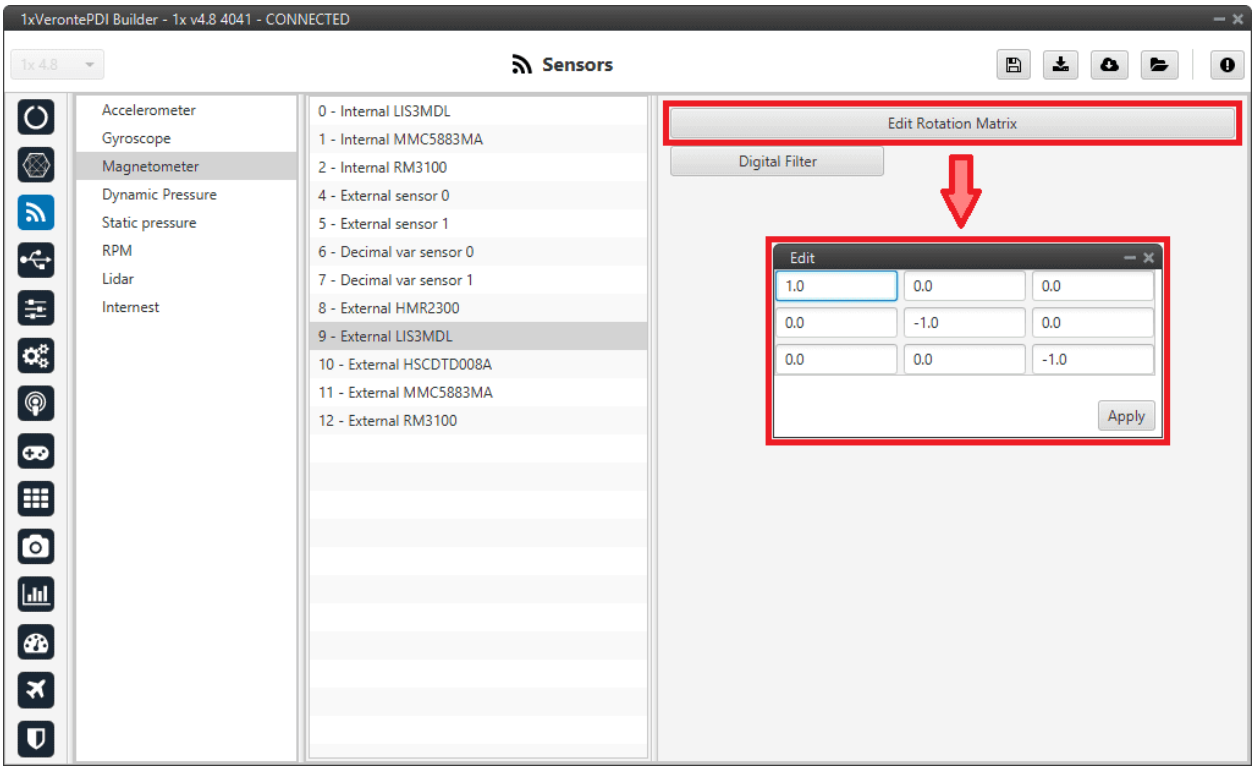


Fig. 67: **Magnetometer panel - External magnetometers configuration**

The user can modify the following parameters:

- **Edit Rotation Matrix**: It must be modified in the case that the axes of the magnetometer do not coincide with those of the aircraft.

> **Error:** The rotation matrix **cannot** be a **zero matrix** and must respect **the orthogonality of the axes**. Not complying with this requirement means an invalid rotation and, consequently, the calibration of this magnetometer will not be possible.

- **Digital filter**: By enabling the **Sensor filter** checkbox, the user enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

2.3.4 Dynamic Pressure

Autopilot 1x has three pressure input lines, two for static pressure to determine the absolute pressure and one for pitot in order to determine the dynamic pressure.

This panel allows the user to configure a Dynamic Pressure sensor input in 1x.

2.3.4.1 Navigation

The **Navigation** panel has 5 parameters that are configured independently from the Dynamic Pressure sensor selected.

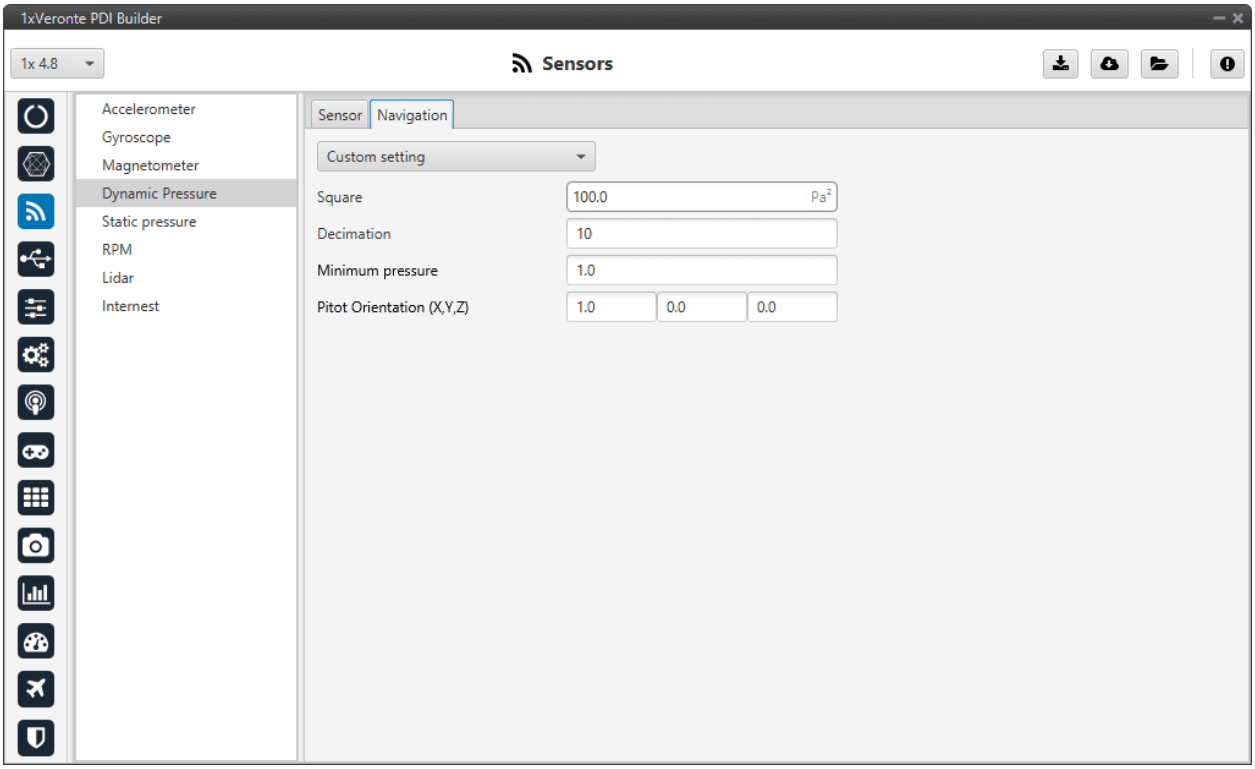


Fig. 68: **Dynamic Pressure panel - Navigation parameters**

- **Enable**: User can choose from *Disabled*, values from the sensor will be received, but they will not enter the Navigation Filter, or *Custom settings*, which will take into account all the following parameters.
- **Square Error**: Sensor error square, which defines the weight if the measurement into the Navigation filter.
- **Decimation**: Defines the bunch of data from which 1 value will be stored. For example, if decimation is 10, every 10 measurements 1 will be taken into account. This procedure is used to reduce the number of samples.

  If users have any questions about decimation, please refer to the *What does decimation mean? - FAQ* section of this manual.
- **Minimum pressure**: Minimum pressure readable from the sensor.
- **Pitot Orientation (X,Y,Z)**: Vector defining the Pitot orientation on the platform.

2.3.4.2 Sensor

The user can choose between 3 types of source:

- **Internal**: Autopilot 1x uses the internal sensor.
- **Integer var sensor 0-1**: Autopilot 1x uses a integer value provided by an external sensor.
- **Decimal var sensor 0-1**: Autopilot 1x uses a decimal value provided by an external sensor.

Internal

This panel displays the possible parameters that can be configured for the internal Dynamic Pressure sensor.

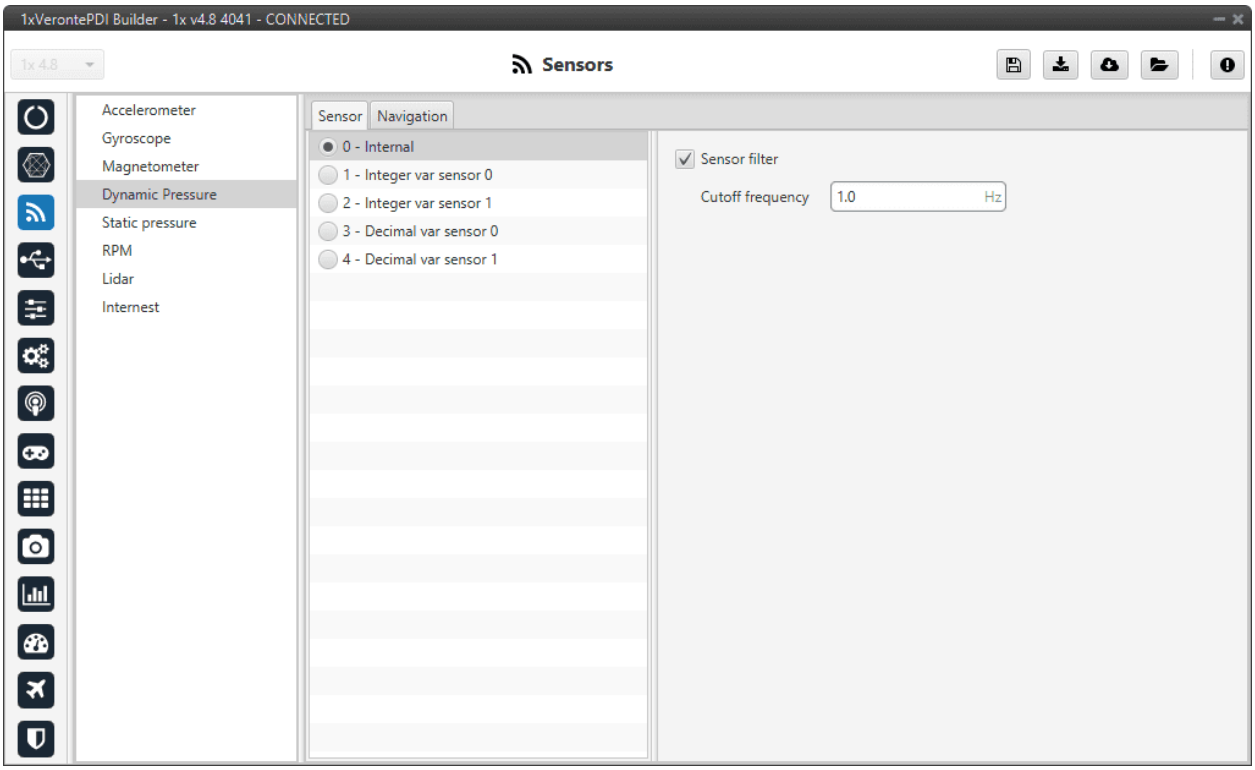


Fig. 69: **Dynamic Pressure panel - Internal sensor configuration**

In this panel it is possible to set different options regarding filters.

- **Sensor filter**: Enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

Integer var sensor 0-1

In this panel it is possible to configure integer variables provided by an external sensor.

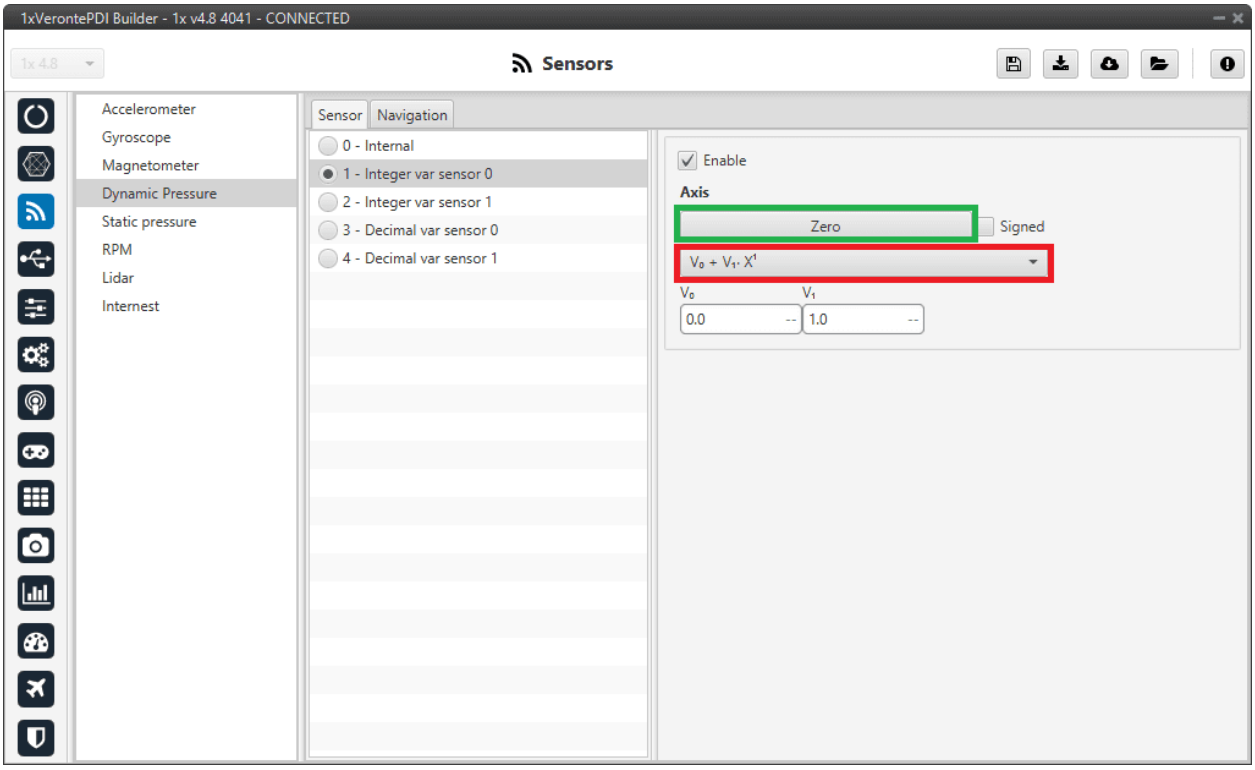


Fig. 70: **Dynamic Pressure panel - Integer var sensor 0-1 configuration**

When Integer var sensor 1 or 2 are selected, the previous panel will be shown. In this panel, the user selects the variable that has been stored in a user variable (Green Box) and the operations that will be carried on (Red Box). It is possible **to use the signal through a linear or quadratic relation**. The following image shows an example of a linear relation:

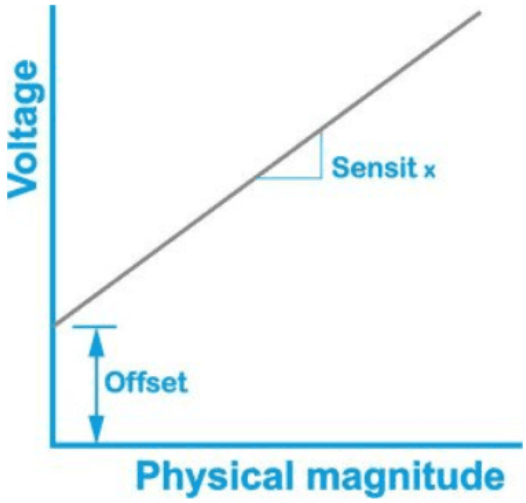


Fig. 71: **Linear relation of 2 variables**

In addition, users must indicate whether the **integer value is with sign or without sign**. That is, if the "**Signed**" box is:

- **Enabled**: Integer value with sign.
- **Disabled**: Integer value without sign.

The process of configuration has to be done using custom messages. This is to be configured in the *Custom Messages panel* of the **Input/Output menu**. The configuration will depend on the device in use and its communication protocol.

Decimal var sensor 0-1

In this panel, the user selects a real variable, this does not require a signal treatment. The process of configuration is similar to the one carried out when configuring a Integer Variable.

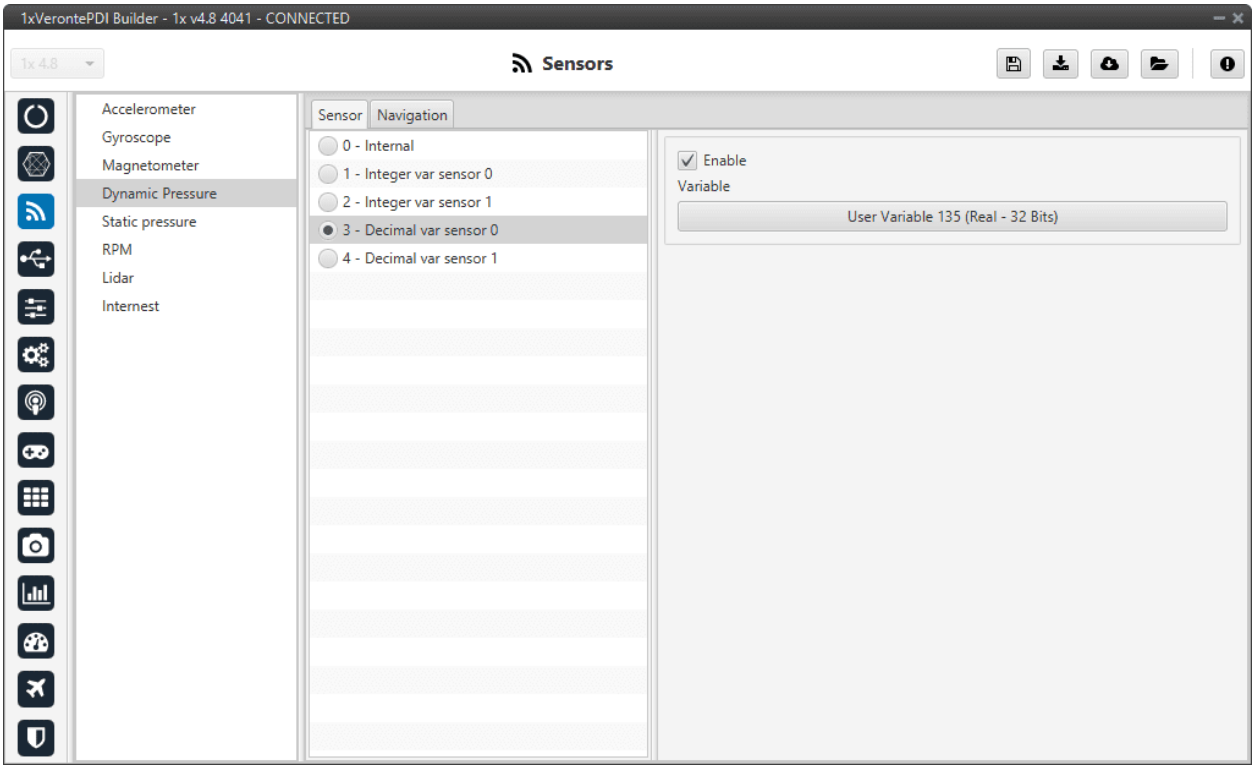


Fig. 72: **Dynamic Pressure panel - Decimal var sensor 0-1 configuration**

2.3.5 Static Pressure

Autopilot 1x has three pressure input lines, two for static pressure to determine the absolute pressure and one for pitot in order to determine the dynamic pressure.

This panel allows the user to configure a static pressure sensor input in 1x.

Important: In this panel, the user can only modify some configurable parameters, the selection of the static pressure sensor and the configuration of the **Variance** are done in the *Static Pressure Sensor - Sensors blocks* of the **Block Programs menu**.

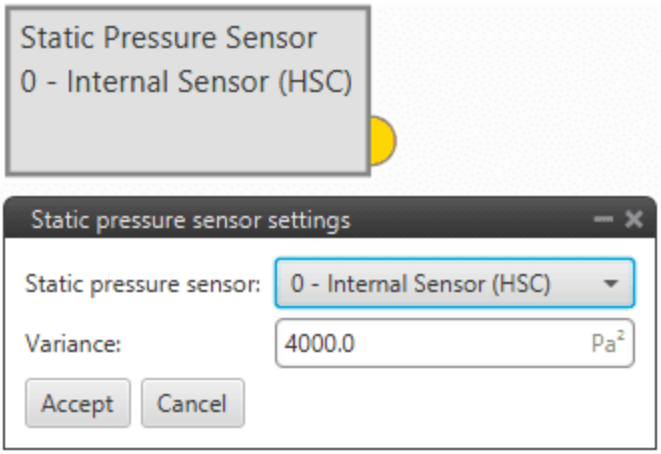


Fig. 73: **Static Pressure Sensor block**

2.3.5.1 Atmospheric calibration export

The **Atmospheric calibration export** allows, in a standard ground-air configuration, to continuously send information regarding the static pressure configured on the ground platform to the desired air platform. Therefore, it is a **configuration** that must be performed **on the Veronte Autopilot 1x ground unit**.

This function is **useful when making long flights** as the static pressure varies significantly throughout the day and therefore the altitude estimation will also vary.

> **Warning:** For correct operation both Veronte Autopilots 1x (ground and air units) must measure the same pressure at the same height (this must be checked).

This option is configured independently of the selected Static Pressure sensor.

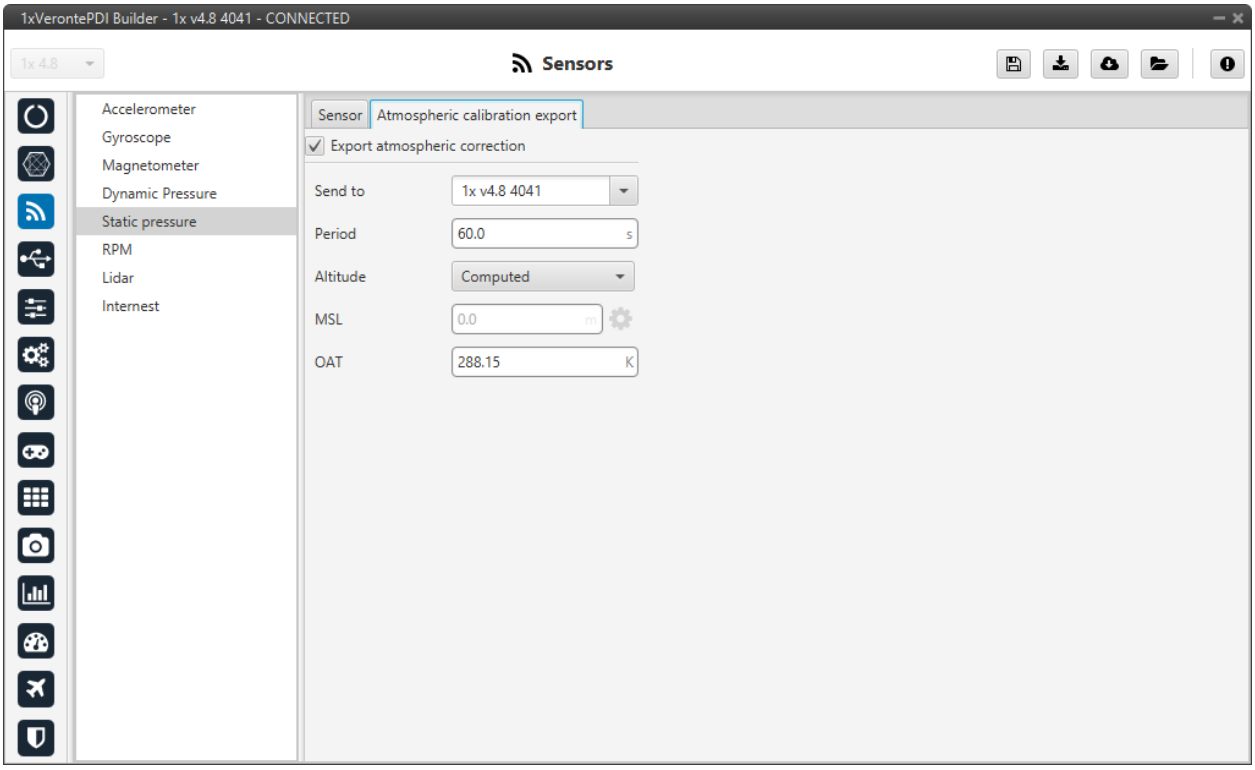


Fig. 74: **Static Pressure panel - Atmospheric calibration export parameters**

- **Export atmospheric correction**: Enables this feature.

This panel has 5 parameters that need to be configured:

- **Send to**: Points to the address where the correction will be sent. For more information on the available addresses, see List of addresses section of the **1x Software Manual**.
- **Period**: Period of time spent between sending each correction.
- **Altitude**: There are 2 options available for this parameter:
  - **External**: Users can manually enter the altitude if known.
  - **Computed**: By selecting this option, the altitude will be computed by the system.

> **Warning:** When this option is selected, it is necessary to add an **automation** in the 1x ground unit to calibrate the atmosphere **periodically** (see *Atmosphere Calibration action* of the **Automations menu**), to prevent the MSL from drifting.

- **MSL**: Can be entered manually or by a system variable.

> **Important:** This parameter is only enabled when the **External** option is selected as Altitude.

- **OAT**: Outside Atmospheric Temperature, to be defined by the user.

2.3.5.2 Sensor

The user can choose between 3 types of source:

- **Internal Sensor (HSC/MS56/DPS310)**: Autopilot 1x uses the internal sensor.
- **Integer var sensor 0-1**: Autopilot 1x uses a integer value provided by an external sensor.
- **Decimal var sensor 0-1**: Autopilot 1x uses a decimal value provided by an external sensor.

Hint: The following static pressure sensors are suggested:

- **Internal Sensor MS56**
- **Internal Sensor DPS310**

Internal Sensor HSC/MS56/DPS310

This panel displays the possible parameters that can be configured for the internal Static Pressure sensors.

Autopilot 1x has embedded 3 digital static pressure sensors: the **DPS310**, the **MS56** and the **HSC**. See more information on the pressure ports in Pressure lines - Hardware Installation of the **1x Hardware Manual**.

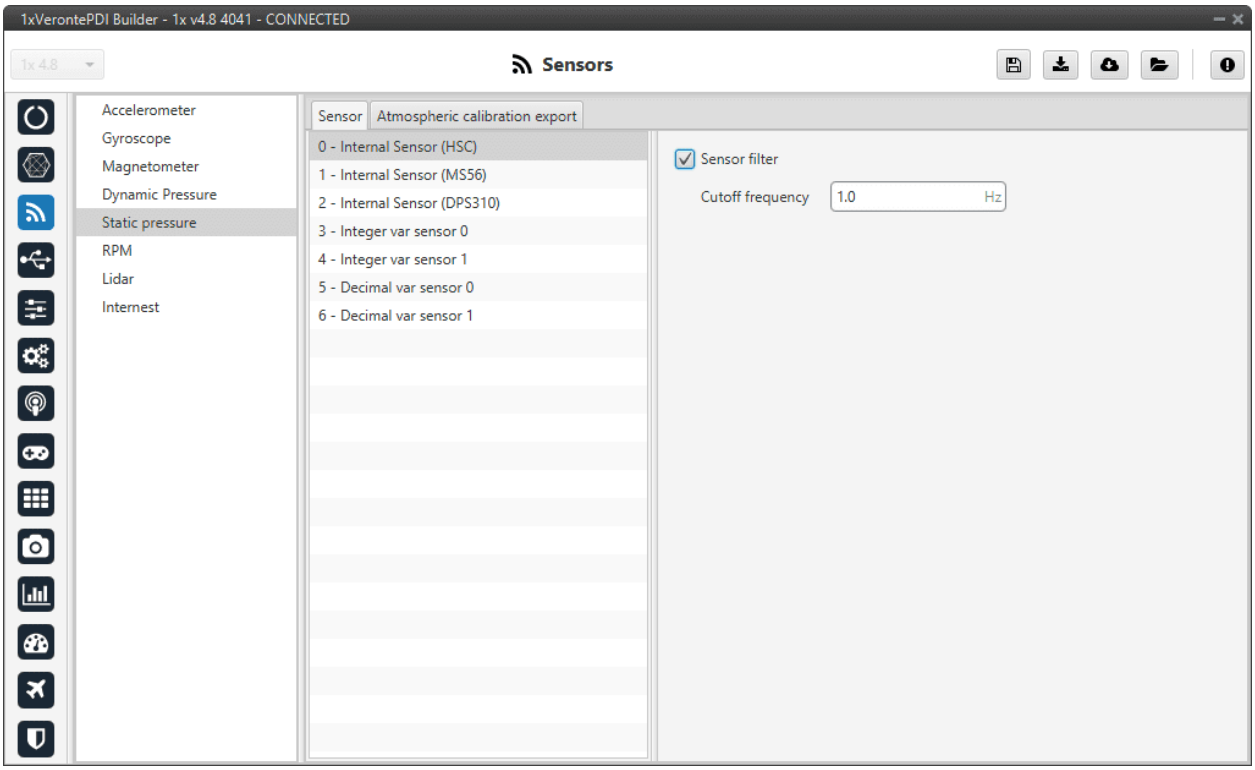


Fig. 75: **Static Pressure panel - Internal sensors configuration**

In this panel it is possible to set different options regarding filters.

- **Sensor filter**: Enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

Integer var sensor 0-1

In this panel it is possible to configure integer variables provided by an external sensor.

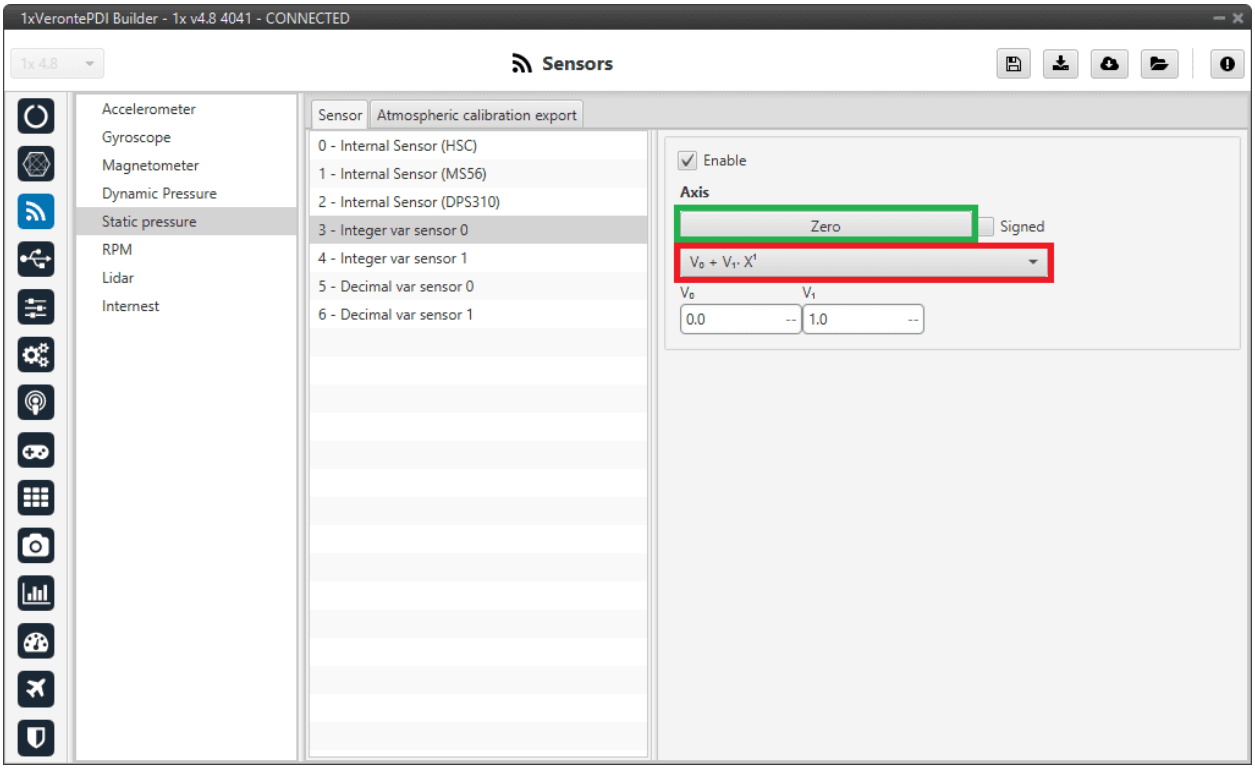


Fig. 76: **Static Pressure panel - Integer var sensor 0-1 configuration**

When Integer var sensor 1 or 2 are selected, the previous panel will be shown. In this panel, the user selects the variable that has been stored in a user variable (Green Box) and the operations that will be carried on (Red Box). It is possible **to use the signal through a linear or quadratic relation**. The following image shows an example of a linear relation:

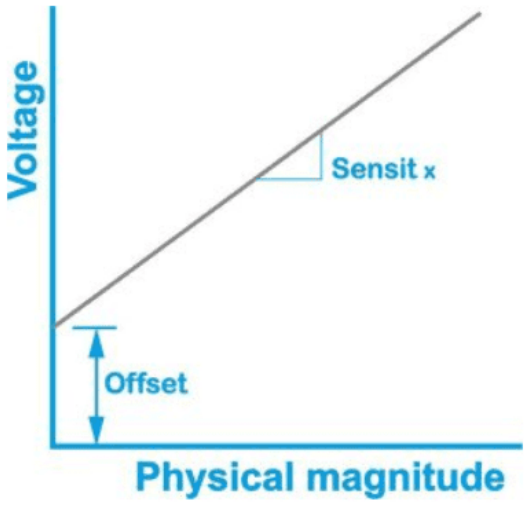


Fig. 77: **Linear relation of 2 variables**

In addition, users must indicate whether the **integer value is with sign or without sign**. That is, if the "**Signed**" box is:

- **Enabled**: Integer value with sign.
- **Disabled**: Integer value without sign.

The process of configuration has to be done using custom messages. This is to be configured in the *Custom Messages panel* of the **Input/Output menu**. The configuration will depend on the device in use and its communication protocol.

Decimal var sensor 0-1

In this panel, the user selects a real variable, this does not requiere a signal treatment. The process of configuration is similar to the one carried out when configuring a Integer Variable.

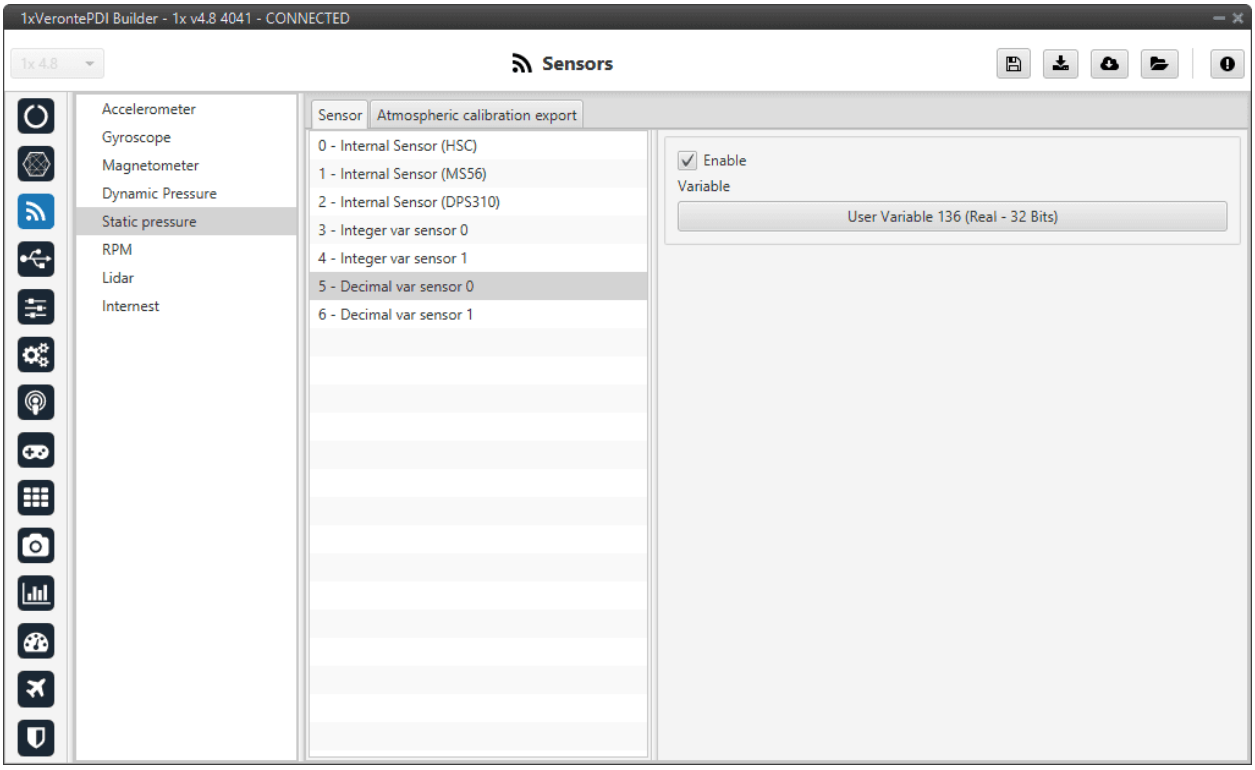


Fig. 78: **Static Pressure panel - Decimal var sensor 0-1 configuration**

2.3.6 RPM

Autopilot 1x can measure RPMs by reading from up to 6 input sources:

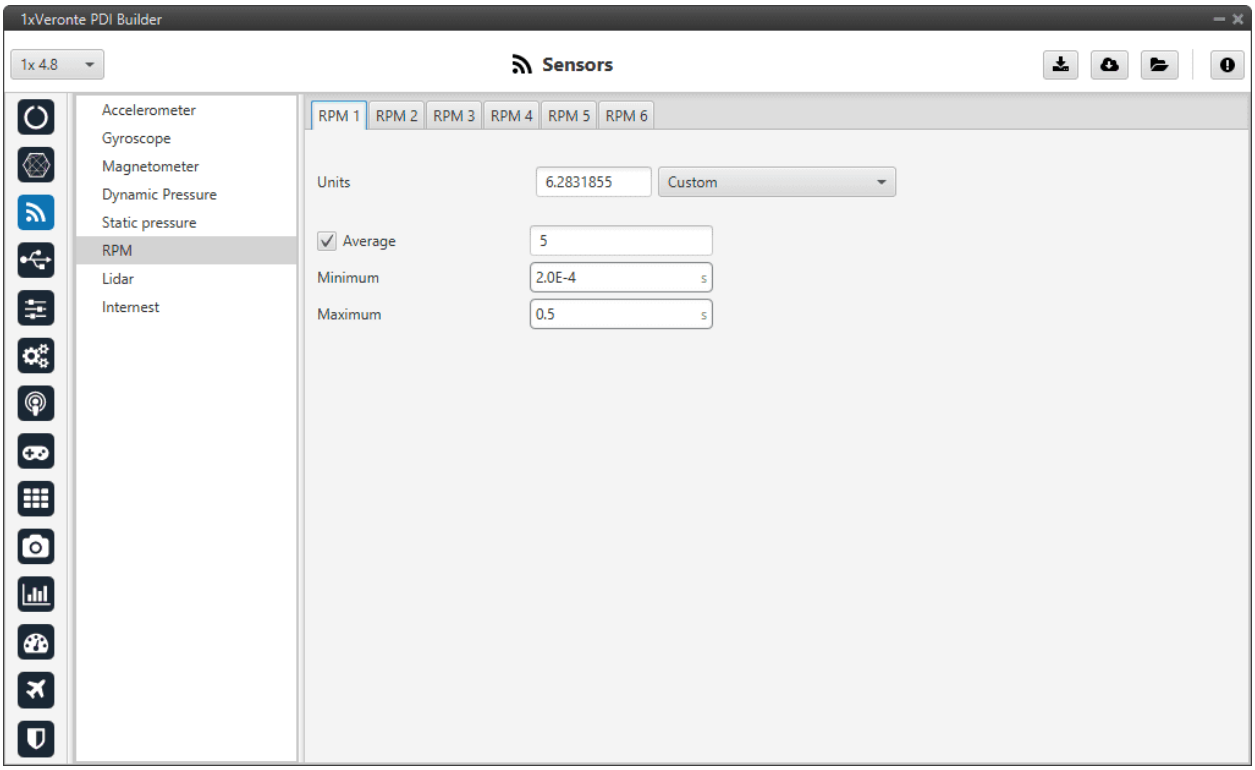


Fig. 79: **RPM panel**

- **Units**: Sensor conversion factor. It can be *Custom*, *Radians per pulse* or *Pulses per cycle*.
- **Average**: Filter to prevent voltage spikes. The readout of the pulse can be filtered as an average output. The amount of measurements to do the average needs to be specified.
- **Minimum**: Here the minimum expected pulse period needs to be specified. This will discard spurius pulses (e.g. induced by EMI) which are smaller than this minimum pulse.
- **Maximum**: This is the maximum period of time allowed without capturing. If no incoming pulse is received for more than this time, the output RPMs will be 0.

2.3.7 Lidar

The I2C bus allows the connection of several devices with different addresses to the same line via master-slave communication. At this moment, Autopilot 1x supports the following Lidar devices:

- **Garmin Lidar Lite v3**: Optical distance measurement sensor with a range of 5 cm to 40 m.
- **SF11 Lidar**: Long range laser altimeter. Supported SF11/B and SF11/C with a range of maximum 50 m and 0.2 m to 120 m respectively.
- **SF20 Lidar**: OEM laser altimeter module. Supported SF20/C with a range of 0.2 m to 100 m.

1x allows up to 5 Lidar devices to be connected to the system at the same time. The configuration panel can be seen below:

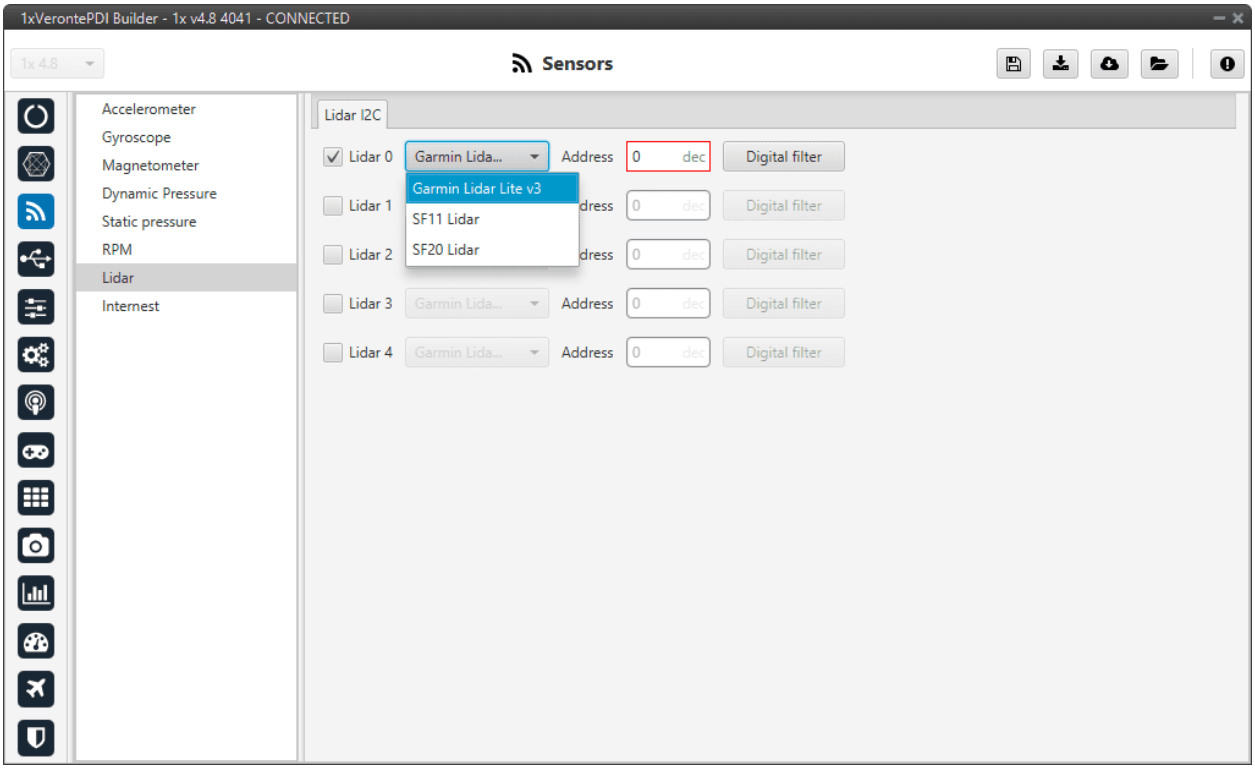


Fig. 80: **Lidar panel**

After enabling the needed number of Lidar devices, configurable parameters are:

- **Type of Lidar**.
- **Address**: With an accepted value between 16 - 239, this is the origin address from the Lidar being configured.
- **Digital filter**: By enabling the **Sensor filter** checkbox, the user enables a low pass filter which its **cutoff frequency** is configured manually, allowing the user to input any desired value in Hz. It is a **software filter**.

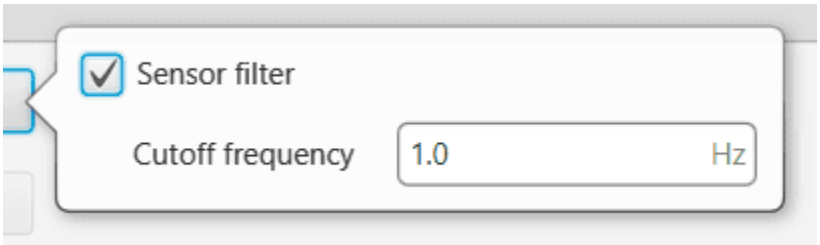


Fig. 81: **Lidar panel - Digital filter**

Warning: I2C address will be different for different devices. Make sure to define it properly by checking the manufacturer documentation.

Note: The Lidar number (Lidar 0/4) needs to be kept in order to properly configure the Altimeter later (this must be

done in the *Altimeter - Sensors blocks* of the **Block Programs menu**).

2.3.8 Internest

An **ultrasound sensor** computes Veronte Autopilot 1x position by measuring the time the signal sent out takes to return. The following panel together with *Relative position sensor block* (see *Relative position sensor - Block Programs* section) allows the user to configure an **Internest system** with Veronte Autopilot 1x.

This panel allows the user to choose the version of Internest to be used, its range and the rotation matrix:

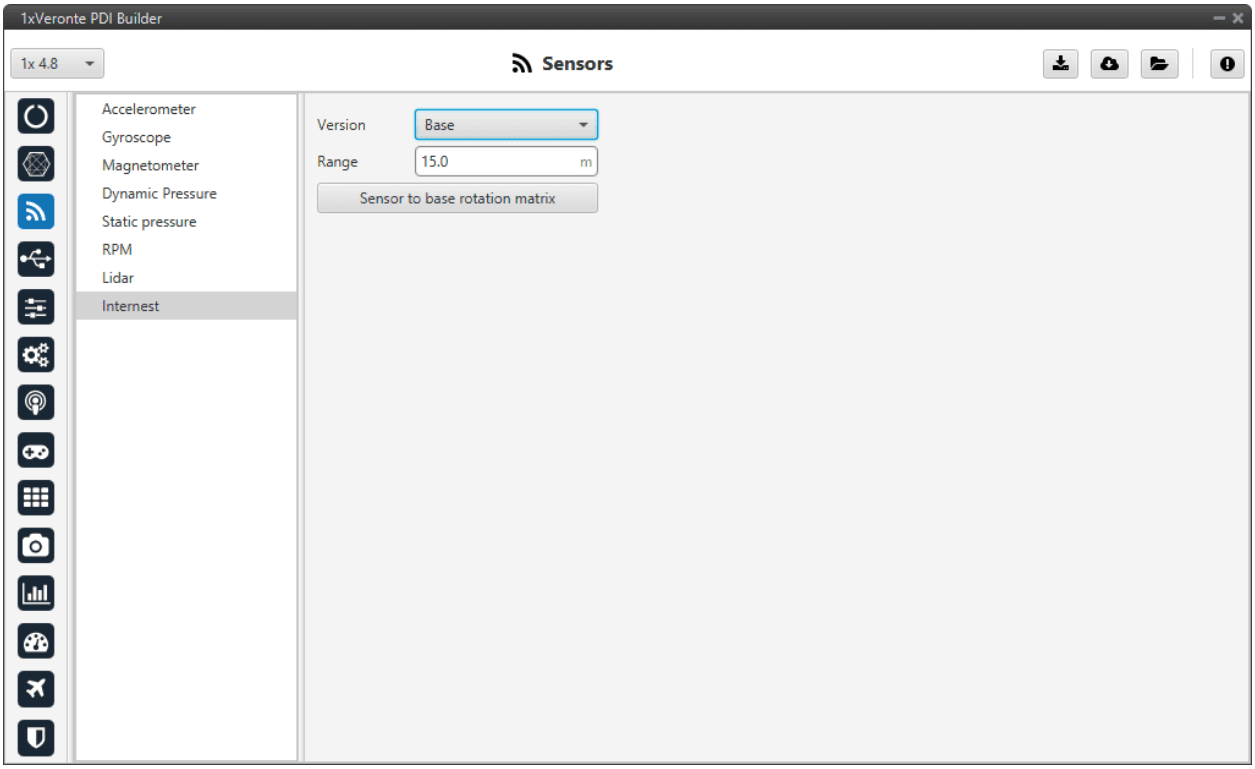


Fig. 82: **Internest panel**

- **Version**: Users must choose the version of the Internest system, the available options are **Base** and **Explore**.
- **Range**: Defines the distance at which Internest values will start to be valid.
- **Sensor to base rotation matrix**: Matrix to rotate the system to match the Veronte Autopilot 1x coordinate system.

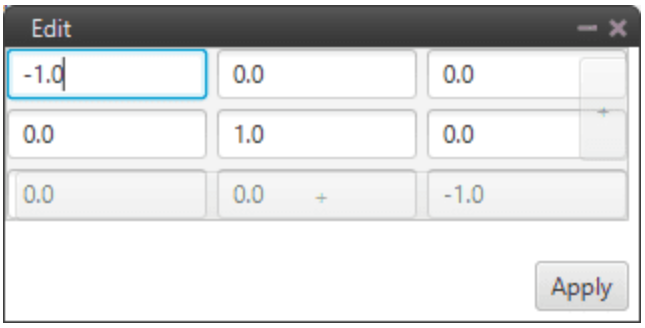


Fig. 83: **Internest panel - Rotation matrix**

2.4 Input/Output

This section of the manual contains the information about external sensors/devices configuration. These devices are configured on the different ports available in **Veronte Autopilot 1x**:

Port/Manager	Description
I/O Setup	Configuration of serial port connections, Serial Custom Messages, etc.
CAN Setup	Configuration of the two CAN buses (A and B), CAN Custom Messages and Mailboxes
Digital Input	Configuration of PPM signals, pulses or RPM sensors
Serial	Configurable Veronte LOS, RS232 and RS485 serial ports

As **Custom Messages** need to be defined for both serial and CAN communication, there will be a **specific section** for this after the CAN Setup section ⇒ *Custom Messages types*.

2.4.1 I/O Setup

In this panel the user can establish the relationship between a determined signal with a I/O port. This allows users to configure external sensors, messages between 1x units (Tunnel) and custom messages.

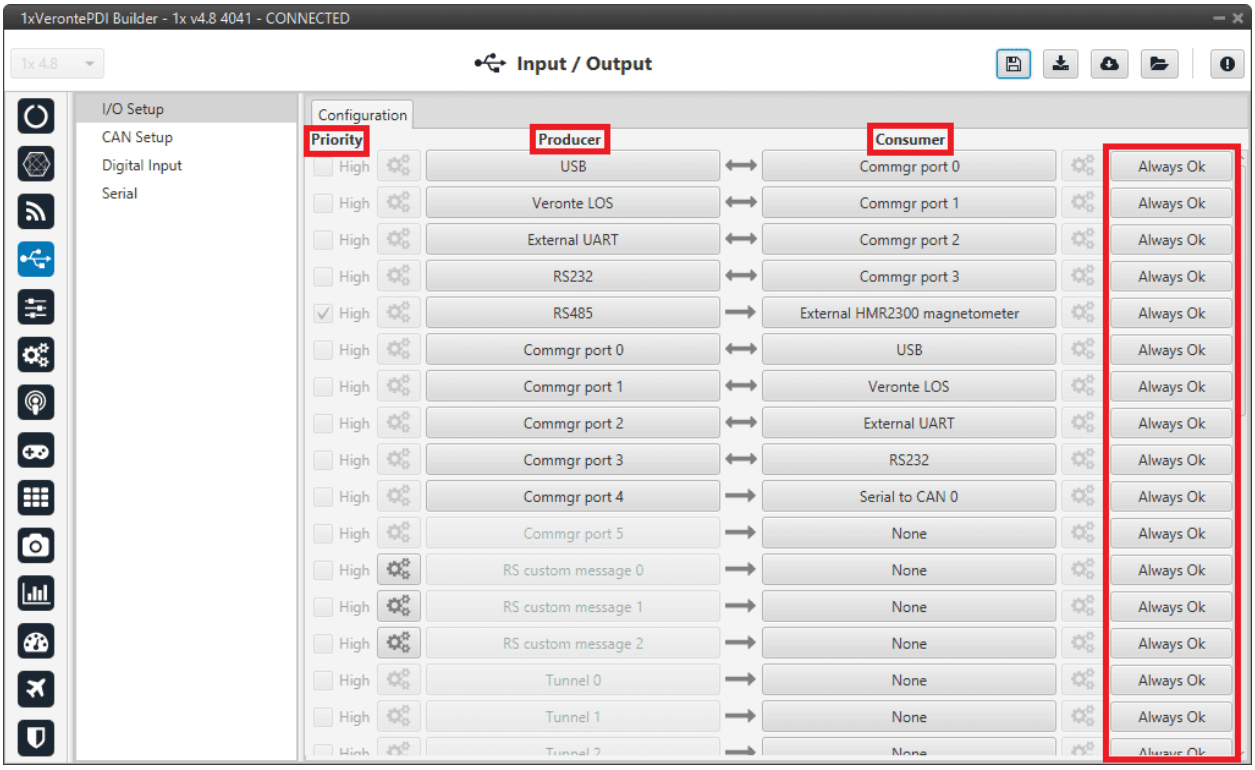


Fig. 84: **I/O Setup panel**

- **Priority**: Connections between I/O ports can be marked with **high priority** with this checkbox. If enabled, they will **run at high frequency: 1000 Hz**.

  The following table shows which connections can be **selected** as **high** or **low priority** (marked with **S**) and which connections are always marked as **high** (marked with **H**), so they cannot change to **low priority**.

Product	USB	Vehicle LOS	External UART	RS232	RS485	Commgr port 0-5	RS custom message 0-2	Tunnel 0-2	GPS 0-1 RTCM	External HMR2300 magnetometer	Y splitter 0-2	Iridium	Vectornav VN-300	Unescape port	Serial to CAN 0-5	CANNMEA unwrapper 0-1	Vehicle wrapper	Verone LTE	Verone LTE Auxiliary	External ultrasound
USB	S	S	S	S				S	H	S	S	H	S	S	S	S	S	S	S	H
Verone LOS	S		S	S	S				S	H	S	S	H	S	S	S	S	S	S	H
External UART	S	S		S	S				S	H	S	S	H	S	S	S	S	S	S	H
RS232	S	S	S		S				S	H	S	S	H	S	S	S	S	S	S	H
RS485	S	S	S	S					S	H	S	S	H	S	S	S	S	S	S	H
Commgr port 0-5																				
RS custom message 0-2																				
Tunnel 0-2	S	S	S	S	S				S	H	S	S	H	S	S	S	S	S	S	H
GPS 0-1 RTCM	S	S	S	S	S				S	H		S	H							H
External HMR2300 magnetometer	S	S	S	S	S				S			S	H		S					H
Y 0-2 splitter A-B	S	S	S	S	S				S	H	S	S	H	S	S	S	S	S	S	H
Iridium	S	S	S	S	S				S	H	S		H		S	S				H
Unescape port	S	S	S	S	S				S	H	S	S	H		S	S	S	S	S	H
CANS to serial 0-5	S	S	S	S	S				S	H	S	S	H	S	S	S	S	S	S	H
CANS wrapper 0-1	S	S	S	S	S				S	H	S	S	H	S	S	S	S	S	S	H
Verone LTE	S	S	S	S	S				S	H	S	S	H	S	S	S	S		S	H
Verone LTE Auxiliary	S	S	S	S	S				S	H	S	S	H	S	S	S	S	S		H

- **Producer**: Functions for creating and sending messages.
- **Consumer**: Functions for receiving and parsing messages
- **Bit**: This assigns each connection a bit in a way that allows this connection to be activated/deactivated depending on the status of the selected bit.

  By default, the 'Always Ok' bit is set to all connections so that they are always active.

The following are the steps to setting up reception or transmission between ports:

1. Choose the **Producer** to use.

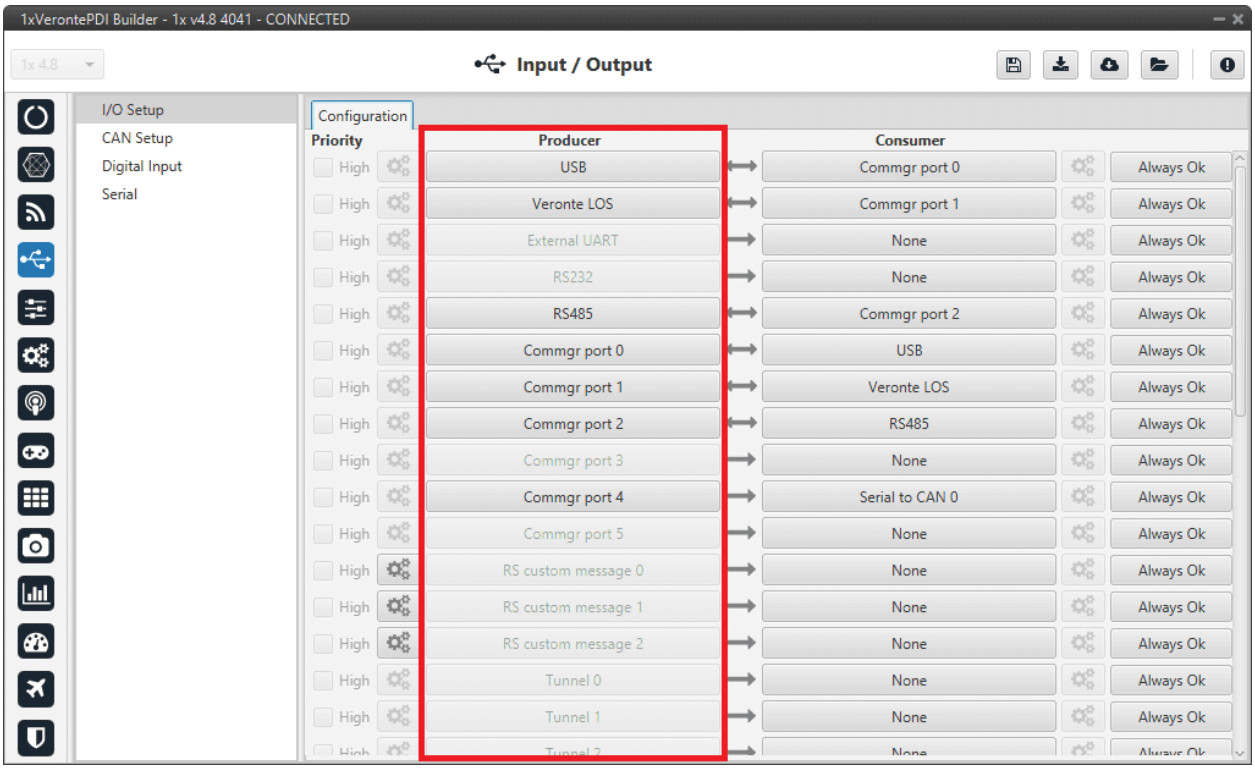


Fig. 85: **I/O Setup panel - Producers**

2. To configure the desired **Consumer** that will be bind to the chosen **Producer**, it is first required to establish the **relationship** between them:

   - **Bind** $\rightarrow$: Unidirectional relationship.
   - **Bind Bidirectional** $\leftrightarrow$: Bidirectional relationship. This enables a port to receive or send information.

Note: Once the **Consumer** has been selected, it is possible to undo the selection by pressing the **Clear** button.

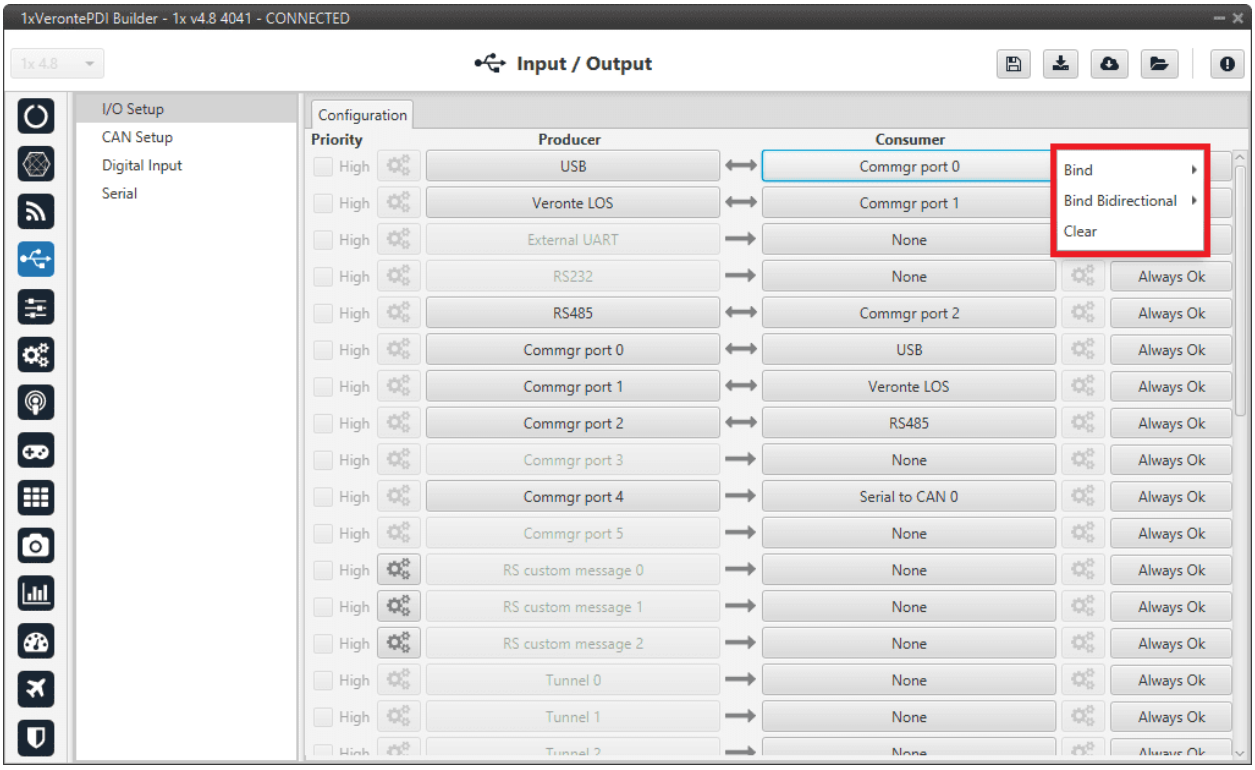


Fig. 86: **I/O Setup panel - Consumer options**

3. Select the desired **Consumer** element.
4. *(optional)* Configure the **Bit** parameter.

The following I/O ports are available:

Field	Description
USB	USB Port
Veronte LOS	Radios
External UART	External UART Port (UART A pin) For more information, see Pinout - Hardware Installation section of the **1x Hardware Manual**
RS232	Serial Port 232
RS485	Serial Port 485
Commgr port	COM Manager ports send and receive VCP messages. VCP is the protocol used by Veronte products to communicate. For more information on this, read the VCP user manual
RS Custom Message	This allows user to send/receive a serial custom message, see *Serial Custom Messages*
Tunnel	Creates a bidirectional brigde between two devices, see *Tunnel*
GPS RTCM	This allows the user to send/receive RTK information from GND unit to AIR unit
External HMR2300 magnetometer	External magnetometer sensor, see the *Magnetometer Honeywell HMR2300 - Integration examples* section
External ultrasound	External ultrasound sensor, see the *Internest - Sensors* section
Y Splitter	Used to split a signal into 2
Iridium	Iridium communication, see the *Iridium - Communications* section
Vectornav VN-300	Vectornav VN-300 is an external IMU. For more information, see the *Vectornav VN-300 - Integration examples* section
Unescape port	This allows user to reconstruct a byte stream with an escape logic, see *Unescape port*
CAN to serial / Serial to CAN	**Serial to CAN** sends serial streams over a CAN Bus / **CAN to serial** undoes the transformation 'Serial to CAN'
CAN wrapper / CAN unwrapper	**CAN wrapper** sends CAN streams over a serial Bus / **CAN unwrapper** undoes this transformation, see the *CAN wrapper/CAN unwrapper* section
NMEA Parser	NMEA 0183 messages parser, see *NMEA Parser*
Veronte LTE	4G Connection. For more information, see the *4G - Communications* section. **Warning:** The configuration for a Veronte Autopilot 1x `hardware version 4.8` is different than for one with `hardware version 4.5`. If the user has any questions, please contact the support team following the user's Joint Collaboration Framework
Veronte LTE Auxiliary	**Warning:** Reserved port

More information about some elements can be found in the following sections.

2.4.1.1 Serial Custom Messages

Warning:

- Autopilot 1x has a **serial limitation** of **64 vectors** (fieldset) per Custom.

  In addition, there is a limit shared with all Customs, including **CAN Custom Messages**:

  - Maximum number of **vectors** (fieldset): **104**
  - Maximum number of **fields**: **2000**

It is possible to configure the messages sent/received through the serial port and its conversion to system variables by selecting the option **RS Custom message** and configuring the I/O port.

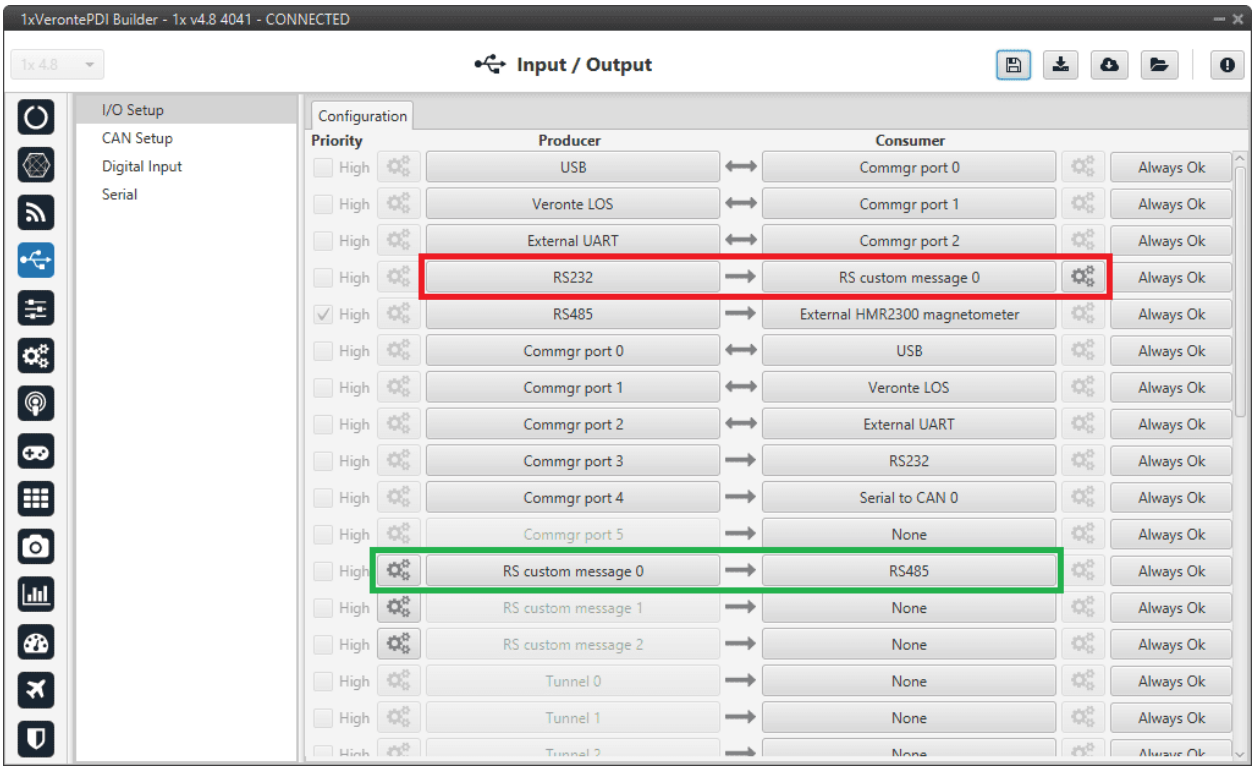


Fig. 87: **Serial Custom Messages**

In the image above two possible configurations using an RS Custom Message can be seen. The 'red' one is configured to receive a given message from an RS232 serial port and the 'green' is used to send a RS Custom Message through an RS485 serial port. It is also possible to use the same RS Custom Message for both tasks if the bidirectional relationship is selected (the arrow indicates this (Bind Bidirectional ↔)).

To configure a RS Custom message, the user must follow the next steps:

1. Press the **configuration button** ( icon) and a pop-up window will be displayed.

   In this window press the  icon to add a custom message, the user can choose between **System variables**, **ADSB Vehicle** or **External Sensor**.

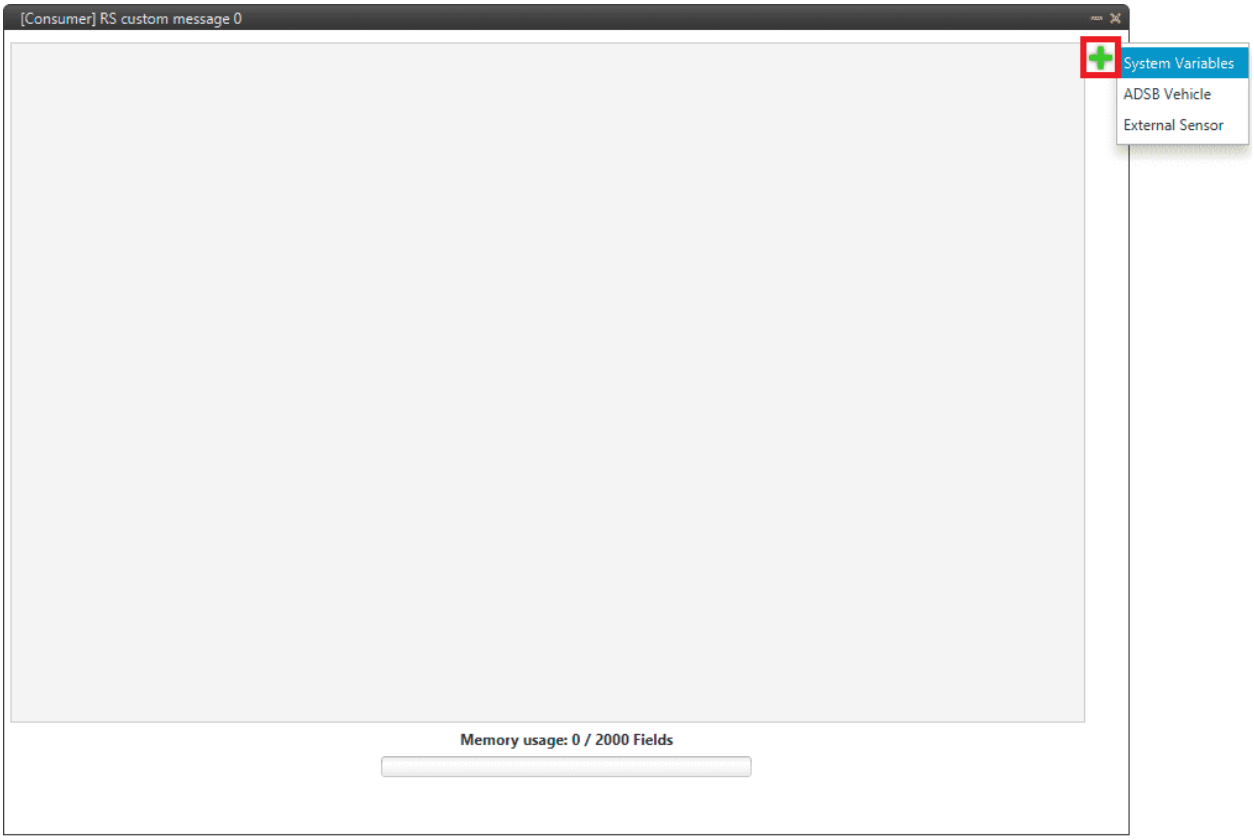


Fig. 88: **Serial Custom Message configuration**

Note: The difference between choosing **System Variables**, **ADSB Vehicle** or **External Sensor** is that when the user selects **Variable** as the custom message type, only system variables will appear when *System Variables* is selected, only ADSB variables when *ADSB Vehicle* is selected and only variables related to external sensors if *External Sensor* is selected.

---

2. When it is already added, the following options are available to configure a custom message:

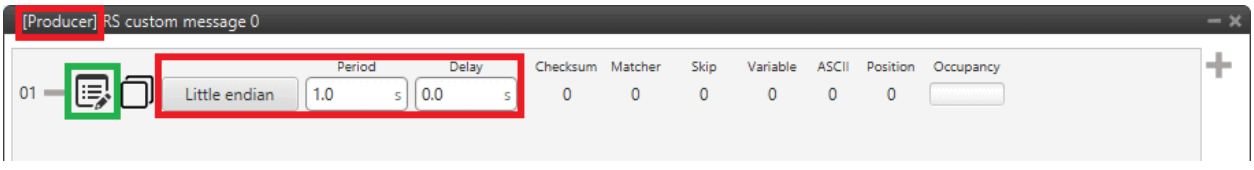


Fig. 89: **Producer RS Custom Message configuration**

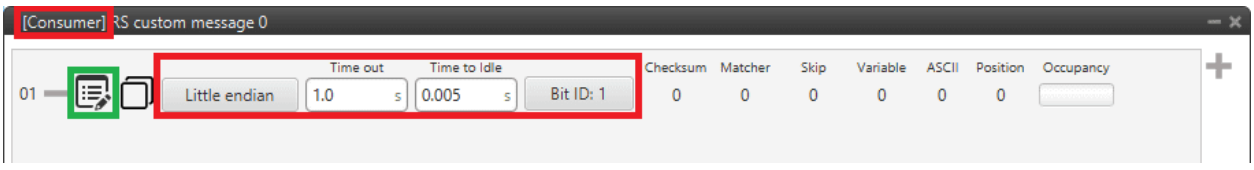


Fig. 90: **Consumer RS Custom Message configuration**

- **Endianness**: Depending on the order in which the device issue the message, it is possible to select:
  - **Big endian**: Set the value from left to right.
  - **Little endian**: Set the value from right to left.
  - **Mixed endian**: Some devices use this format. If users need to configure it, please contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).
- **Period/Time out**: This option has a dual role depending on if it is used to transmit or receive data.
  - **Period - Producer**: It is the inverse of the send frequency.
  - **Time out - Consumer**: This is the threshold time between receptions to consider that the message is not being received correctly.
- **Delay/Time to Idle**: The purpose of this feature is ensuring half-duplex communication. This option has a dual role depending on if it is used to transmit or receive data.
  - **Delay - Producer**: After sending the message, the Serial bus will be disabled for this amount of time in order to allow half-duplex communication.

    Warning: Delay time must not be longer than the specified period, since this will result in out-of-order message sending.

  - **Time to Idle - Consumer**: This is the time Autopilot 1x waits before discarding partially parsed bytes.
- **Bit ID**: This option is only available when a message is configured as **Consumer**. The user bit selected in Bit ID box will be true if the message is being received correctly.

  Warning: Pay attention that the user bit selected in **Bit ID** is not in use for another task.

3. To create the structure of the message, click on the **edit message button** ( icon) and then press the  icon to add fields to it. The following type of messages are available to configure a structure: **Variable**, **Checksum**, **Matcher**, **Skip**, **Parse ASCII** and **Position**.

   The configuration of each structure is covered in the *Custom Messages types - Input/Output* section of this manual.

Warning: Before configuring any message, user has to know the structure it has to have according to the device that is connected to the port. Each device may have a different message structure when it sends or receives information.

To check serial messages transmission, see the *Debug serial messages transmission - Troubleshooting* section of this manual.

2.4.1.2 Tunnel

It is possible to configure a Tunnel, which is a bidirectional bridge between 1x units that communicate to each other sharing information about an external device connected to the Serial or Digital port.

Imagine that it is desired to have a button connected to the 1x air unit to launch a parachute. It is not possible to physically connect the button because the air autopilot is in the flying platform, so a different option is needed. Here is where the tunnel becomes useful. The button could be connected through the Serial or Digital port to the 1x ground unit, and then the signal is sent through the tunnel to the air unit. With this configuration it would be like if the button were physically connected to the aircraft.

Let's consider the following image:

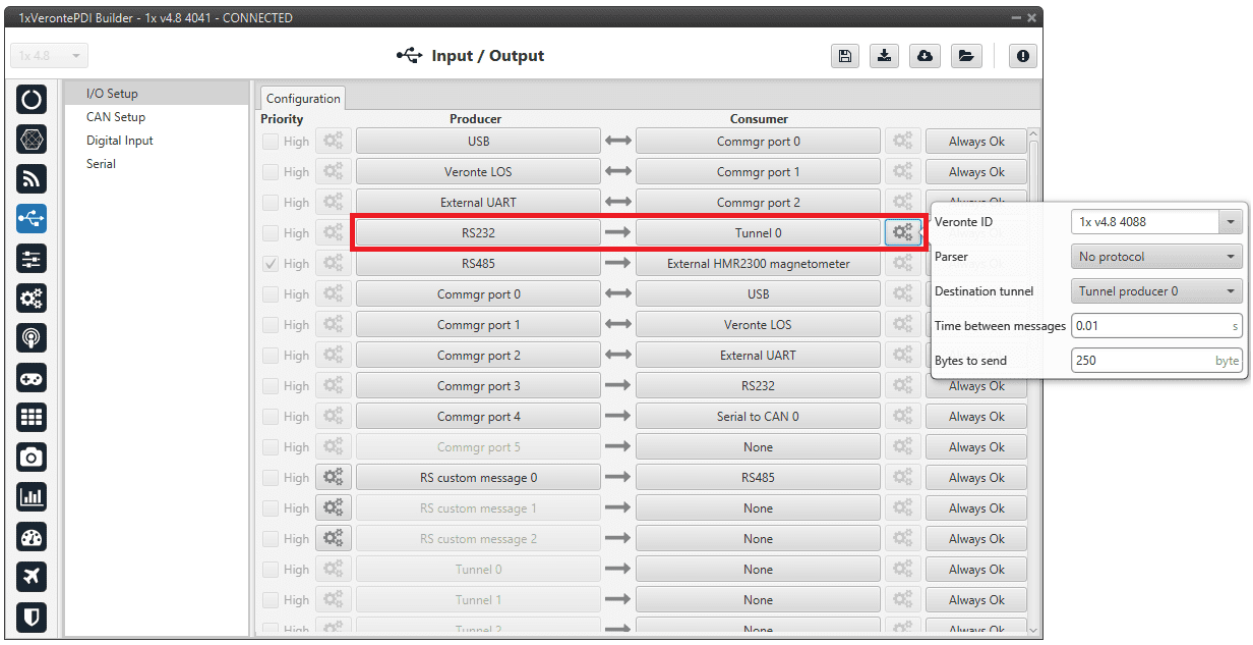


Fig. 91: **Tunnel configuration**

In the image above there is a device connected to the **RS232 (Producer)** and there is a **Tunnel (Consumer)** which sends that information to other Autopilot 1x with a determined ID. On the other hand, **1x air unit** has to be configured to receive the signal sent by other device. In that case the **Producer** will be **Tunnel** and **Consumer** will be the **port or destination tunnel where the device is connected**.

The options available when configuring **Tunnel as Consumer** are:

- **Veronte ID**: Enter the address that will receive the information. The following options are the most common:
  - **App 2**: Veronte applications address.
  - **Broadcast**: All units on the network. Select this option for a generic configuration.
  - Address of a specific Veronte unit, it can be a 1x, a 4x, a CEX, etc.

    For more information on the available addresses, see List of addresses section of the **1x Software Manual**.
- **Parser**: The user can choose protocol to parse message data. The options available are:
  - No protocol
  - RTCM3

  - CANserial

- **Destination tunnel**: Number of port is used to avoid mistakes and identify a Tunnel when using more than one, *Tunnel 0, 1 and 2* are available.
- **Time between messages**.
- **Bytes to send**: Sets the message size to send.

When configuring **Tunnel as Producer** (i.e. on the unit that receives the information), no configuration is required. It is only necessary to connect it to a Consumer, usually to a serial port.

2.4.1.3 Unescape port

To understand what unescape is, the user must first understand what an **escape byte** is.

Let's consider that there is a protocol that defines a '**Flag**' as the start and end byte of the frame. In case the flag or an escape value appears in the frame data, and in order not to misinterpret the message, an escape byte or the same value repeated will be added before them so that, at the time of parsing, it will be reconstructed with the original byte.

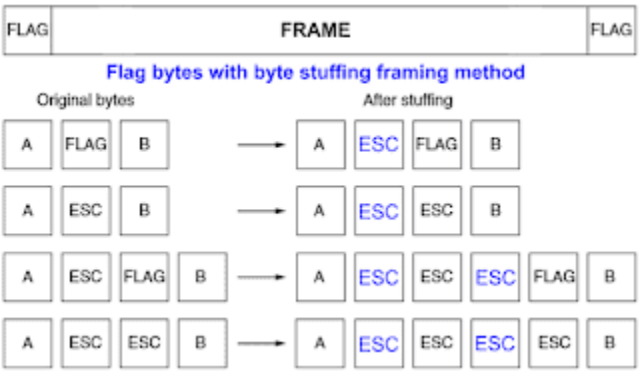


Fig. 92: **Escape byte**

In **1x PDI Builder**, an Unescape port has been implemented to allows to reconstruct a byte stream with an escape logic.

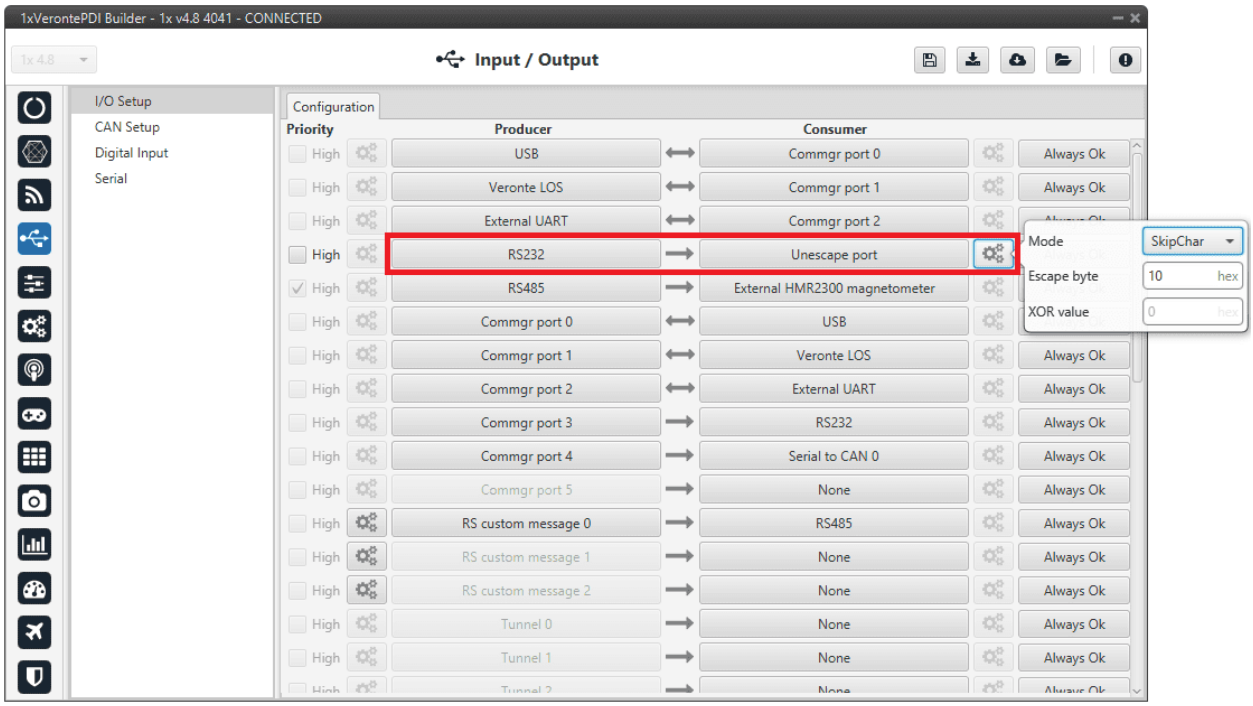


Fig. 93: **Unescape port configuration**

Two modes of escapes are supported:

- **SkipChar**: The unfolding of the value to be escaped, the byte to escape has been repeated in the message.
- **SkipAndXOR**: In this case, the escape byte is entered first and then the value to escape (i.e. the value to escape XOR escape byte).

In addition to this, two more options are available in this pop-up window:

- **Escape byte**: Escape byte added.
- **XOR value**: Only available when '**SkipAndXOR**'option is selected.

2.4.1.4 NMEA Parser

NMEA Parser is another way to add an external GNSS device. This consumer allows to receive NMEA 0183 messages and parses them directly. The NMEA Parser configuration menu includes the following parameters:

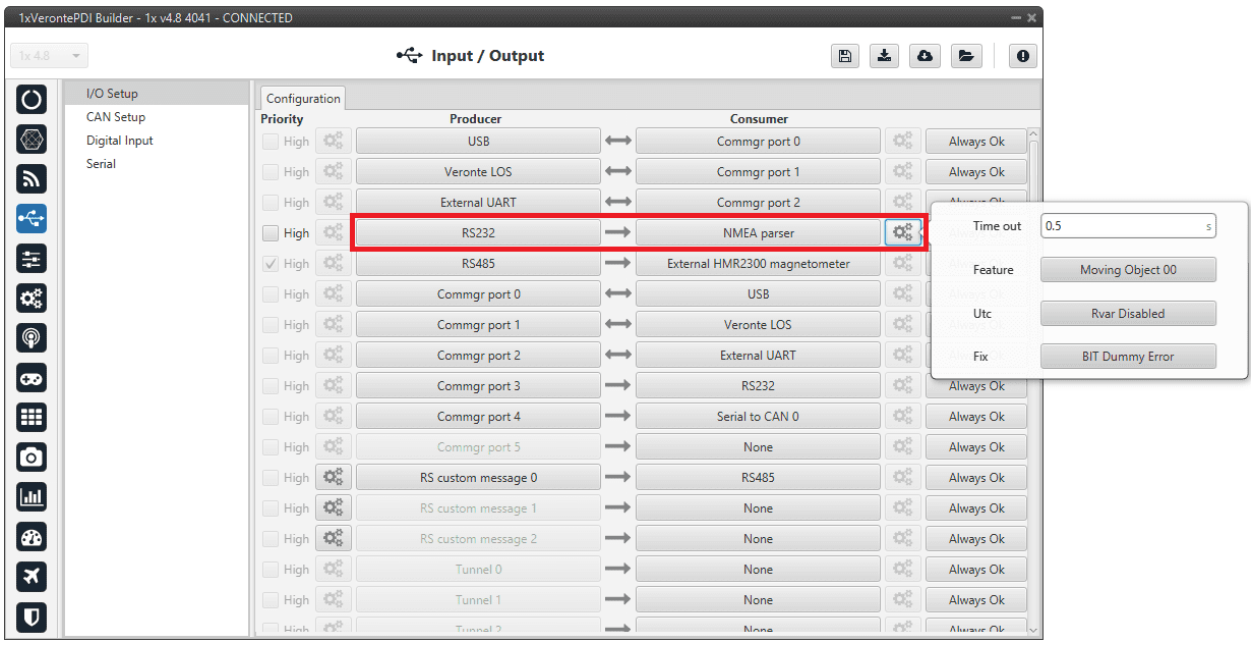


Fig. 94: **NMEA Parser configuration**

- **Time out**: Defines the period of incoming information from the external system.
- **Feature**: Variable extracted from the message defining the GNSS position. Usually Moving Object variables are used.
- **Utc**: Variable extracted from the message defining the UTC.
- **Fix**: Data provided by the external device which is important to know the status of the positioning.

Once the NMEA message has been parsed, the variables used for **Fix** and **Feature** can be selected in the GPS External configuration of the GNSS block as **Fix Bit** and **GPS Position**. For more information about this configuration, see *Sensors blocks - Block Programs* section of this manual.

2.4.1.5 CAN wrapper/CAN unwrapper

The CAN wrapper and CAN unwrapper ports allow CAN communication via serial. For this, **CAN wrapper** creates a message with a specific sequence which ensures the transmission of CAN message data; and **CAN unwrapper** undoes this transformation to a regular CAN message.

The structure of the messages sent/received by the CAN wrapped and CAN unwrapped respectively is the following:

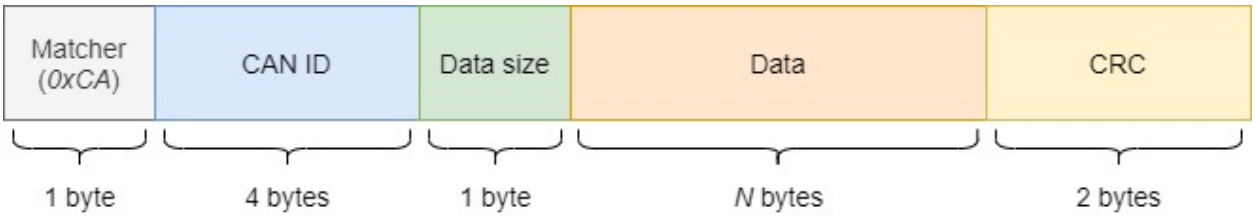


Fig. 95: **CAN wrapper/unwrapper - Message structure**

Content	Size
Matcher: *0xCA*	1 byte
CAN ID	4 bytes
Data size	1 byte
Data	*N* bytes
CRC	2 bytes

Note: For further details on how to use these ports, consult the *CAN message transmission via serial - Integration examples* and the *CAN message reception via serial - Integration examples* sections of the present manual.

2.4.2 CAN Setup

A CAN (Controller Area Network) Bus is a robust vehicle bus standard widely used in the aviation sector. Autopilot 1x is fitted with two CAN buses that can be configured independently.

The structure of a CAN message can be seen in the following image:

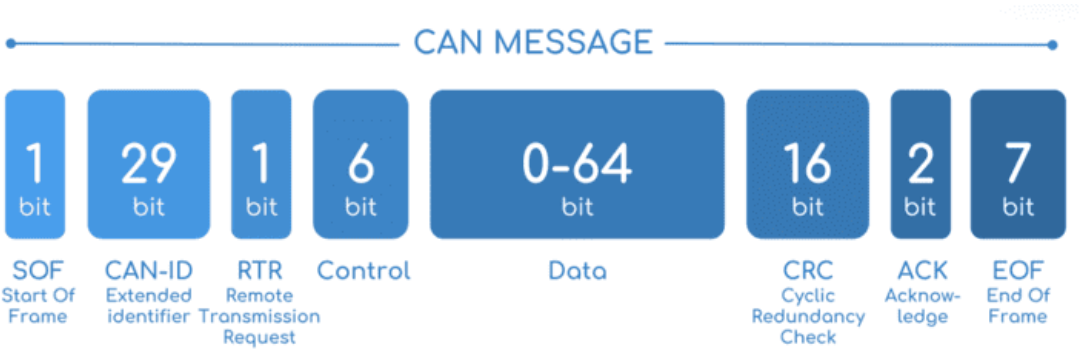


Fig. 96: **CAN message structure**

Only the ID is introduced in the system, the rest of the message layout is already coded. The data field is the one build by the user to send, and parsed when received.

The baud rate of both CAN buses can be configured in the *Mailboxes panel*.

The steps to be followed from the moment a CAN message arrives at or is sent from the Autopilot 1x are described in the *CAN communication - Integration examples* section of this manual.

2.4.2.1 Configuration

This panel allows the configuration of communications between different devices.

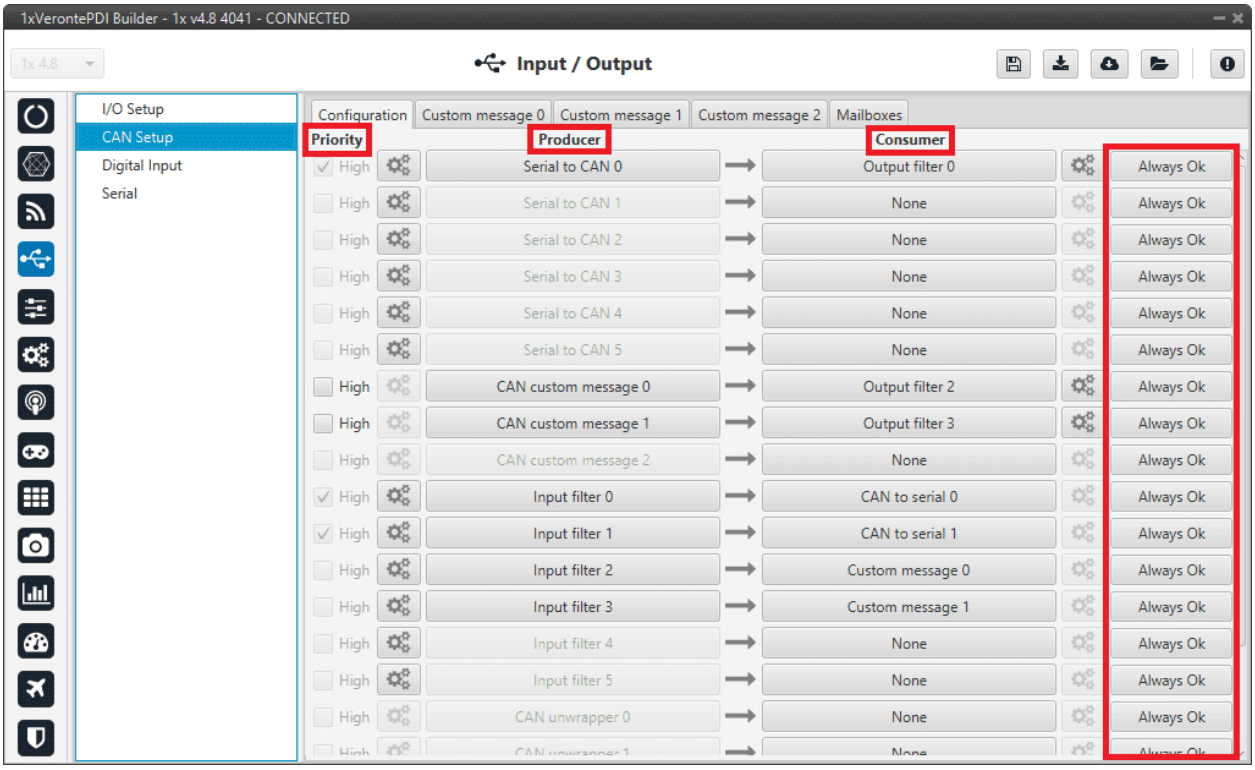


Fig. 97: **Configuration panel**

In this panel, the user can find the same 'columns' (**Priority**, **Producer**, **Consumer** and **Bit**) as in the *I/O Setup panel*. In addition, the process for configuring producers and consumers is also the same as described in the *I/O Setup - Input/Output* section.

Regarding the **Priority** option, users should note:

- Connections that are automatically configured as **high priority** (checkbox marked) and cannot change to **low priority**:
  - **Input filter 0-5 → CAN to serial 0-5**
  - **Serial to CAN 0-5 → Output filter 0-5**
- Connections that **can be enabled** as **high priority** (checkbox marked):
  - **CAN custom message 0-2 → Output filter 0-5**
  - **CAN custom message 0-2 → CAN wrapper 0-1**
  - **Input filter 0-5 → Output filter 0-5**
  - **Input filter 0-5 → CAN wrapper 0-1**
  - **Input filter 0-5 → CAN 4x**
  - **CAN unwrapper 0-1 → Output filter 0-5**
  - **CAN GPIO remote 0-1 → Output filter 0-5**
  - **CAN GPIO remote 0-1 → CAN wrapper 0-1**
  - **CAN 4x → CAN to serial 0-5**
  - **CAN 4x → Output filter 0-5**

  - **CAN 4x → CAN wrapper 0-1**
  - **CAN 4x → CAN 4x**

For some connections the priority option is not available due to computational burden.

> **Warning:** In CAN, while the specified period is not guaranteed in the Low state, in the High state it is.
>
> However, only those **messages that are critical for external devices** should be set as **high priority**, as this may disrupt the proper functioning of Veronte Autopilot 1x.

On the one hand, Autopilot 1x has the **producers** shown below:

- **Serial to CAN**: Serial messages over CAN output, it has to be connected to *I/O Setup* **consumer**. It can be configured in the **configuration button** ( icon), a pop-up window will appear:

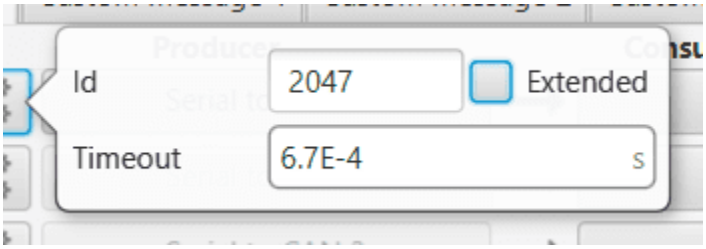


Fig. 98: **Serial to CAN configuration**

  - **Id**: CAN Id must be set and it is used to identify messages. The value set has to be **decimal format**.
  - **Extended**: If enabled, the frame format will be this, 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
  - **Time out**: This is the threshold time between receptions to consider that it is not being received correctly.

- **CAN custom message**: CAN custom messages transmission. They are configured in the Custom message tabs, explained in *Custom Messages* section.
- **Input filter**: CAN input filters. Those CAN messages received in one filter can no longer be received in subsequent filters. Input filter must be configured in the **configuration button** ( icon), a pop-up window will appear:

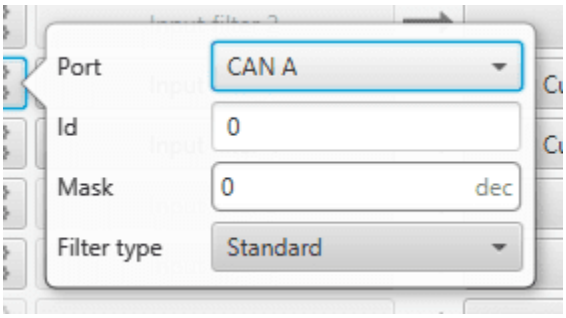


Fig. 99: **Input filter configuration**

  - **Port**: It is required to configure the CAN bus from which it listens, the user can choose between *CAN A*, *CAN B* or *BOTH*.

  - **Id**: CAN Id must be set and it is used to identify messages. The value set has to be **decimal format**.
  - **Mask**: Here a CAN Id mask can be set to filter messages. **The mask marks which bits of the message id (in binary) are matched**.

    For example, to admit standard Ids (11 bits) from 8 to 11 (100 to 111 in binary) the user should set the mask to binary 11111111100, that is 2044 in decimal.

    > **Warning:** Make sure that mask is set properly to be able to receive the desired CAN messages.
    >
    > The mask should be **11 bits for Standard** frame format and **29 bits for Extended** frame format.
    >
    > More information about this can be found in *How to calculate a mask - FAQ* section of this manual.

  - **Filter type**: The options available are *Standard*, *Extended* and *Both*.
- **CAN unwrapper**: This undoes the 'CAN wrapper' action, it has to be connected to *I/O Setup* **consumer**.
- **CAN GPIO remote**: CAN messages to GPIO peripherals such as CEX, MEX and Arbiter. It can be configured in the **configuration button** ( icon), a pop-up window will appear:

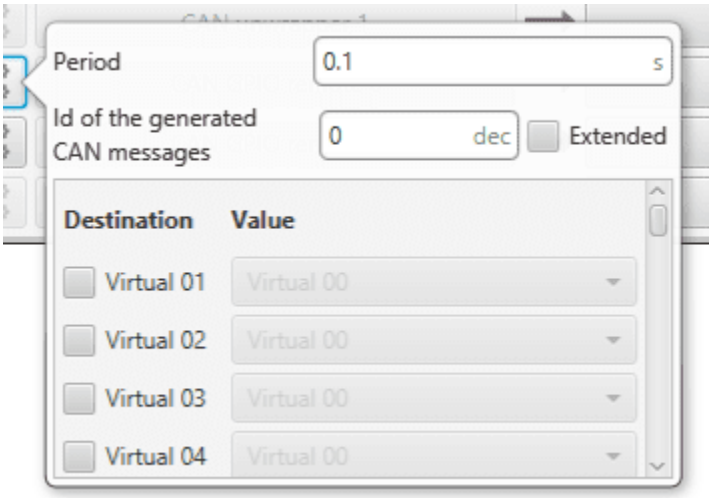


Fig. 100: **CAN GPIO remote configuration**

  - **Period**: This is the period of sending messages.
  - **Id of the generated CAN message**: CAN Id must be set and it is used to identify messages.
  - **Extended**: If enabled, the frame format will be this, 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
  - **Destination**: Here the user select the destination CEX (or other Veronte peripheral) pins.
  - **Value**: The user must select the 1x pin to be connected to the CEX (or other Veronte peripheral) pin.
- **CAN 4x**: CAN message transmission already configured for correct **communication between the Autopilots 1x within the Veronte Autopilot 4x**. If the user has any questions, please contact the support team following the Joint Collaboration Framework.

On the other hand, the **consumers** are the following:

- **CAN to serial**: This undoes the 'Serial to CAN' action, it has to be connected to *I/O Setup* **producer**.

- **Custom message**: CAN custom messages reception. They are configured in the Custom message panels, explained in *Custom Messages* section.
- **Output filter**: CAN output filters. The user can choose between *CAN A*, *CAN B* or *BOTH* in the **configuration button** ( icon).

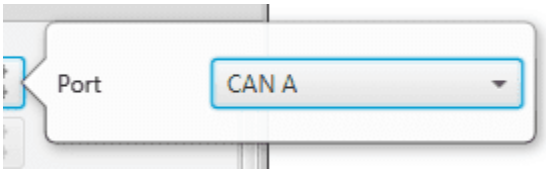


Fig. 101: **Output filter configuration**

- **CAN wrapper**: CAN messages over serial output, it has to be connected to *I/O Setup* **producer**.
- **CAN 4x**: CAN message reception already configured for correct **communication between the Autopilots 1x within the Veronte Autopilot 4x**. If the user has any questions, please contact the support team following the Joint Collaboration Framework.

2.4.2.2 Custom Messages

In the custom message tabs (there are 3 available), the user chooses the variables to be sent/received over the CAN buses. The following elements can be configured:

- **TX Ini**: Used to configure transmitted messages that are only sent once at the beginning of the operation (sent when the autopilot boots up). They can be used to initialize some devices.
- **TX**: Used to configure transmitted messages.
- **RX**: Used to configure the reception messages (where they are stored).

Warning:

- The **maximum capacity** of a **CAN message** is **64 bits** (8 bytes), so to send more information it must be divided into several messages.
- Autopilot 1x has a **CAN limitation** of **40 TX** messages per Custom, **40 TX Ini** messages per Custom and **80 RX** messages per Custom.

  In addition, there is a limit shared with all Customs, including **RS Custom Messages**:

  - Maximum number of **vectors** (fieldset): **104**
  - Maximum number of **fields**: **2000**

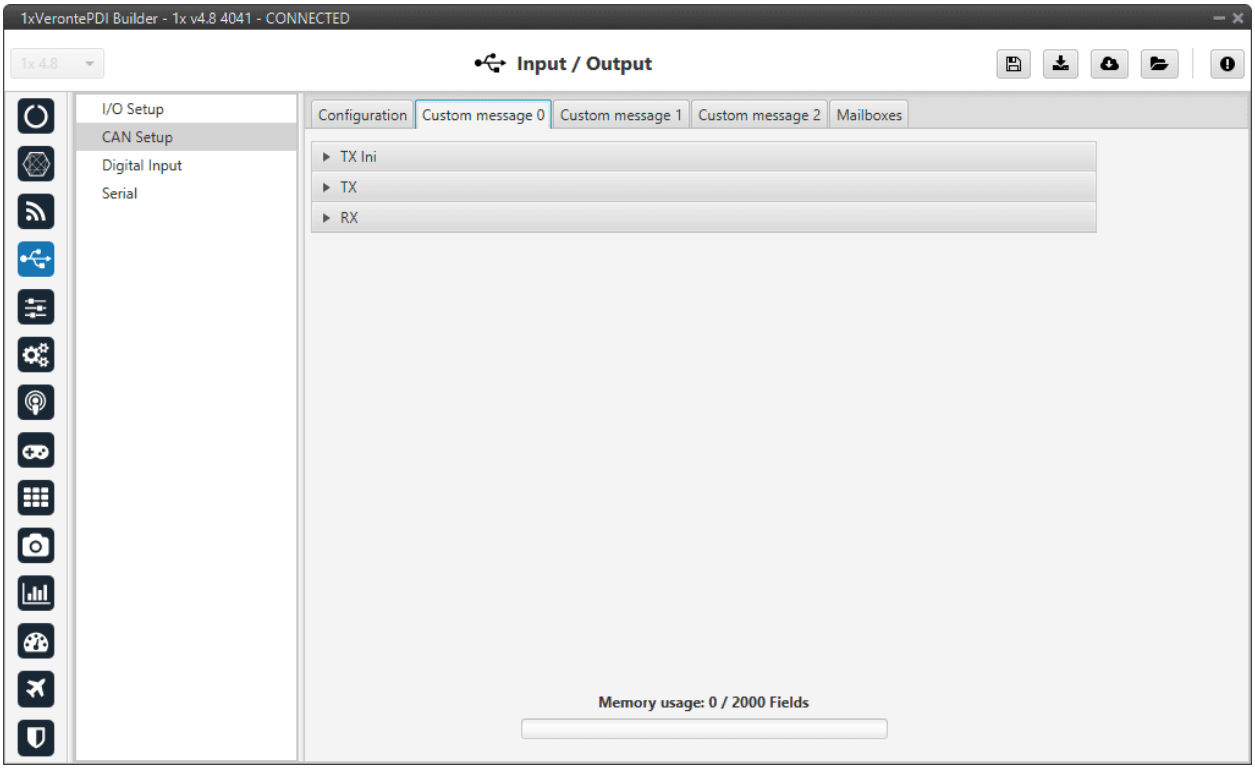


Fig. 102: **Custom Message panel**

TX/TX Ini Messages

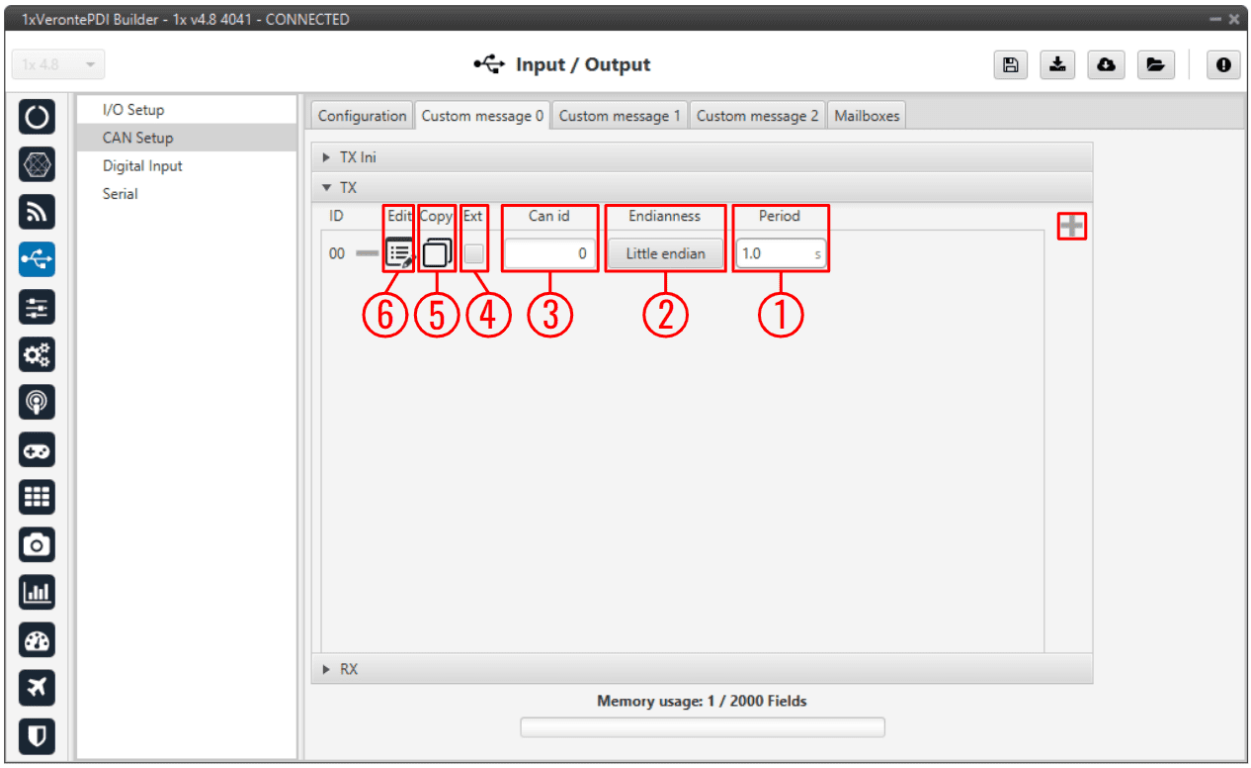


Fig. 103: **CAN Custom Message - TX**

In order to add a new custom message the user needs to press + and a new element will be added into the panel.

1. **Period**: This is the time in seconds between TX messages delivery.
2. **Endianness**: The endianness of the message must be configured, which indicates how the bytes that it contains are sent/read:
   - **Big endian**: Set the value from left to right.
   - **Little endian**: Set the value from right to left.
   - **Mixed endian**: Some devices use this format. If users need to configure it, please contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).
3. **Can id**: 11-bits (Standard) or 29-bits (Extended) ID used to identify TX messages. The value set has to be **decimal format**.
4. **Ext**: If enabled, the frame format will be 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
5. **Copy**: This message will be duplicated.
6. **Edit** : Displays the menu to configure how the bits/bytes of the message are divided and sent.

   There are six different options that can be added when setting up a custom message: **Variable**, **Checksum**, **Matcher**, **Skip**, **Parse ASCII** and **Position**.

   The explanation of how to configure these different types of custom messages is detailed in the following section ⇒ *Custom Messages types*.

RX Messages

The procedure is similar to the one followed in TX messages.

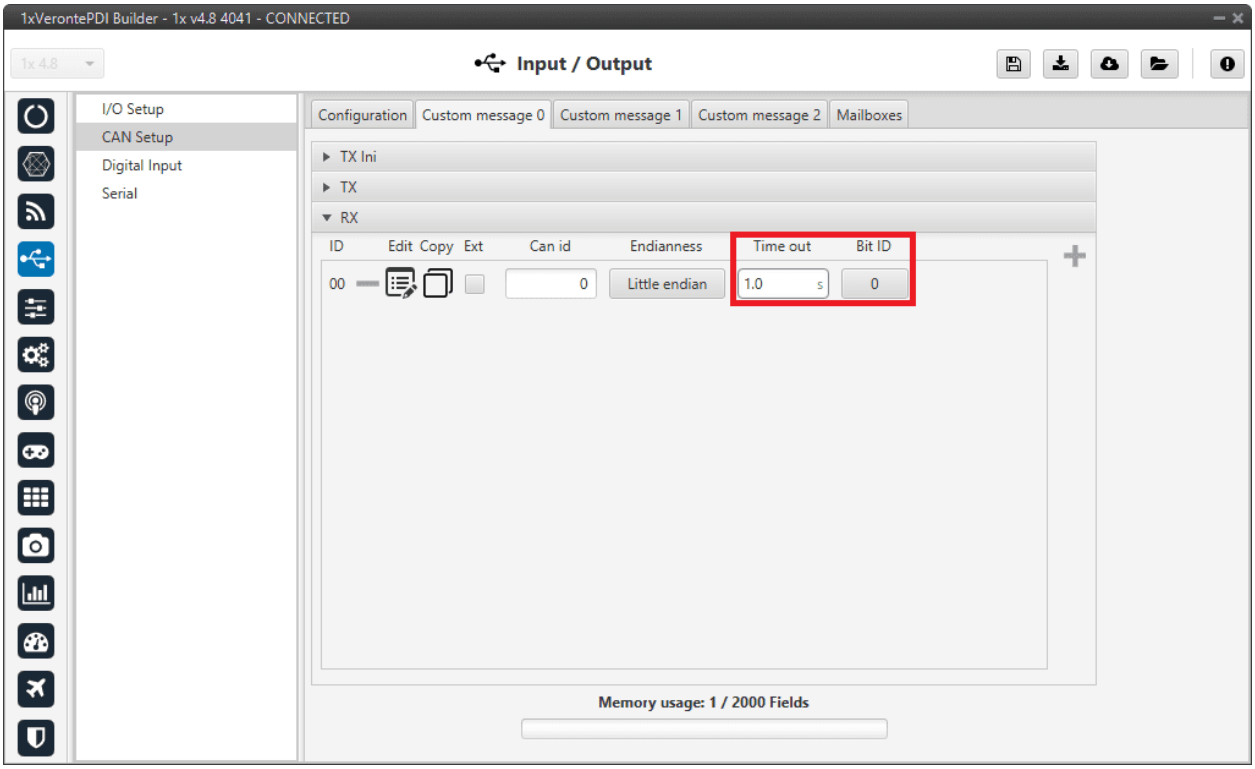


Fig. 104: **CAN Custom Message - RX**

The options and parameters to configure here are almost the same as those described in *TX/TX Ini Messages* above, except for one:

- **Can id**: The custom message needs to have the expected ID with which the external device/sensor is going to be sending information.

> **Attention:** It is important to configure a mailbox for every single reception ID. See *Mailboxes* section for more information.

Also, unlike TX messages there is **two additional variables per message**:

- **Time out**: for RX messages. This is the threshold time between receptions to consider that it is not being received correctly. For example, if time out is set to 1s and it passes more than 1s since the last reception, the Bit ID will be set to false.
- **Bit ID**: The user bit selected in Bit ID box will be true if the message is being received correctly.

> **Warning:** Pay attention that the user bit selected in **Bit ID** is not in use for another task.

The custom message structure needs to match the reception data-format. **User variables** (real - 32 bits , integer - 16 bits or boolean - 1 bit) may be used to store that data.

2.4.2.3 Mailboxes

Main screen to configure baudrate and reception mailboxes of each CAN bus (CAN A or CAN B) and enable an internal CAN resistor.

Since Veronte Autopilot 1x is going to receive data on the CAN Bus, it is mandatory to configure a certain number of mailboxes to store that data until Autopilot 1x reads it. A mailbox can be configured for multiple CAN message IDs as long as the mask is configured correctly and these messages are **sent spaced out** with enough time between them to allow the high priority core to read each one individually. More information on masks can be found in *How to calculate a mask - FAQ* section of this manual.

> **Warning:** Since **1x PDI Builder allows up to 32 mailboxes**, users should make sure to **leave at least one mailbox free for transmission (TX)**.
>
> If any mailbox is **full** and another message arrives, the **new** message is **discarded**.

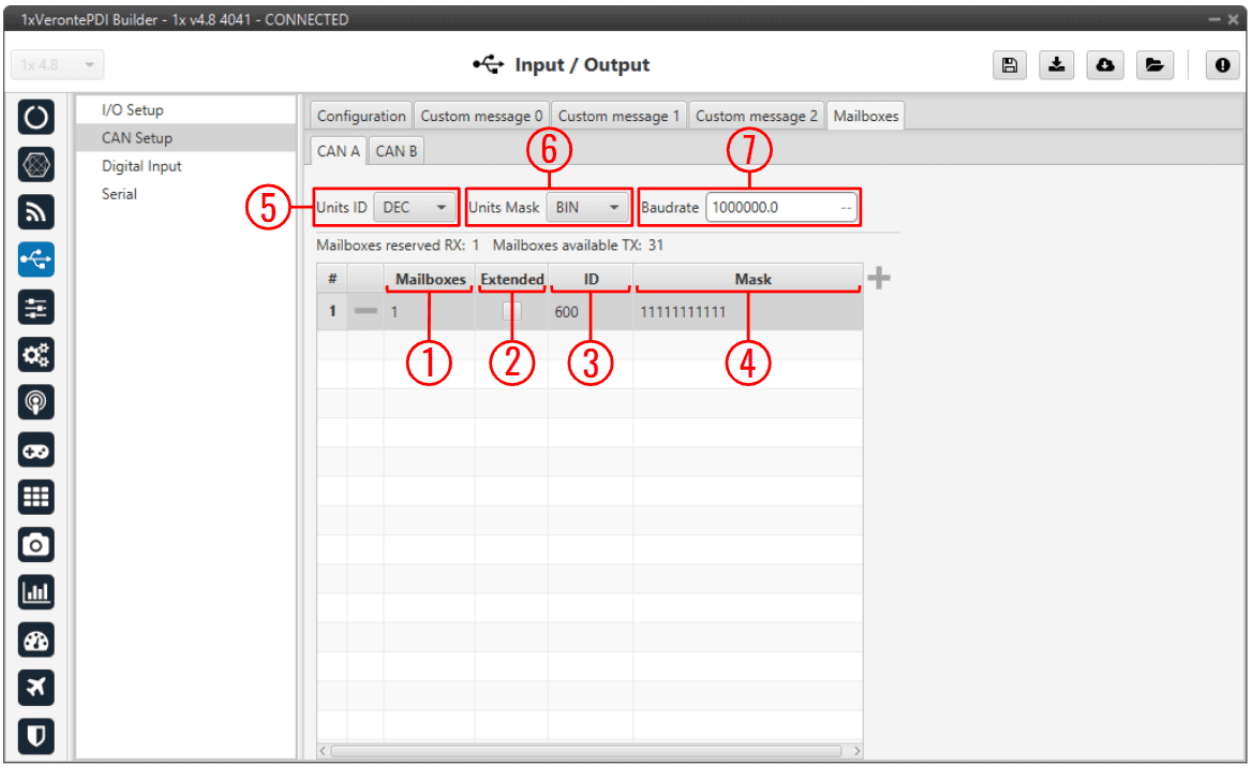


Fig. 105: **Mailboxes panel**

In order to add a mailbox, press the + icon.

The configurable parameters when adding a new mailbox are:

1. **Mailboxes**: Number of mailboxes assigned to that ID.
2. **Extended**: If enabled, the frame format will be this, 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
3. **ID**: 11-bits (Standard) or 29-bits (Extended) ID used to identify RX messages. The value set can be defined in different units, this is configured in **(5)**.

4. **Mask**: This filter is configured for reception messages; received data will be stored on mailboxes where message ID coincides with mailbox ID.

   Mask adds some flexibility on the reception, when comparing message with mailbox data, only the value of binary digits configured as 1 on the mask will be taken into account. The value set can be defined in different units, this is configured in **(6)**.

   For example, for a configuration **mask**: **11** 000 and **ID**: **10** 110, all incoming messages addressed to **10** XXX will be received in this mailbox.

   > **Warning:** Make sure that mask is set properly to be able to receive the desired CAN messages.
   >
   > The mask should be **11 bits for Standard** frame format and **29 bits for Extended** frame format.
   >
   > More information about this can be found in *How to calculate a mask - FAQ* section of this manual.

5. **Units ID**: Units available are **Decimal**, **Hexadecimal** or **Binary**.
6. **Units Mask**: Units available are **Decimal**, **Hexadecimal** or **Binary**.
7. **Baudrate**: CAN Baudrate can be configured here.

Example

Regarding the configuration of masks for mailboxes, it is possible to have only 1 mailbox for the reception of several messages in order to have more mailboxes available.

To do so, let's take as an example the configuration in the image above, with **ID**: **0010 0101 1000** (600 in DEC) and **Mask**: **1111 1111 1100**, all incoming messages addressed to **0010 0101 10XX** will be received in this mailbox.

This is because the mask that has been configured, looks for the **first 10 bits to match** the configured ID, and **ignores the last 2 bits**.

So, the following messages with CAN ID 600, 601, 602 and 603 could be stored in this mailbox (one at time), because:

- 600 DEC in Binary is ⇒ **0010 0101 1000**
- 601 DEC in Binary is ⇒ **0010 0101 1001**
- 602 DEC in Binary is ⇒ **0010 0101 1010**
- 603 DEC in Binary is ⇒ **0010 0101 1011**

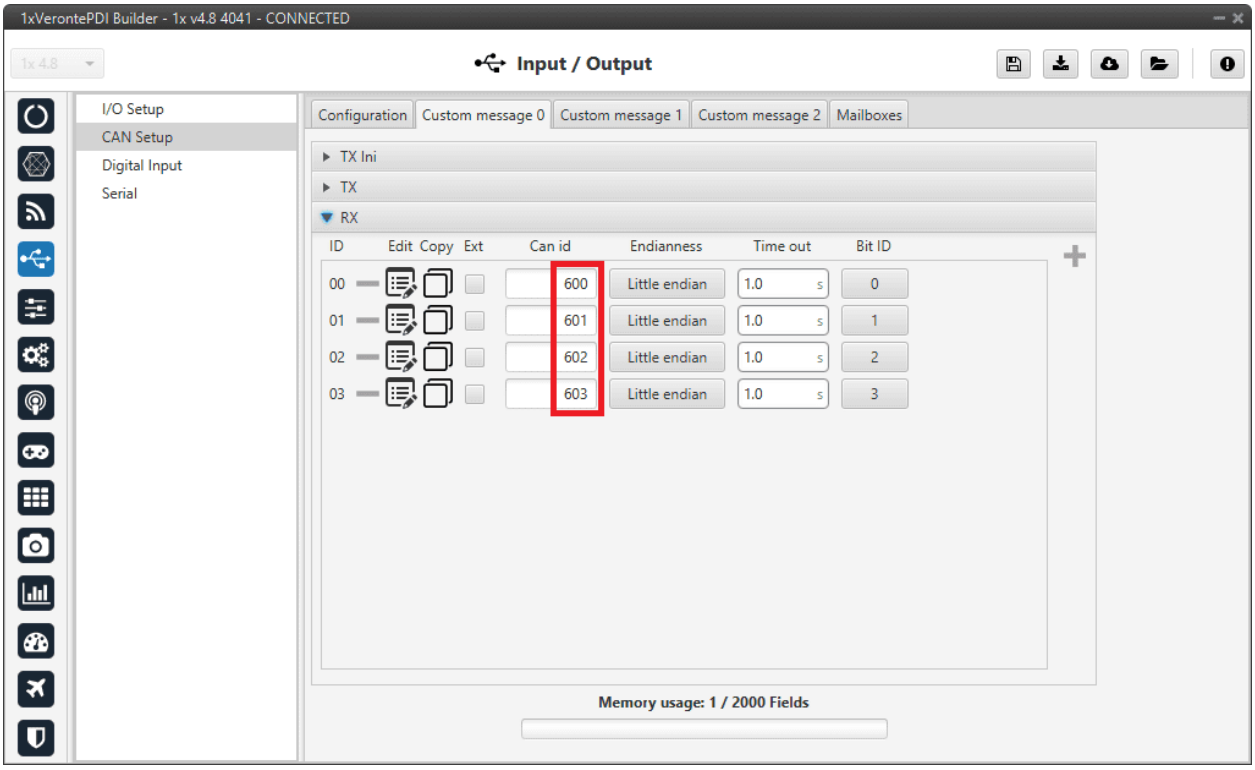


Fig. 106: **Mailboxes panel - Mask example**

2.4.3 Custom Messages types

There are six different options that can be added when setting up a custom message: **Variable**, **Checksum**, **Matcher**, **Skip**, **Parse ASCII** and **Position**.

2.4.3.1 Variable

Used to **store certain nº bits in a system variable** (**RX**) or to **send a certain variable** (**TX**).

> **Caution:** Users must take into account that Veronte system can handle float variables up to 32 bits (single precision float). Therefore, compression option must be considered to avoid data loss.

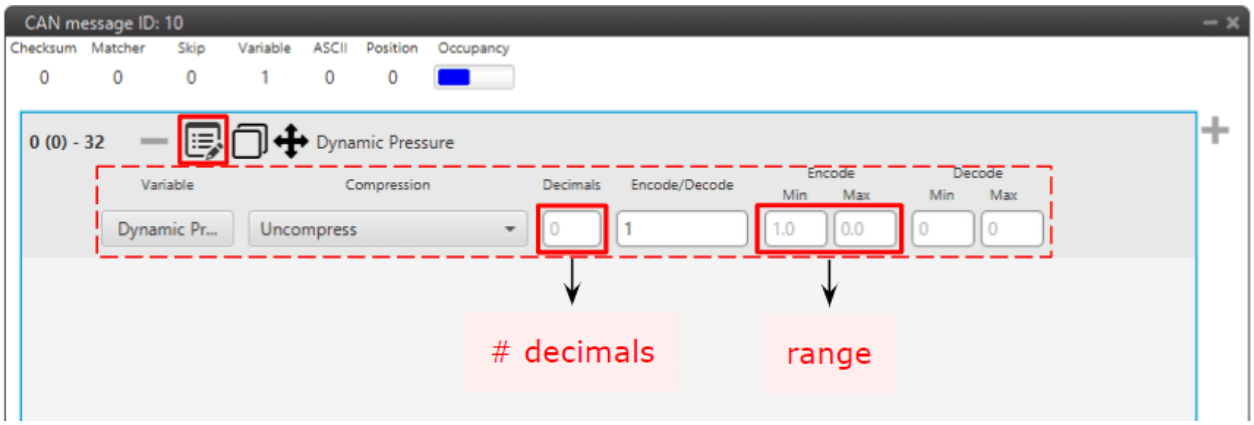


Fig. 107: **Variable configuration - CAN Custom Message**

The following parameters are configurable:

- **Variable**: Here the user select the desired system variable.
- **Compression**: The first step is to configure which kind of compression will be used for this variable:
  - **Uncompress**: The variable is taken in its full length, with no value modification.
  - **Uncompress - 64 bits**: Converts the selected variable type to a double precision float:
    - **Real variables (32 bits)**: In **TX**, uncompress from 32 to 64 bits. In **RX**, uncompress from 64 to 32 bits.
    - **Integer variables (16 bits)**: In **TX**, uncompress from uint 16 to float 64 bits. In **RX**, uncompress from float 64 to uint 16 bits.
    - **Bit variables (1 bit)**: In **TX**, uncompress from 1 to 64 bits. In **RX**, uncompress from 64 to 1 bit.

    Warning: Be careful! This transformation implies a loss of precision in both directions.

  - **Compress**: Compress to the specified number of bits. Users have to specify it in the field **Max** of the **Encode** parameter as follows: $2^n - 1$, where n is the desired number of bits to use to send/receive the variable.

    Note that this value (Max field) is the maximum value that can be obtained with the desired number of bits (n).

    Important: This is only available when **Integer variables (16 bits)** are selected.

  - **Compress - Decimals**: The variable is compressed according to the number of decimals specified and the range specified (max and min values). The resultant compression (number of bits) follows the relation $(max - min) \cdot 10^{decimals}$, which yields the encoding of the maximum value of the range (and the number of bits necessary for that). The range needs to be specified on the **Encode - Min/Max** field.
  - **Compress - Bits Signed**: Specify the number of bits to be compressed to (**negative values accepted**). It is necessary that the user configures **Encode/Decode** options.
  - **Compress - Bits Unsigned**: Specify the number of bits to be compressed to (**no negative values accepted**). It is necessary that the user configures **Encode/Decode** options.

Important: The compression options *Compress - Decimals, Compress - Bits Signed* and *Compress - Bits Unsigned* are **not available** when an **Integer variable (16 bits)** is selected.

  - **Uncompress - 16 bits**: Converts the selected variable type to a half precision float:
    - **Real variables (32 bits)**: In **TX**, uncompress from 32 to 16 bits. In **RX**, uncompress from 16 to 32 bits.
    - **Integer variables (16 bits)**: In **TX**, uncompress from uint 16 to float 16. In **RX**, uncompress from float 16 to uint 16.
    - **Bit variables (1 bit)**: In **TX**, uncompress from 1 to 16 bits. In **RX**, uncompress from 16 to 1 bit.

- **Encode/Decode**: These values are used to apply a scaling factor after the transformation from binary to decimal value, or before the transformation from decimal to binary value.

Note: If no compression is desired, the same values must be set in min/max Encode and min/max Decode. For example, Encode min=0 / max=1 and Decode min=0 / max=1.

Example

In the example shown below, a real user variable (32 bits) is being used to receive data from an external device. This data corresponds to the heading angle of the aircraft (which goes **from 0 to 359 degrees**). The device is sending this information in a **16-bit** data frame and the angle value times 100 (hence why the **Decode** parameter goes **from 0 to 35900**). This needs to be saved in **1x PDI Builder** in the user variable in the range 0 to 359 (**Encode**).

Fig. 108: **Variable configuration example**

2.4.3.2 Checksum (CRC)

Sometimes, **control codes** are needed for **preventing random errors in transmission**, where a bits frame is operated and the result is sent to the receiver to check it. To do so the CheckSum option is used.

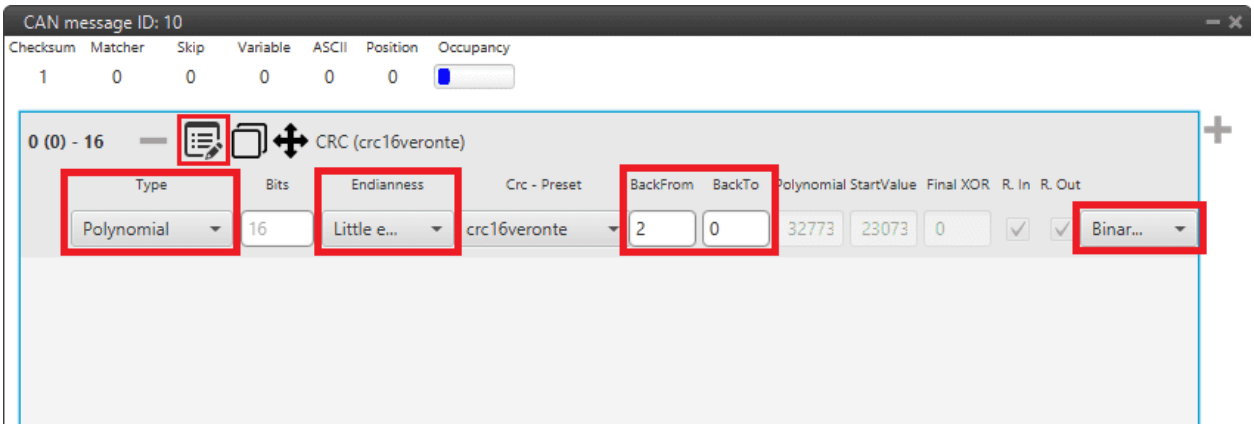


Fig. 109: **Checksum configuration - CAN Custom Message**

- **Type**: User can choose the type of CRC that will be applied.
  - *Polynomial*: Polynomial algorithm for CRC. Select from a list of predefined Embention CRC (**CRC-Preset** option). This is the most used type.
  - *sum8*: The CRC-8 algorithm (sum8) calculates an 8-bit checksum, which is used for error detection purposes.

    Basically, it processes the sum of all bytes from a sequency, and then performs a bitwise division by 255 to retrieve the CRC result code.
  - *sumMod*: The CRC module (sumMod) is a process to calculate a checksum, which is used for error detection purposes.

    It processes the sum of all bytes from a sequency, and uses a euclidian division by the module parameter to obtain the CRC result code.
  - *Mavlink*: Embention has implemented the Mavlink checksum, used only for Mavlink protocol communications.
  - *8-bit sagetech checksum*: It is an owned checksum algorithm from Sagetech, that is used by Sagetech devices for error detection purposes. It is based on Fletcher checksum.

The following parameters must be set **independently of the type** of checksum selected:

- **Endianness**: The endianness of the message must be configured, which indicates how the bytes that it contains are sent/read:
  - **Big endian**: Set the value from left to right.
  - **Little endian**: Set the value from right to left.
  - **Mixed endian**: Some devices use this format. If users need to configure it, please contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).
- **Back From**: Indicates that the CRC will be computed from the indicated byte (inclusive).
- **Back To**: Indicates that the CRC will be computed to the indicated byte (exclusive).

Explanation

- Byte 0 it is referred to the first byte of the Checksum block.

– The range of calculation of the CRC is defined by the 'Back to' and 'Back from' parameters. They define, respectively, a number of bytes as an offset from the position of the CRC.

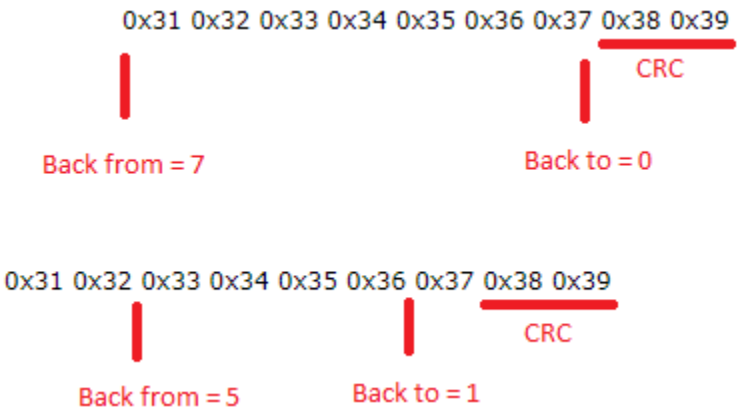


Fig. 110: **Back to/Back from explanation**

- Drop-down menu: User can choose the mode in which the CRC will be output:
  - **Binary mode**
  - **ASCII as hexadeciaml values**
  - **ASCII as decimal values**

The specific parameters for each checksum type, will be described below:

2.4.3.2.1 Polynomial type

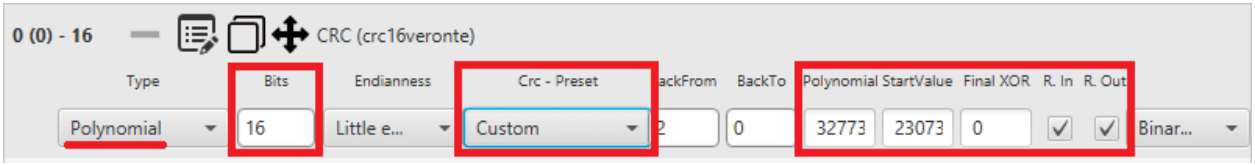


Fig. 111: **Checksum configuration - Polynomial example**

In addition to the 'general' parameters described above, one further parameter must be configured for this type of CRC:

- **CRC - Preset**: List of predefined Embention CRC, where fields nº Bits, Polynomial, Start Value, Final XOR, Reflect In and Out are defined.

  The last option is Custom, where all the above mentioned fields can be defined by the user. Check Polynomial CRC online for more information.

  – **Bits**: This defines the width of the result CRC value (nº bits).

  – **Polynomial**: Used generator polynomial value.

  – **Start Value**: The value used to initialize the CRC value / register.

  – **Final XOR**: The Final XOR value is xored to the final CRC value before being returned. This is done after the 'Result Output' step. Obviously a Final XOR value of 0 has no impact.

- **Reflected Input**: If this is **enabled**, each input byte is **reflected** before being used in the calculation. Reflected means that the bits of the input byte are used in reverse order. So this also means that bit 0 is treated as the most significant bit and bit 7 as least significant.
- **Reflected Output**: If this is **enabled**, the final CRC value is reflected before being returned. The reflection is done over the whole CRC value, so e.g. a CRC-32 value is reflected over all 32 bits.

2.4.3.2.2 sum8 type

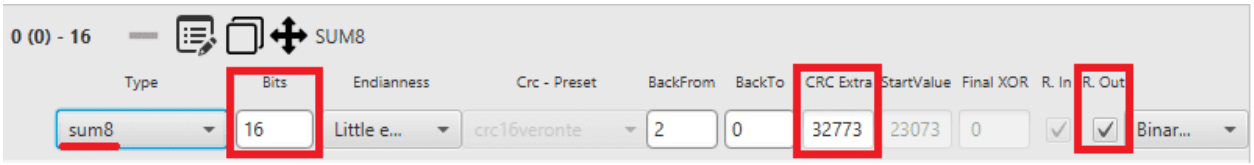


Fig. 112: **Checksum configuration - sum8 example**

In addition to the 'general' parameters described above, 3 further parameters must be configured for this type of CRC:

- **Bits**: This defines the width of the result CRC value (nº bits).
- **CRC Extra**: Extra CRC added at the end of the message, it will be required by the communication protocol used.
- **Reflected Output**: If this is **enabled**, the final CRC value is reflected before being returned. The reflection is done over the whole CRC value, so e.g. a CRC-32 value is reflected over all 32 bits.

2.4.3.2.3 sumMod type

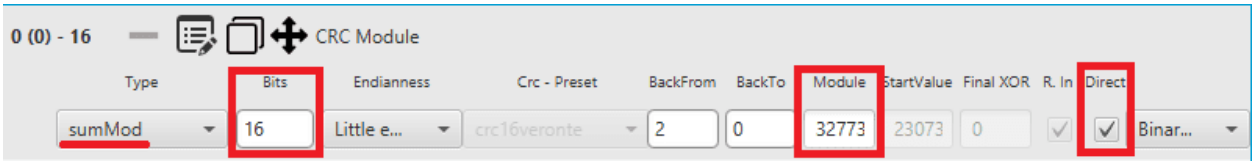


Fig. 113: **Checksum configuration - sumMod example**

In addition to the 'general' parameters described above, 3 further parameters must be configured for this type of CRC:

- **Bits**: This defines the width of the result CRC value (nº bits).
- **Module**: This value is the dividend of the operation carried out.
- **Direct**:
  - If **enabled**, sumMod retrieves the result code directly from the remainder of the division.

    The mathical operation done is: $CRC \,\%\, module$.
  - If **disabled**, sumMod keeps the subtraction of the remainder of the division.

    The mathical operation done is: $module - (CRC \,\%\, module)$.

2.4.3.2.4 Mavlink type

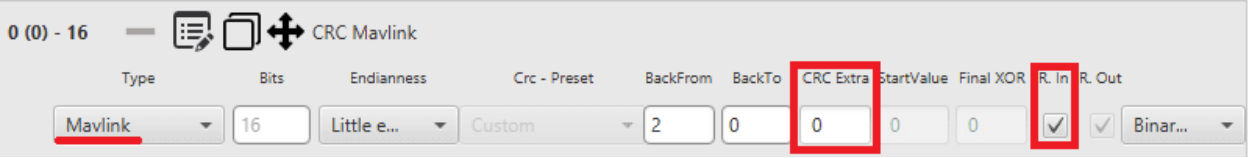


Fig. 114: **Checksum configuration - Mavlink example**

In addition to the 'general' parameters described above, 2 further parameters must be configured for this type of CRC:

- **CRC Extra**: Extra CRC added at the end of the message, it will be required by the Mavlink protocol.
- **Reflected Input**: If this is **enabled**, each input byte is **reflected** before being used in the calculation. Reflected means that the bits of the input byte are used in reverse order. So this also means that bit 0 is treated as the most significant bit and bit 7 as least significant.

2.4.3.2.5 8-bit sagetech checksum

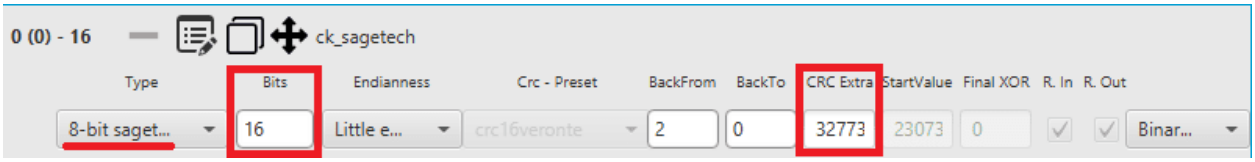


Fig. 115: **Checksum configuration - 8-bit sagetech checksum example**

In addition to the 'general' parameters described above, 2 further parameters must be configured for this type of CRC:

- **Bits**: This defines the width of the result CRC value (nº bits).
- **CRC Extra**: Extra CRC added at the end of the message, it will be required by the communication protocol used.

2.4.3.3 Matcher

This option is used to send a **constant value through the bus** in TX or wait for a particular value in RX.

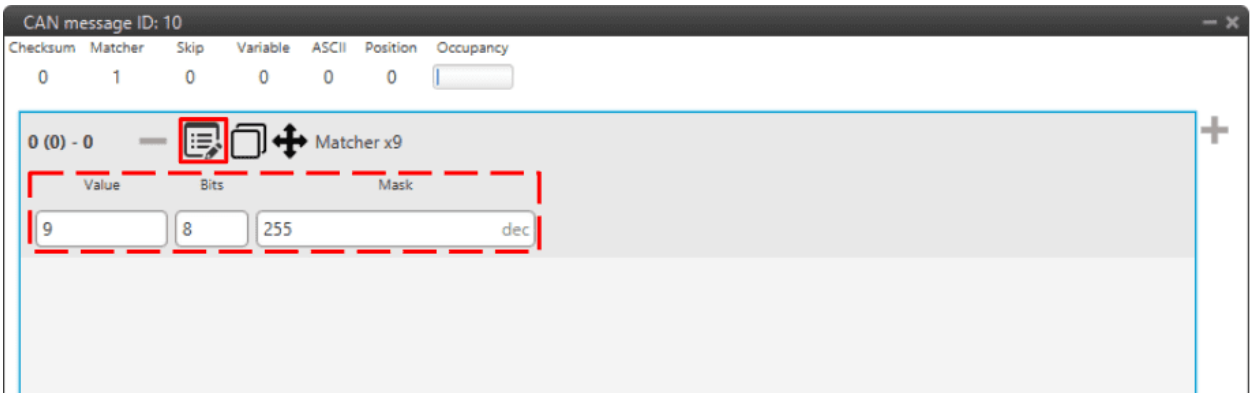


Fig. 116: **Matcher configuration - CAN Custom Message**

- **Value**: Sent/received value for the nº of bits defined below.
- **Bits**: Number of bits in which the matcher is performed.
- **Mask**: It is automatically set when the nº of bits is assigned.

For example, a matcher of 8 bits with a value of 9 will be reading/sending: 00001001.

2.4.3.4 Skip

This option is used to **discard a certain number of bits** from the message (the **maximum** number of bits that can be skipped with a single "Skip" are **32**).

This tool can be used when there are variables incoming that are from no interest for the user, not loading unnecessary information into the system.

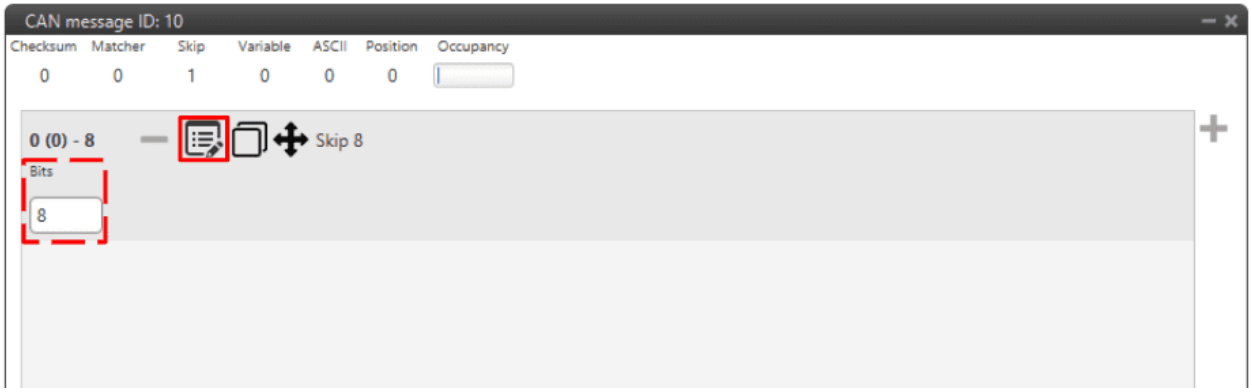


Fig. 117: **Skip configuration - CAN Custom Message**

2.4.3.5 Parse ASCII

Parsing ASCII is used when the ASCII protocol is required.

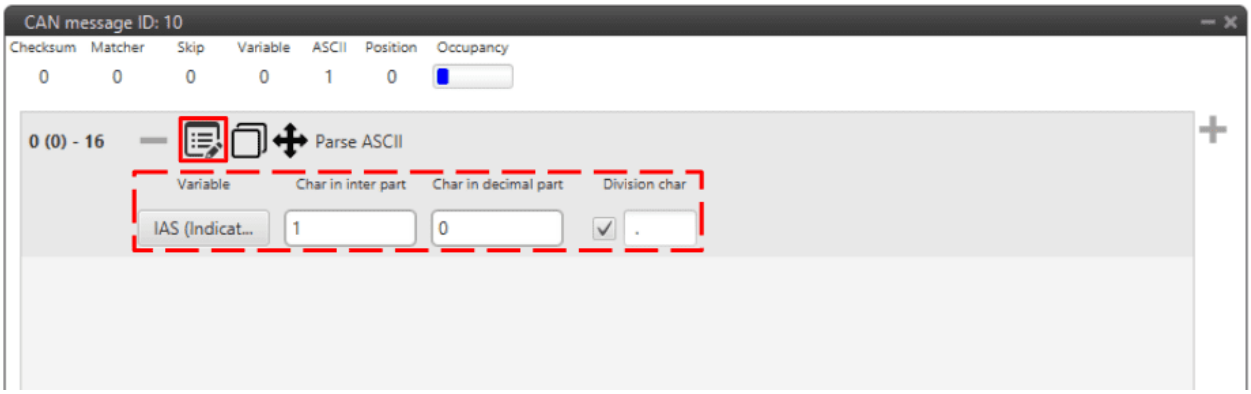


Fig. 118: **Parse ASCII configuration - CAN Custom Message**

ASCII protocol is used for **transforming a character array into decimal values**. For such task, the user needs to define the following parameters:

- **Variable**:
  - If used as **TX**, this variable is where the ASCII will be saved ("uncompress").
  - If used as **RX**, this is the read variable to be transformed into ASCII ("compress").
- **Char in inter part**: The number of characters in the integer part.
- **Char in decimal part**: The number of characters in the decimal part.
- **Division char**: This is the division character ('**.**', '**,**', etc.)

2.4.3.6 Position

Position is used to **input/output a data set with a particular format**. When created, the user can only choose variables from Features variables list.

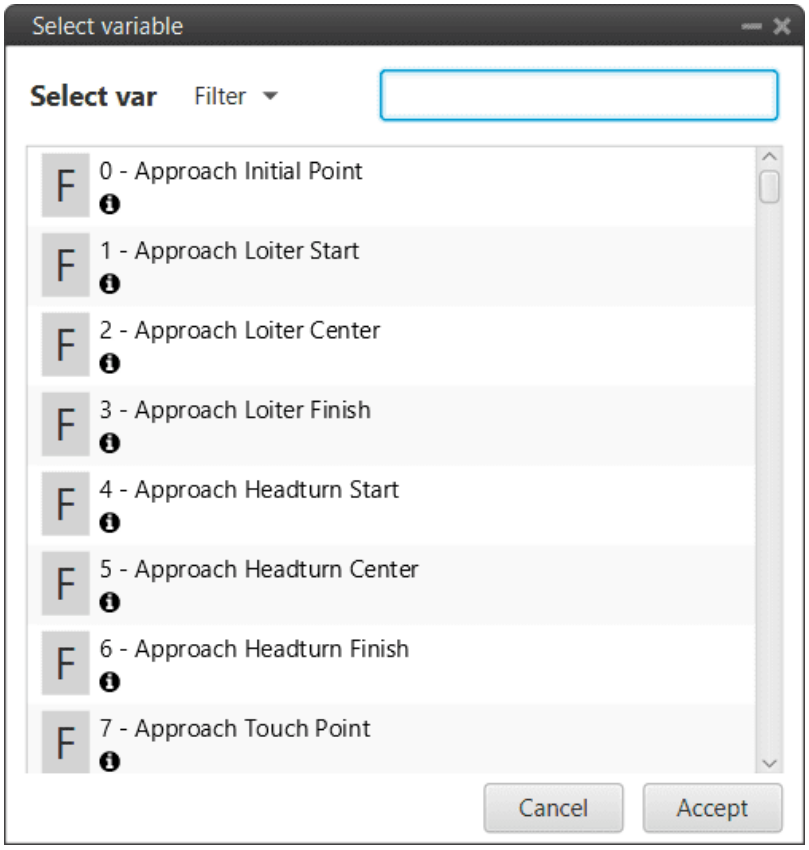


Fig. 119: **Position variables**

The window display below is the configurable menu. The information stored is the **WGS84 coordinates** in the following order: Latitude, Longitude and Height. All of them are stored with **double precision**.

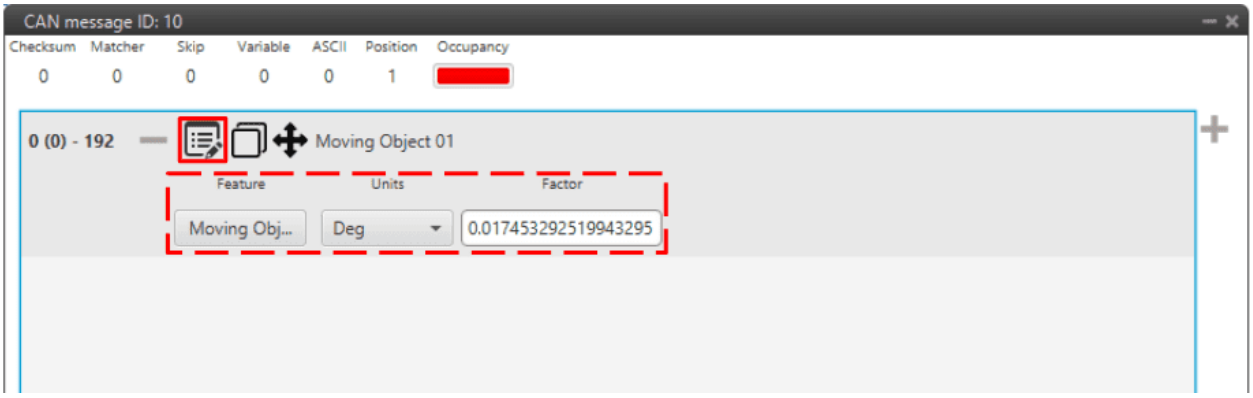


Fig. 120: **Position configuration - CAN Custom Message**

- **Feature**: User can select from Features variables.
- **Units**: Units available are **Radians**, **Degrees**, **Gradians** and Custom.
- **Factor**: As radians are the unit that **1x PDI Builder** works with, if another unit is selected, the conversion factor between this unit and radians is automatically calculated.

2.4.4 Digital Input

Digital inputs can be used to measure pulse count, pulse widths and PPM signals from a RC radio. Each source shall be connected to the desired consumer to allow measurements.

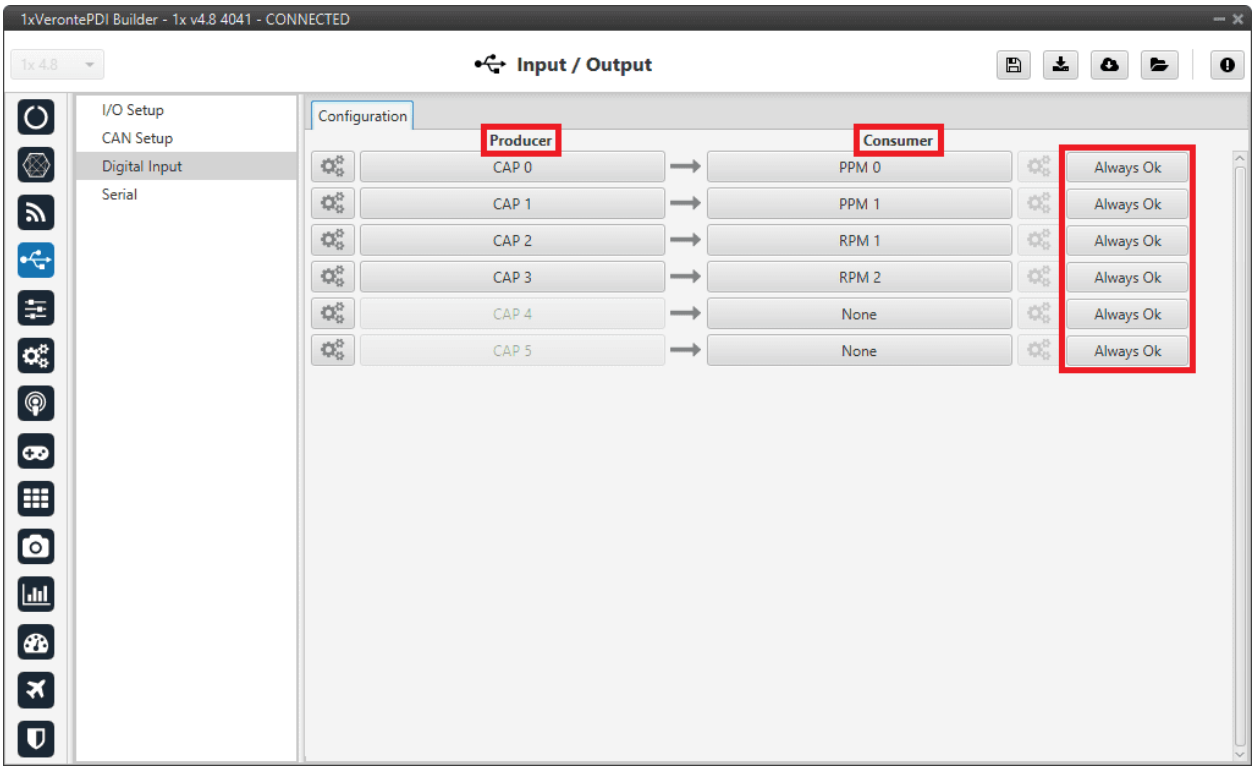


Fig. 121: **Digital Input panel**

In addition, in this menu the user can also find the same 'columns' (**Producer**, **Consumer** and **Bit**) as in the *I/O Setup panel*. In addition, the process for configuring producers and consumers is also the same as described in the *I/O Setup - Input/Output* section.

The process to configure a device can be done as follows:

1. Select and configure a **Producer**. There are 6 possible producers: CAP 0 - 5.

   Press on the configuration button ( icon) and a new pop-up window will show.

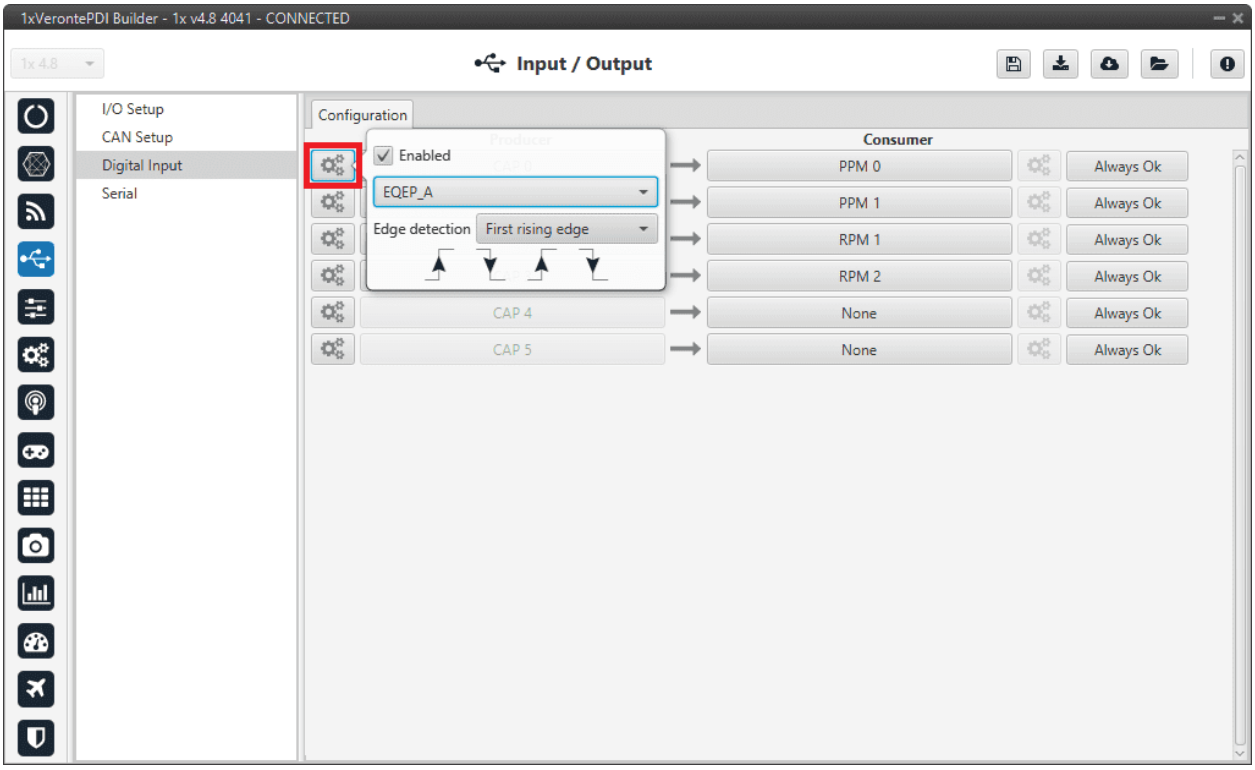


Fig. 122: **Digital Input panel - Producer**

The pop-up window contains the following configurable elements:

- **Enable**: By ticking this checkbox, the corresponding producer is enabled.
- **CAP pin entry**: Selects which **pin** this CAP is associated and, therefore, to which device is connected. It is possible to select these pins. Pins available are GPIO 0 to 15, and EQEP A, B, S and I.

Note: When using the harness provided by Embention the transmitter Digital Input is connected to the pin 55 (EQEP_A) with pin 59 as Ground.

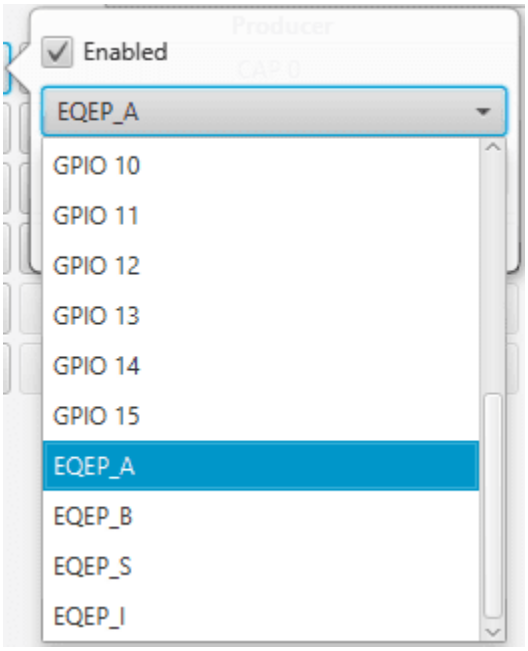


Fig. 123: **Digital Input panel - CAP**

- **Edge detection**: How the pulses are read and transformed into a digital signal (how they are processed).

  By clicking on the drop-down menu, the following options can be selected:

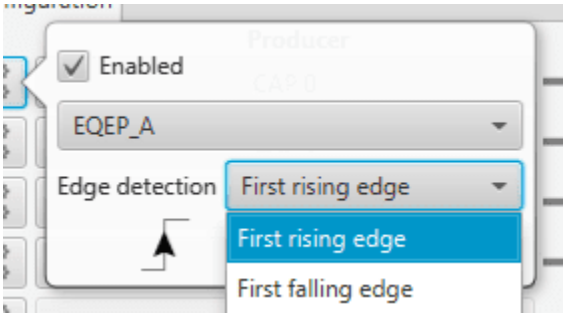


Fig. 124: **Digital Input panel - Edge detection option**

  - **First rising edge**: With this option, when the **rise** of the pulse is detected, the data will start to be stored. Recommended when consumer is **PPM** or **Pulse**.
  - **First falling edge**: With this option, when the **fall** of the pulse is detected, the data will start to be stored.

Note: By clicking on the arrows, it can also be configured as desired. For example, if the user has selected the 'First rising edge' option, but clicking on the arrows gets the arrow scheme of the 'First falling edge' option, the name of the edge detection will not be 'First rising edge', but will become 'First falling edge'.

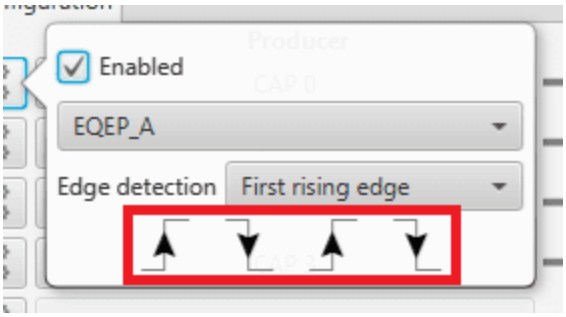


Fig. 125: **Digital Input panel - Edge detection arrows**

2. Click on the **Bind** button to select the type of **Consumer**, it is possible to choose among a PPM 0-3 (Stick PPM), RPM 0-5 (RPM sensor) or Pulse 0-3 (Pulse).

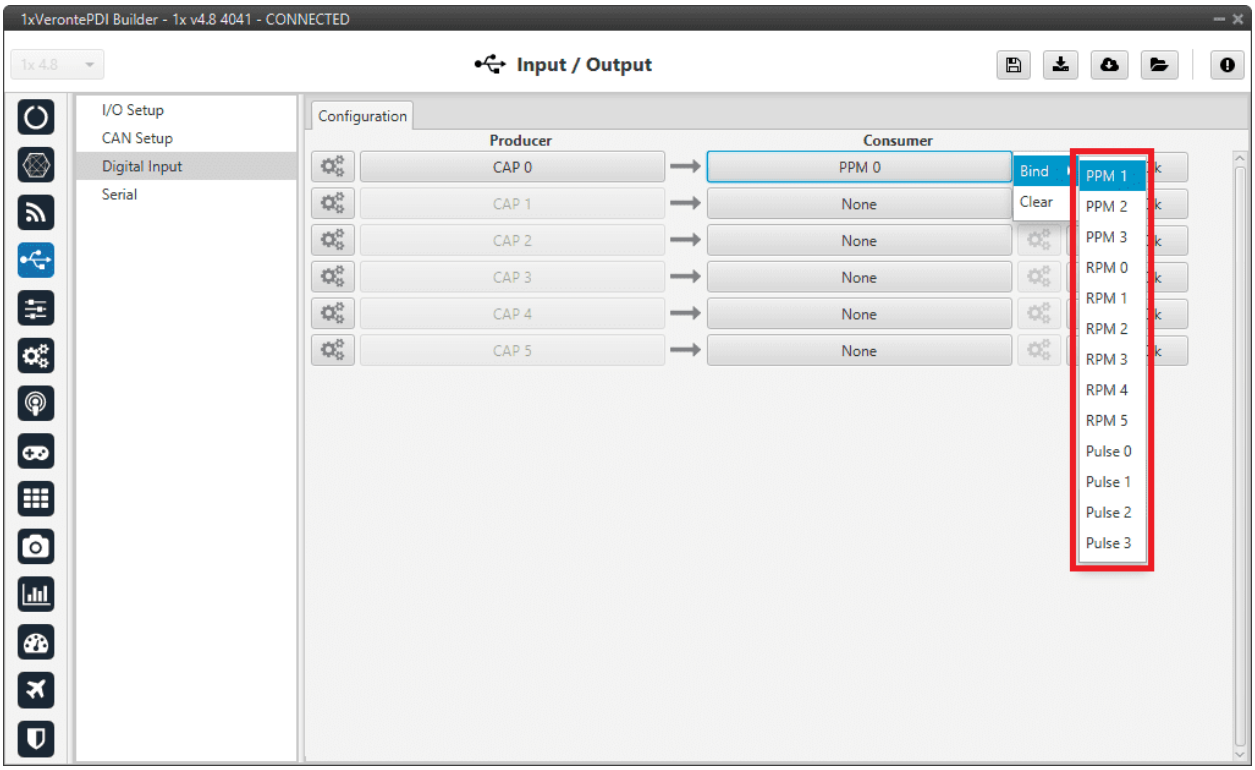


Fig. 126: **Digital Input panel - Consumer**

- **PPM 0-3** selected: PPM is configured in the *Stick panel*.
- **RPM 0-5** selected: '**RPM 0-5**' variables store the information read here. For more information on the configuration of RPM, see the *RPM* section.
- **Pulse 0-3** selected: '**Captured pulse 0-3**' variables store the information read here. It is possible to configure it clicking on the **configuration button** ( icon):

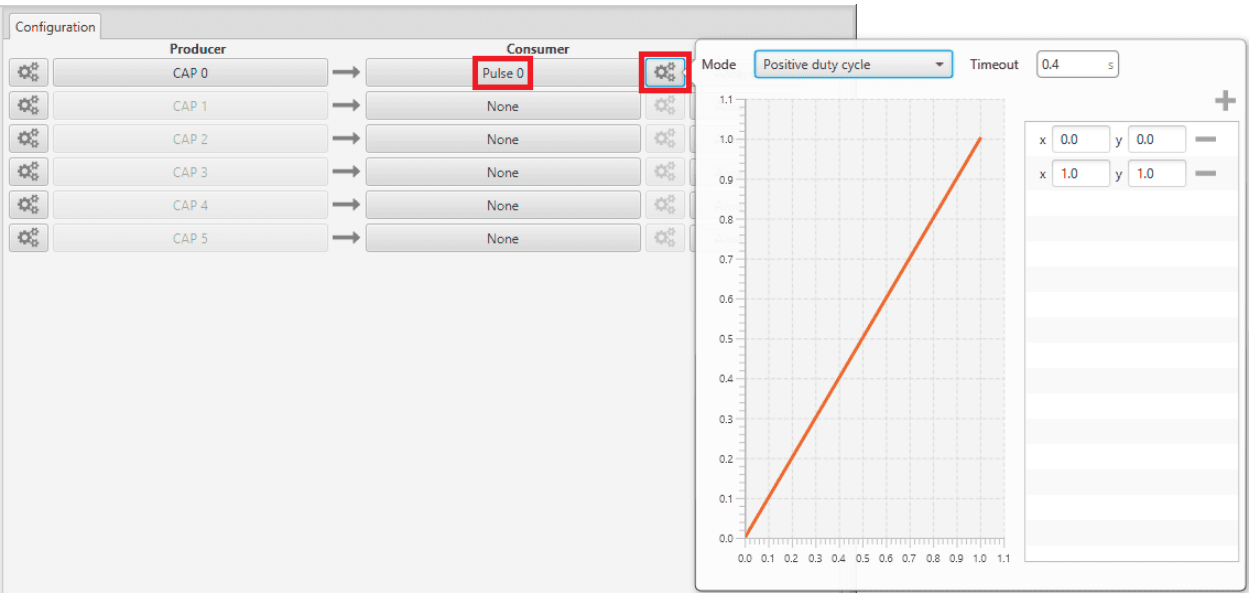


Fig. 127: **Digital Input panel - Pulse**

In the pop-up window, users will find the following options for configuration:

- **Mode**:
  - **Positive pulse duration**: The period of the pulse is obtained. It takes the time in 'High' state.
  - **Negative pulse duration**: The period of the pulse is obtained. It takes the time in 'Low' state.

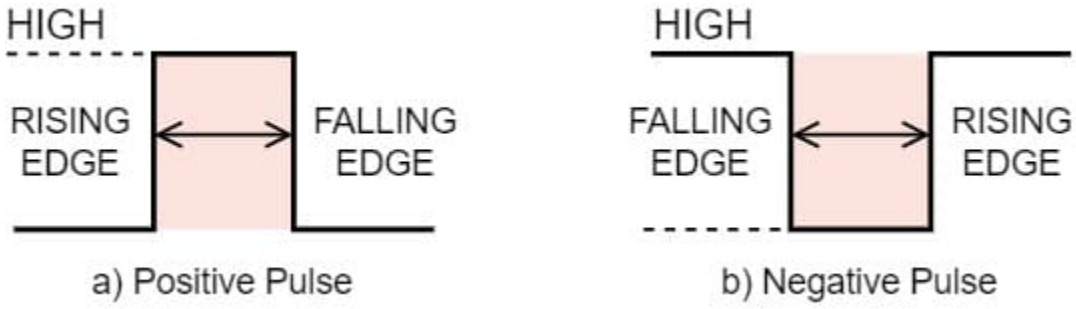


Fig. 128: **Positive/Negative pulse duration**

  - **Positive duty cycle**: The duty cycle. It takes the time in 'High' state.
  - **Negative duty cycle**: The duty cycle. It takes the time in 'Low' state.

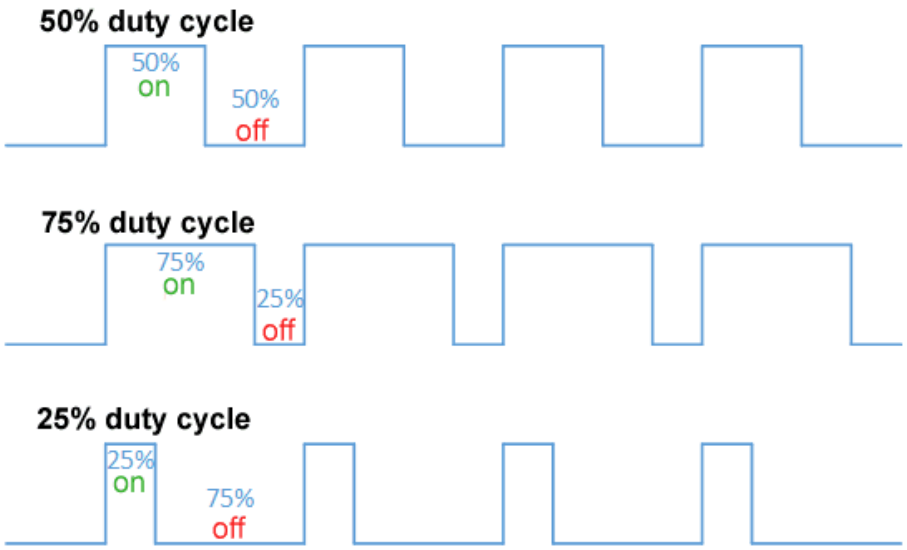


Fig. 129: **Positive/Negative duty cycle**

- **Time out**: This defines the time to consider that no signal is received.
- **Function**: Here the user can customize a function to handle the values. Normally, a function is set with the points [0,0] and [1,1], so no transformation is applied, input = output. However, the user can configure it as desired.

Example

Let's imagine that **First rising edge** has been selected as the edge detection option in Producer and the pulse that 1x has to read is a square signal with a period of 2 seconds and a duty cycle of 25% (see image below).

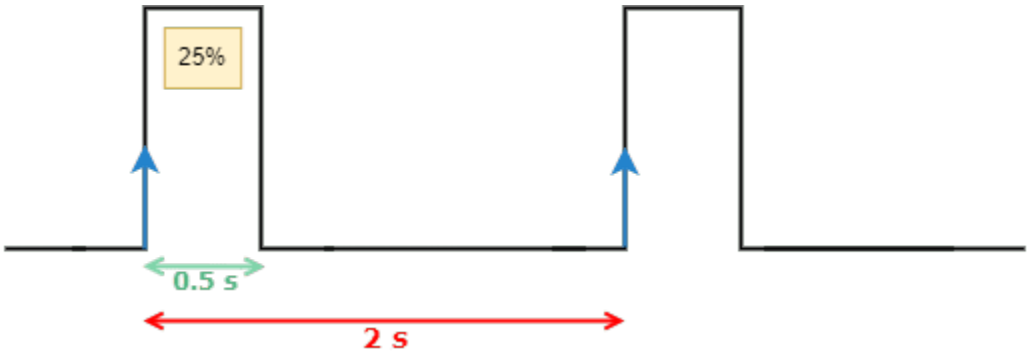


Fig. 130: **Signal generated**

On the other hand, if **Positive pulse duration** is selected as Consumer and it is configured as in the previous image (*Digital Input - Pulse*), the value obtained in the variable **Captured pulse** (*Captured pulse 1* in the following example) will be **0.50s**, this is because it is the period of the "Positive pulse" of that pulse.

However, if **Positive duty cycle** is selected as Consumer, the value obtained in the variable **Captured pulse** (*Captured pulse 2* in the following example) will be **0.25**, this is because it is the positive duty cycle of that pulse.

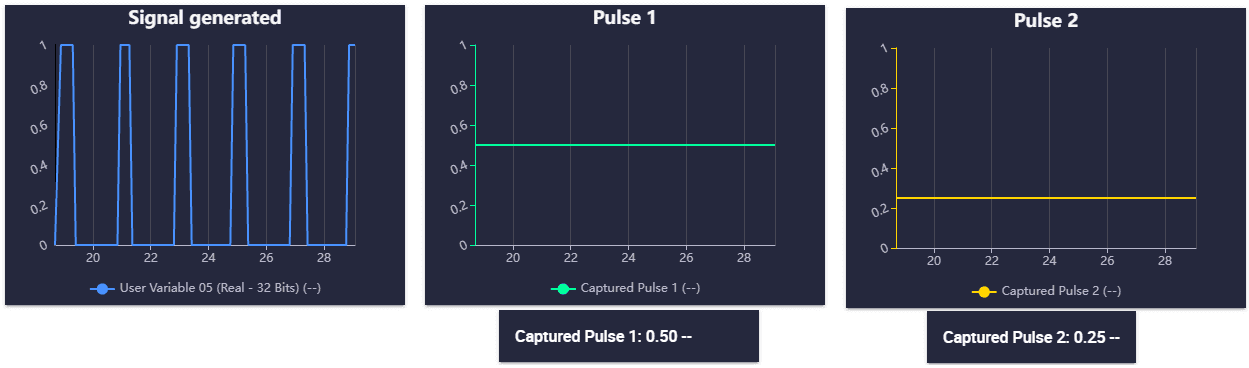


Fig. 131: **Digital Input example**

2.4.5 Serial

In this panel, users must configure the parameters of the different serial ports, *Veronte LOS*, *RS485* and *RS232*, in their corresponding tab.

However, as the parameters to be configured are shared across all serial ports, they are explained here:

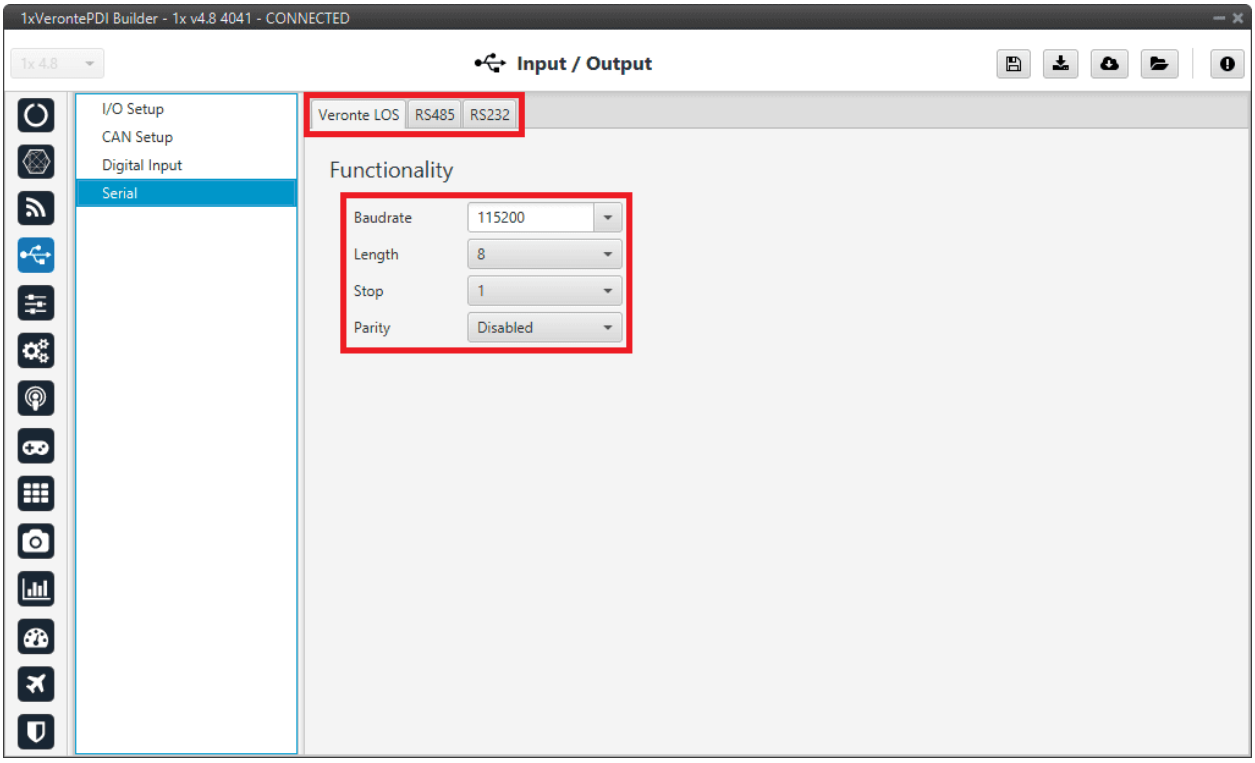


Fig. 132: **Serial menu - Veronte LOS panel**

- **Baudrate**: This specifies how fast data is sent over a serial line.

> **Warning:** **Veronte Autopilot 1x** has a buffer limit of 16 bytes, which is read every 1 ms. If the message is larger than 16 bytes, then the maximum viable baudrate will be 128,000 bps.

If the configured baudrate is higher than 128,000 bps and the message is larger than 16 bytes, then **Autopilot 1x** will lose data. Nonetheless, if the message is smaller or equal to 16 bytes, then the baudrate will not have this limitation.

- **Length**: This defines the number of data bits in each character: 4 to 8 bits.
- **Stop**: Number of stop bits sent at the end of every character: 1, 1.5, 2.
- **Parity**: Is a method of detecting errors in transmission. When parity is used with a serial port, an extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit. **Disabled**, **odd** or **even**.

Note: All these settings are already specified for a given device, therefore, Autopilot 1x should match with them in order to be able to communicate.

2.4.5.1 Veronte LOS

In this panel, the serial port that communicates from the microcontroller to the internal radio is configured.

Warning: If the user changes the baudrate on the internal radio, it is also required to change it here and vice versa.

2.4.5.2 RS485/RS232

Two serial interfaces are available with Autopilot 1x, 1 port RS-232 and 1 port RS-485, however more can be added by using a CEX or MEX. Each one of the serial interfaces is associated with a set of pins.

Compatibility table:

Port name	RS-232	RS-485	
		RS-422	RS-485
Transfer type	Full duplex	Full duplex	Half Full duplex
Maximum distance	15 meters at 9600 bps	1200 meters at 9600 bps	
Topology	Point to point	Point to point	Multi point
Max number of devices	1	1-10 in receive mode	32

2.5 Control

In this menu all the parameters related to the control of the platform can be found. There are 3 panels, each one showing a different menu of configuration.

2.5.1 Phases

In this panel, the flight phases that will control the aircraft at different stages of the operation are created (defined not configured).

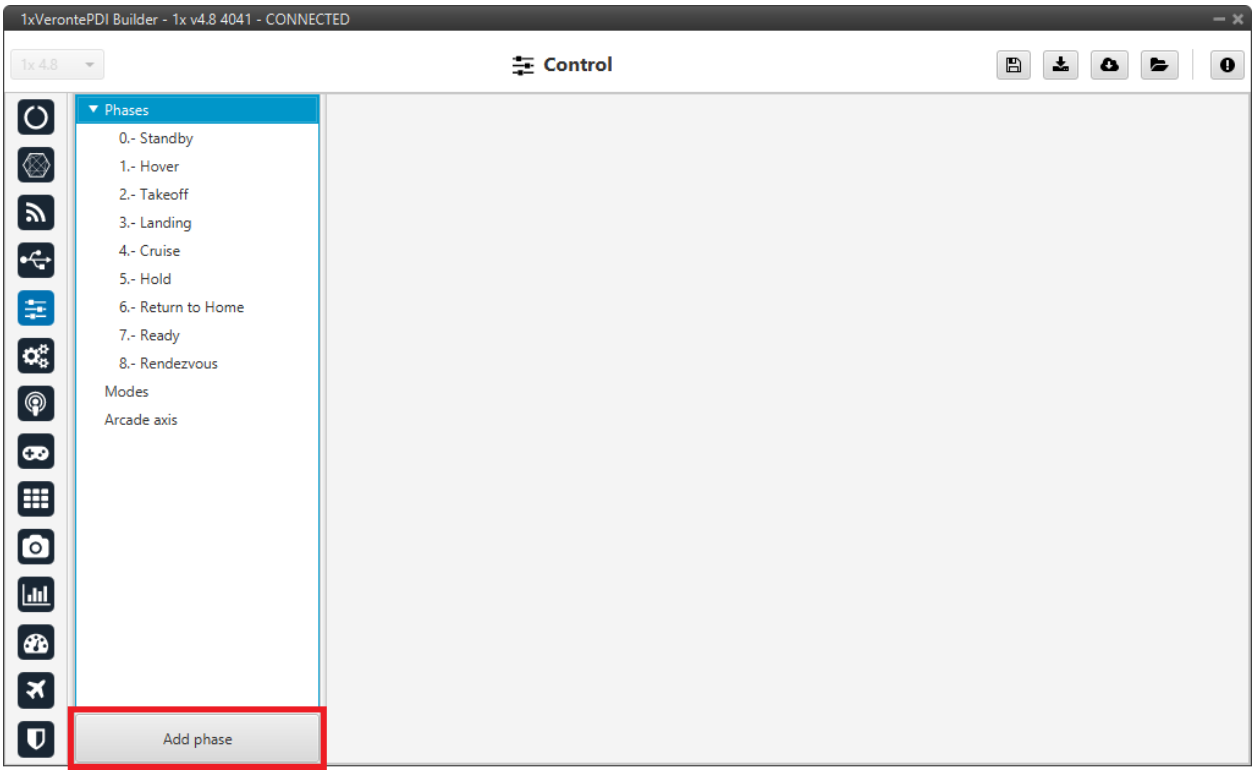


Fig. 133: **Phases panel**

To create a new phase click on *Add phase*, the user can then select a phase already created or create a new one.

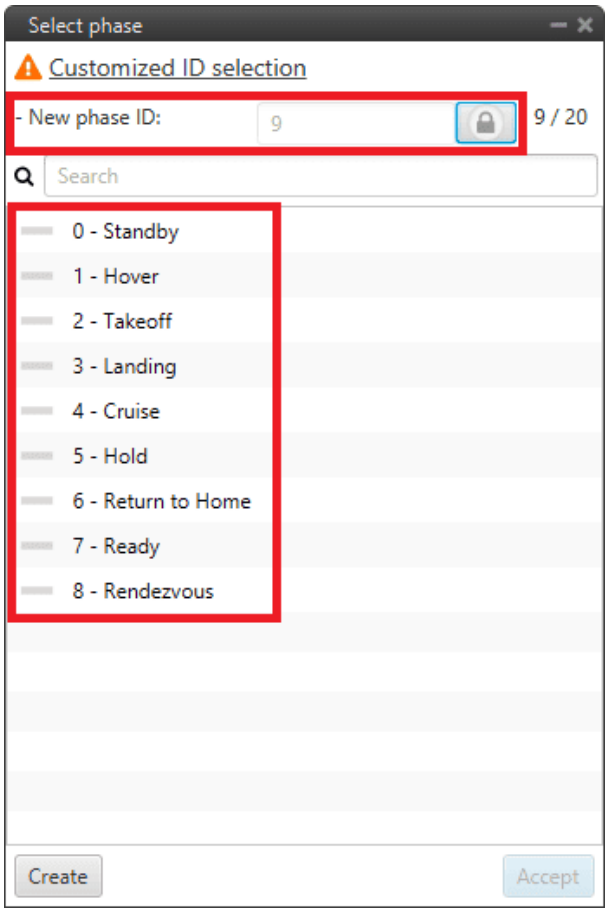


Fig. 134: **Phases panel - Create phase**

- To add a **phase already created**, select the desired phase and click **Accept**.
- By default, when a **new phase** is created, it is assigned with the consecutive ID.

  However, if the user wants to create a new phase with a specific ID, the *New phase ID* option can be "unlocked" by holding down  until the process is fully completed (during the unlocking process this icon  is "painted").

  Once unlocked (when it appears as ), the user can assign the desired ID to the new phase.

  Finally, click **Create** and the new phase will be added to the list of phases.

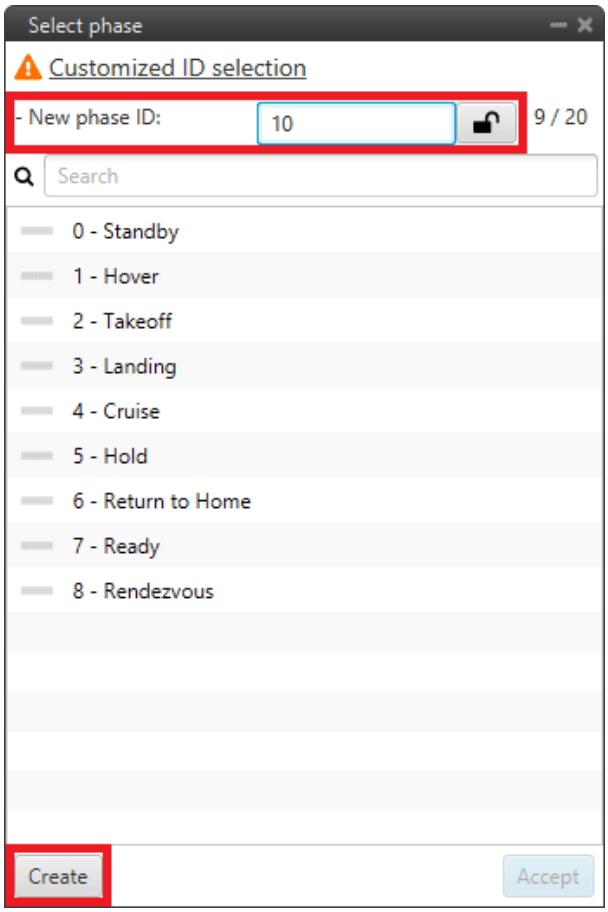


Fig. 135: **Phases panel - New phase ID**

In addition, by *right clicking* on the phase, the user can rename, copy or remove it:

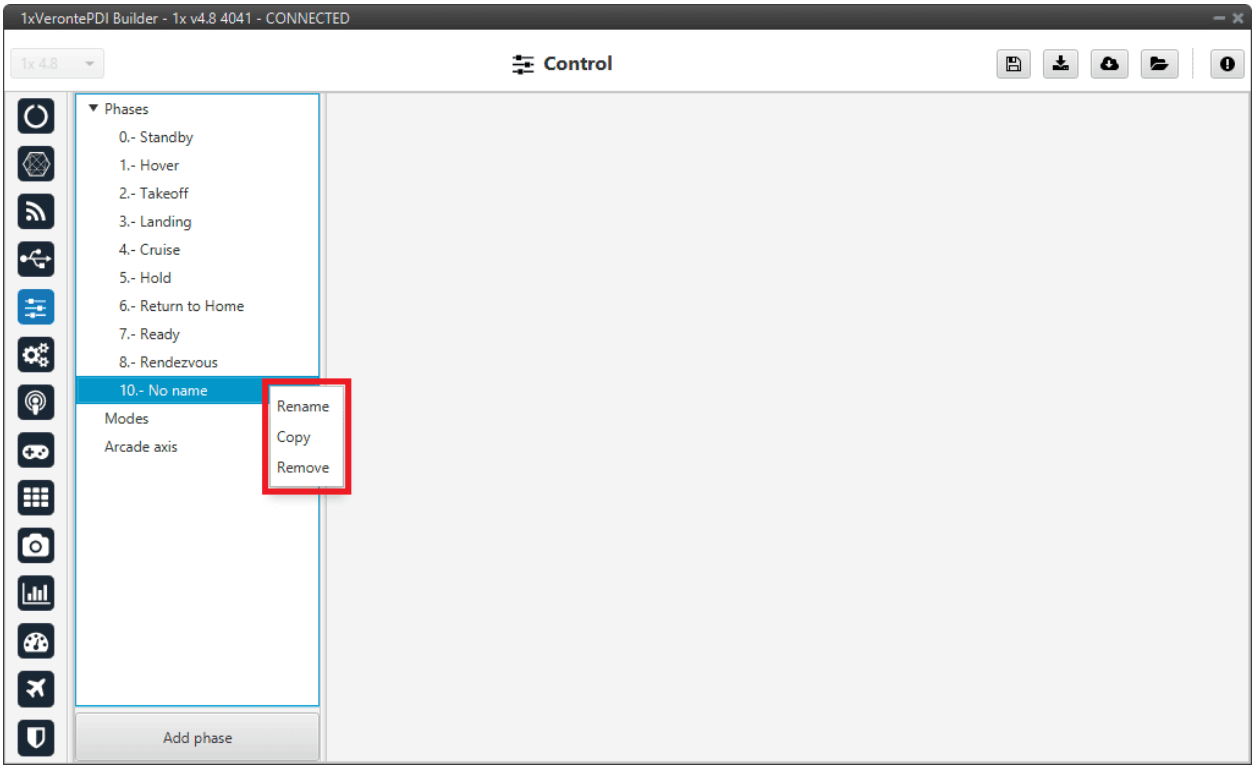


Fig. 136: **Phases panel - Phase options**

Note: The configuration of the phases (guidance and control commands) is done in the *Block Programs menu*.

2.5.2 Modes

2.5.2.1 Modes

This tab allows the creation of custom flight modes. The flight modes determine who is in charge of controlling each one of the aircraft control channels.

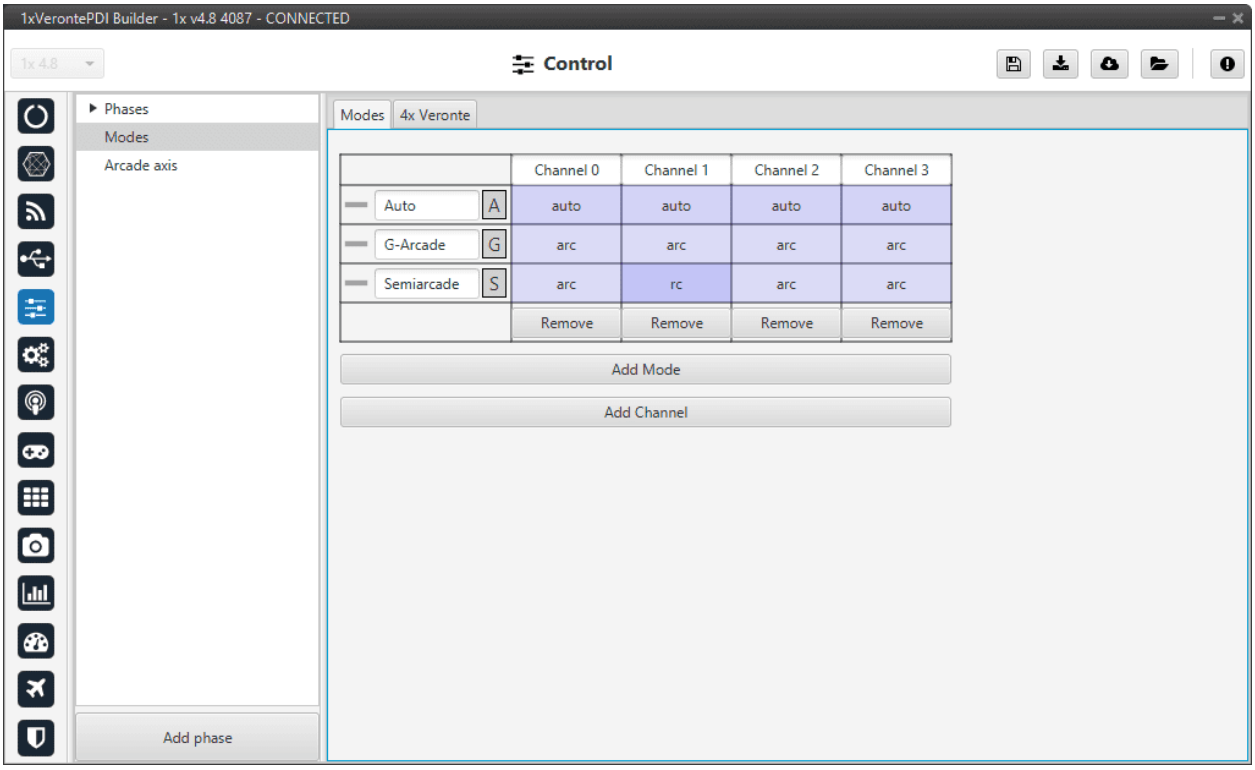


Fig. 137: **Modes tab**

There are 4 different control modes and it is possible to combine them to create custom flight modes. The options available are:

- **auto**: Automatic mode. The control channel is controlled totally by the autopilot.
- **rc**: Radio Control mode. The control is totally carried out manually. The movements on the pilot stick imply directly movements on the servo linked to that control channel.
- **arc**: Arcade mode. The autopilot aids the radio controller during the flight, i.e it could be considered as a mix between automatic and manual. The movements on the pilot stick are the input values on the control system, so the pilot commands a desired pitch, roll, IAS, heading and so on, and is the control system who is in charge of making the platform follow those commands.
- **mix**: In this mode, it is possible to select in which step of the controller will enter the pilot command.

Example

For example, the pitching of an aircraft is commonly controlled with 3 PID being: flight path angle, pitch and pitch rate. In the arcade mode the pilot command will be a desired flight path angle that enters as input of the whole control system, but in the Mix mode is possible to select where we want the command to enter, so the pilot command could be pitch (entering in the second PID directly) or pitch rate (entering directly on the third PID).

The control system will take this input as a disturbance that it wants to discard because the final objective is to match the input of the first PID (a desired flight path angle in this case), so the Mix mode can be used to make small corrections when the aircraft is following a route for example, where we want it to move slightly towards a certain direction by introducing a value directly on the roll PID.

To **change** any of this options, **click on the cell** the user would like to change and the next option will be set.

Warning:

- The name of the mode does not have to correspond to the configuration of the mode.

  For example, the user can **name** the mode as **Auto** but set the **channels** as **rc** (manual):

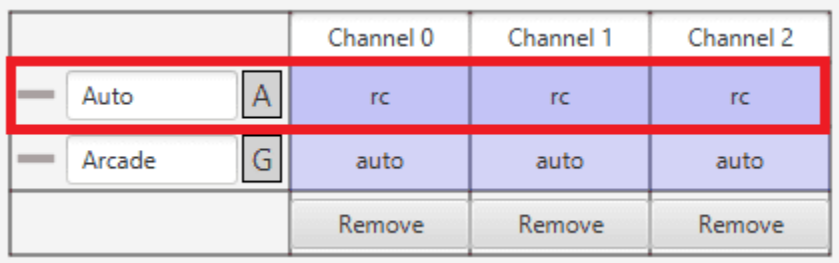


Fig. 138: **Modes configuration**

- Moreover, although the mode is set "sensefully" here, in the block configuration (*Block Programs menu*) the control does not have to correspond to this.

  For example, if a channel is configured as manual (rc) here but then the control is configured so that the stick input does not control the channel, it will be auto control even though manual is specified. See the following example, where for consistency, the blocks in the 'True' and 'False' cases should be inverted:

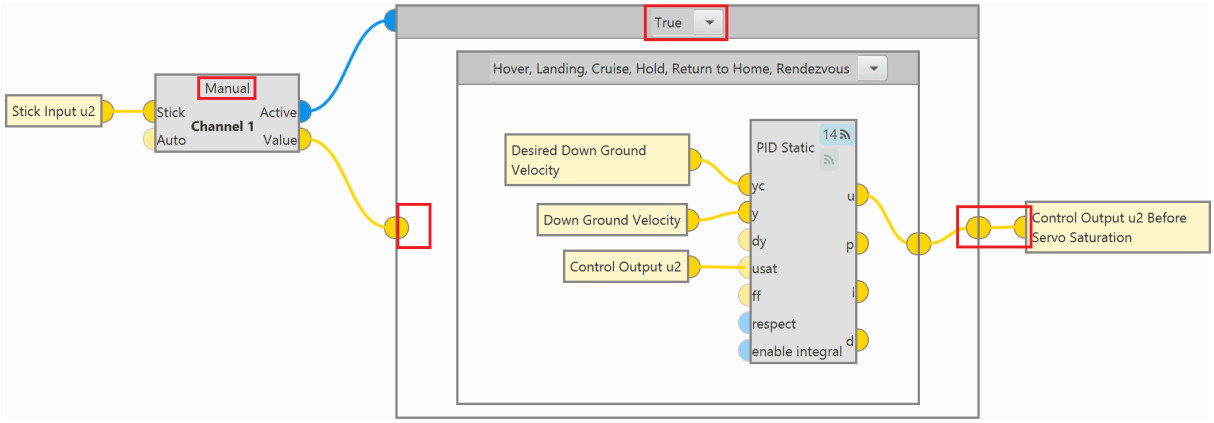


Fig. 139: **Modes configuration in blocks**

So, it is the user's responsibility to build the configuration correctly. In case of having any questions, the user should contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).

2.5.2.2 4x Veronte

This tab allows the user to configure an **Autopilot 1x to operate in an Autopilot 4x**.

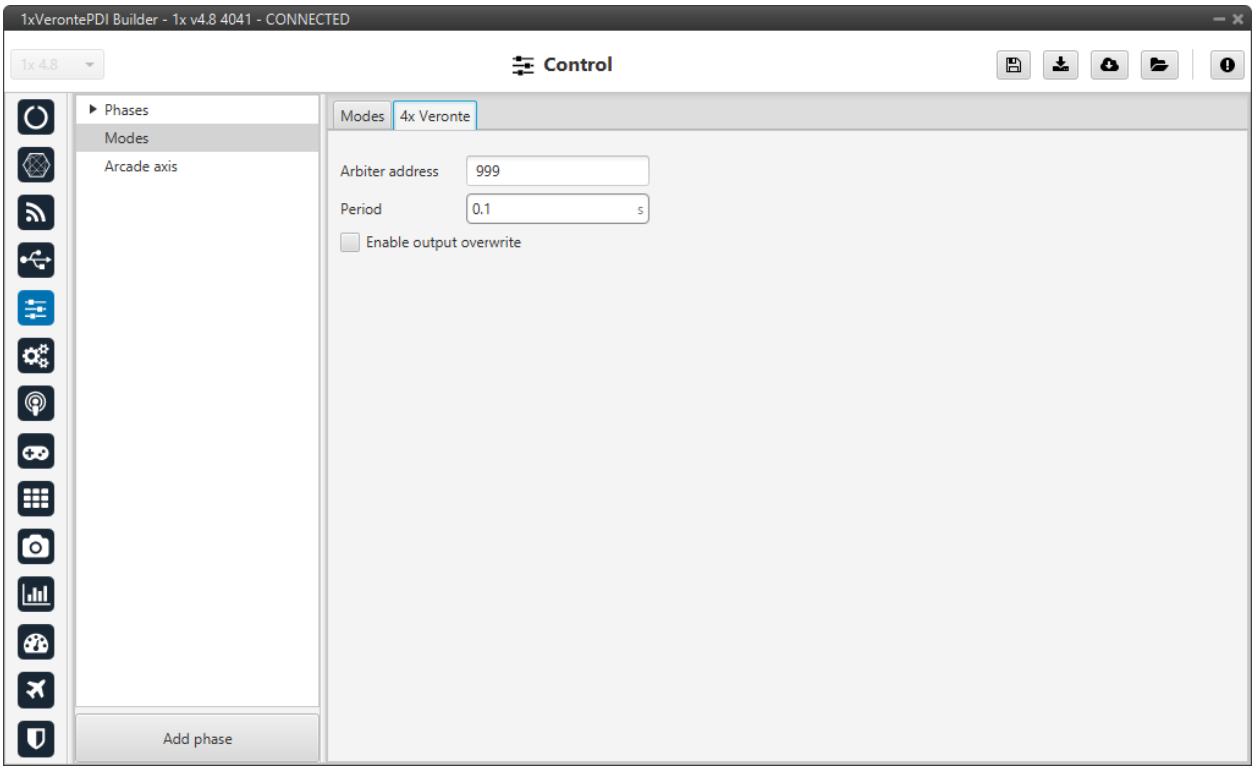


Fig. 140: **4x Veronte tab**

- **Arbiter address**: By adding the arbiter address, **Veronte Ops** will recognise it as part of a 4x unit group, and it will also be possible to do HIL simulations (with **Veronte HIL Simulator**) with this 4x group.

  Note: If the arbiter address is set to 999, there is no arbiter.

- **Period**: Sending period of CAN 4x messages. For more information on the transmission of CAN 4x messages, refer to the *CAN Setup - Input/Output* section of this manual.
- **Enable output overwrite** checkbox: Allows the output to be overwritten when checked.

By enabling it, a table can be created in which columns correspond to each Autopilot 1x and the CanID row to the ID of the CAN message through which each Autopilot 1x communicates information with the other Autopilots 1x within the Veronte Autopilot 4x. Each CAN Id is associated to the **CAN 4x** producer/consumer of the AP in which it is configured. For more information on **CAN 4x**, refer to the *CAN Setup - Input/Output* section of this manual.

Important: This option must work in conjunction with the *AP Selection block*.

An example of configuration is presented below:

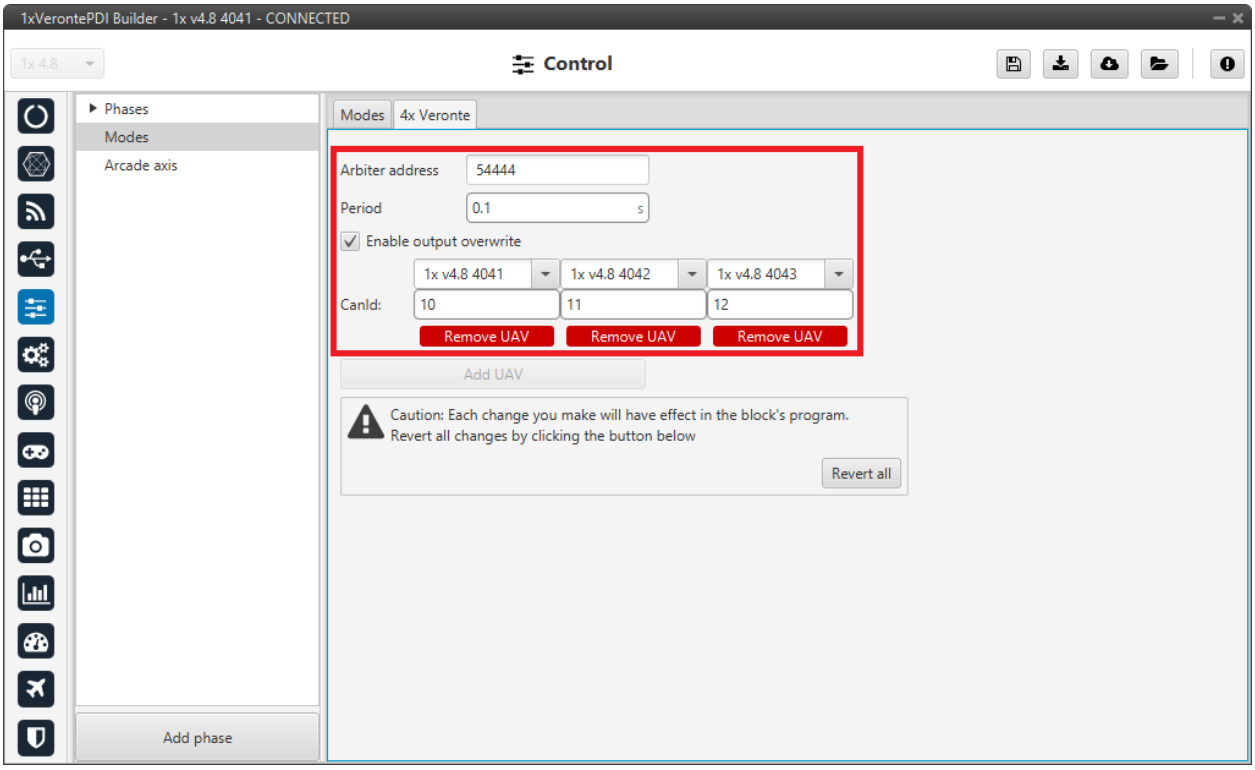


Fig. 141: **4x Veronte panel - Example of use**

So in this example, **Autopilot 1x with address 4041** will share information with Autopilots 1x with addresses 4042 and 4043 via CAN messages with **ID 10**.

2.5.3 Arcade axis

Arcade axis panel enables the option of changing the center of the system axes. This option is used to create axis systems referred to a certain point or direction, for example, it is useful when the pilot wants all movements to be made with respect to them (Ground axes). In this way, if the pilot commands a right turn, the aircraft will turn to the right relative to the pilot, rather than the right relative to the aircraft (Body axes).

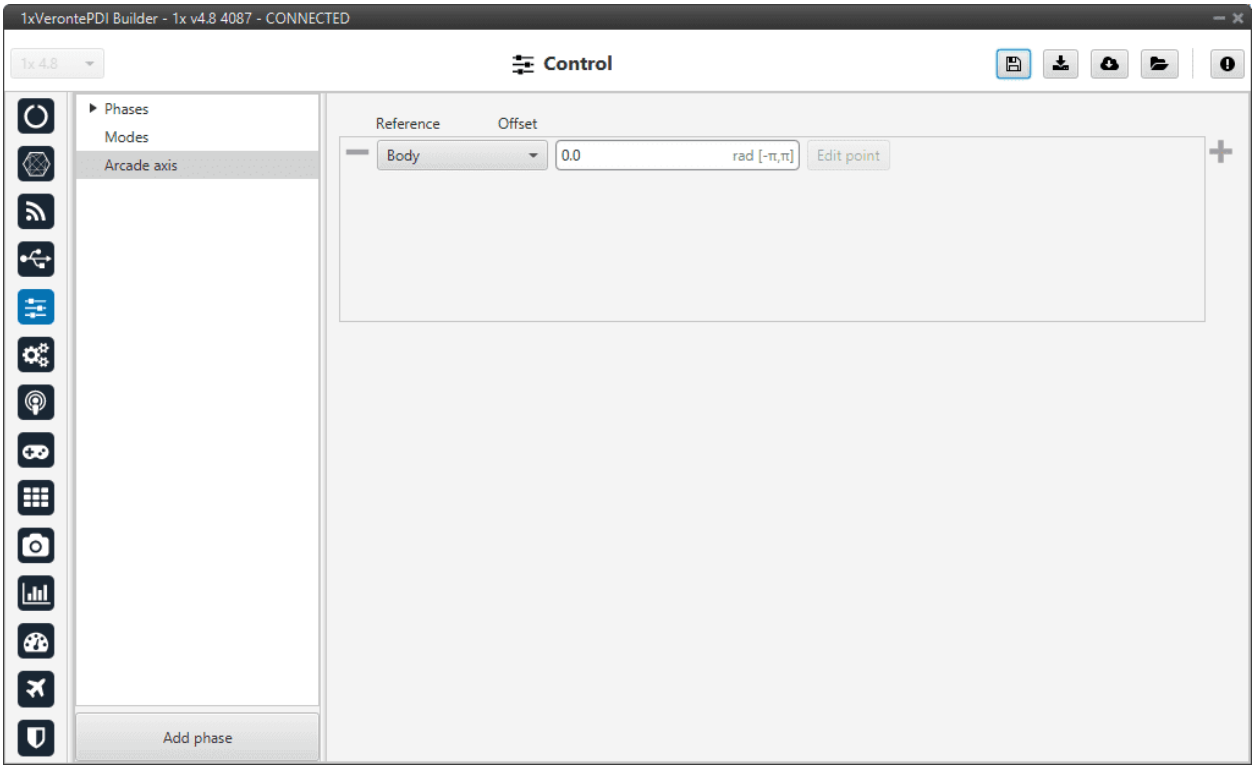


Fig. 142: **Arcade axis panel**

It is possible to add up to 5 axes systems, being able to choose between the following types:

- **Body**: Fix the axes in the UAV. It is standard for the pilot.
- **Ground**: Fix the axes in the 1x GND unit.
- **Point**: Fix the axes in a point that user defines.
- **Heading**: Fix the axes in the the heading defined.
- **Desired heading**: Fix the axes in the desired heading.
- **Tangent direction**: Fix the axes in the tangent direction of the designed path.
- **Desired yaw**: Fix the axes in the desired yaw.

An automation can be used to select an Arcade Axis in flight. For more information, check the *Actions - Automations* section of this manual.

2.6 Automations

Automations are actions that are carried out when a combination of events happen, i.e when the events are accomplished the action is done. An example of what an automation could be a change of phase when reaching a certain altitude and speed, moving a servo when a button is clicked and many other possible combinations.

In this section all the possible events and action will be explained in detail, so the user can combine them to create the automations that best suit their needs.

The following figure shows the layout of the automations menu, with a column for the events and another for the actions linked to these events.

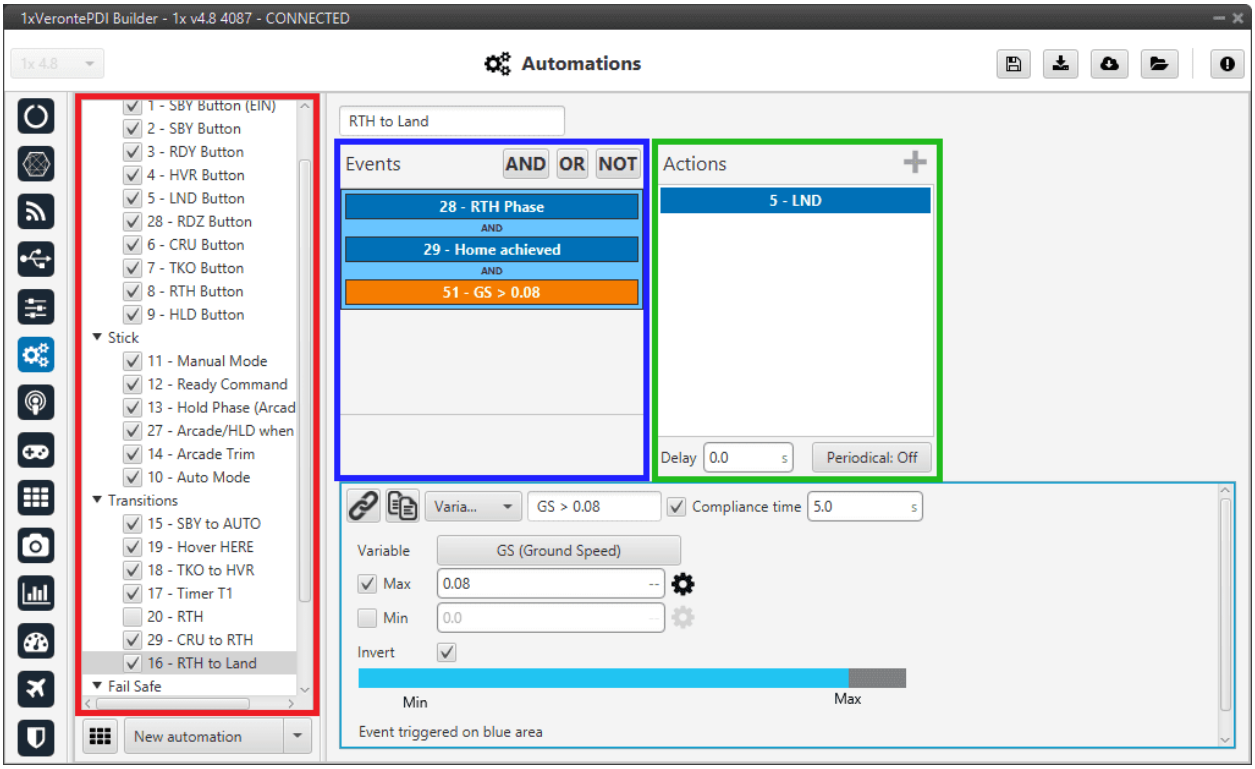


Fig. 143: **Automations menu**

All the automations that have been created (red) are a combination of events (blue) and actions (green). All actions will be performed on event or an event combination triggering.

There are some parameters that can be configured in the events and actions menu and which are applicable independently of the type of event/action configured.

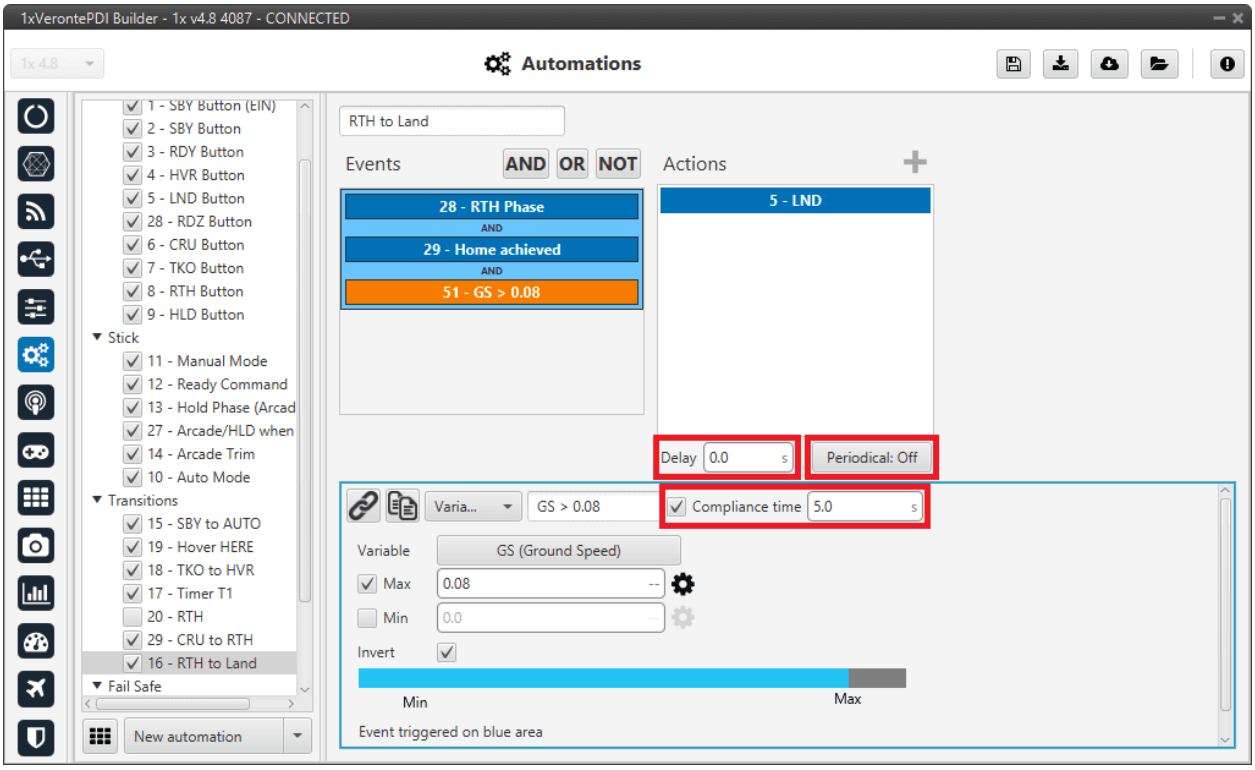


Fig. 144: **Automations menu - Parameters**

- **Compliance time**: It is a value related to the automations. It indicates how much time the event has to be accomplished in order to trigger the action.
- **Delay**: It is the time between the triggering of the event and the beginning of the action.
- **Periodical**: If enabled, it allows to configure actions to take place periodically during the time that the events are active.
  - **Period**: Manually enter the desired periodicity for this action.
  - **Type**: The action can be configured to take place each certain:
    - **Distance**: When using distance, the option **Vector** allows to measure that distance along a direction specified by that vector.

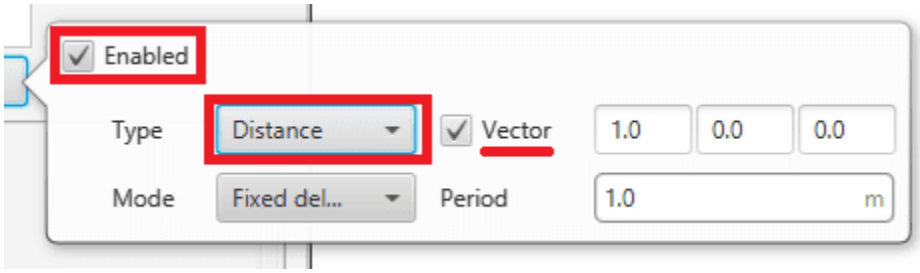


Fig. 145: **Periodical distance configuration**

    - **Time**

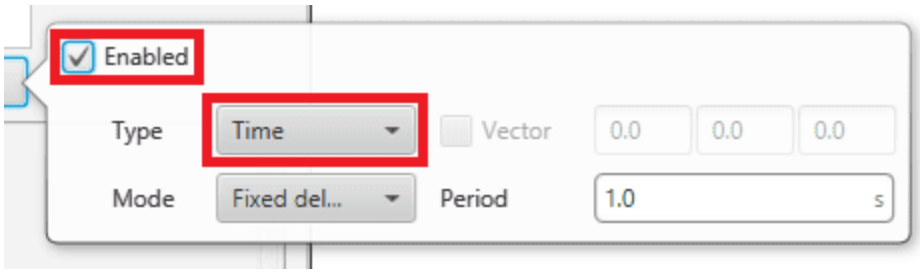


Fig. 146: **Periodical time configuration**

– **Mode**: The two **Modes** available for both time and distance are **fixed delay** and **fixed period**. In order to explain the difference between them, the following figure is presented as an aid to the user.

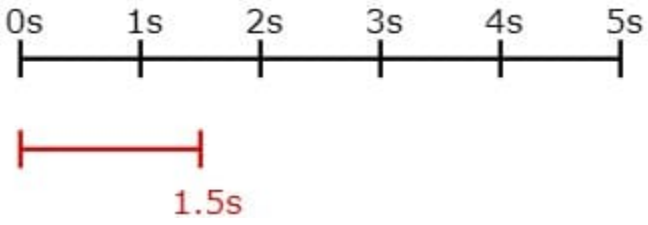


Fig. 147: **Periodical modes**

Let's consider that the system evaluates the automations each second (black line), and the automation that contains the periodical option is configured to be executed every 1.5 seconds (red line). In that case, the **first action** will be **triggered at the second 1.5** but will be **evaluated at second 2**. The **second time** that the action is evaluated will depend on the mode, if it is selected:

* **Fixed delay**, the evaluation of the action will be done **1.5 seconds after it was evaluated** the first time, so that will be at **second 3.5**.
* **Fixed period**, the action will be evaluated **1.5 seconds after the first triggering** (not evaluation) so that would be at the **second 3**.

In the real praxis, the evaluation time for the automation is much lower than 1 second so the difference between the modes is much smaller.

2.6.1 New automation

> **Warning:** It is important to know that there is a limit of **500 events**, **120 actions** and **100 automations**.

To create a new automation press **New Automation** and a new window will be displayed. First, users must add an **event** by selecting a **previous one** (if it exists) or **creating** new one.

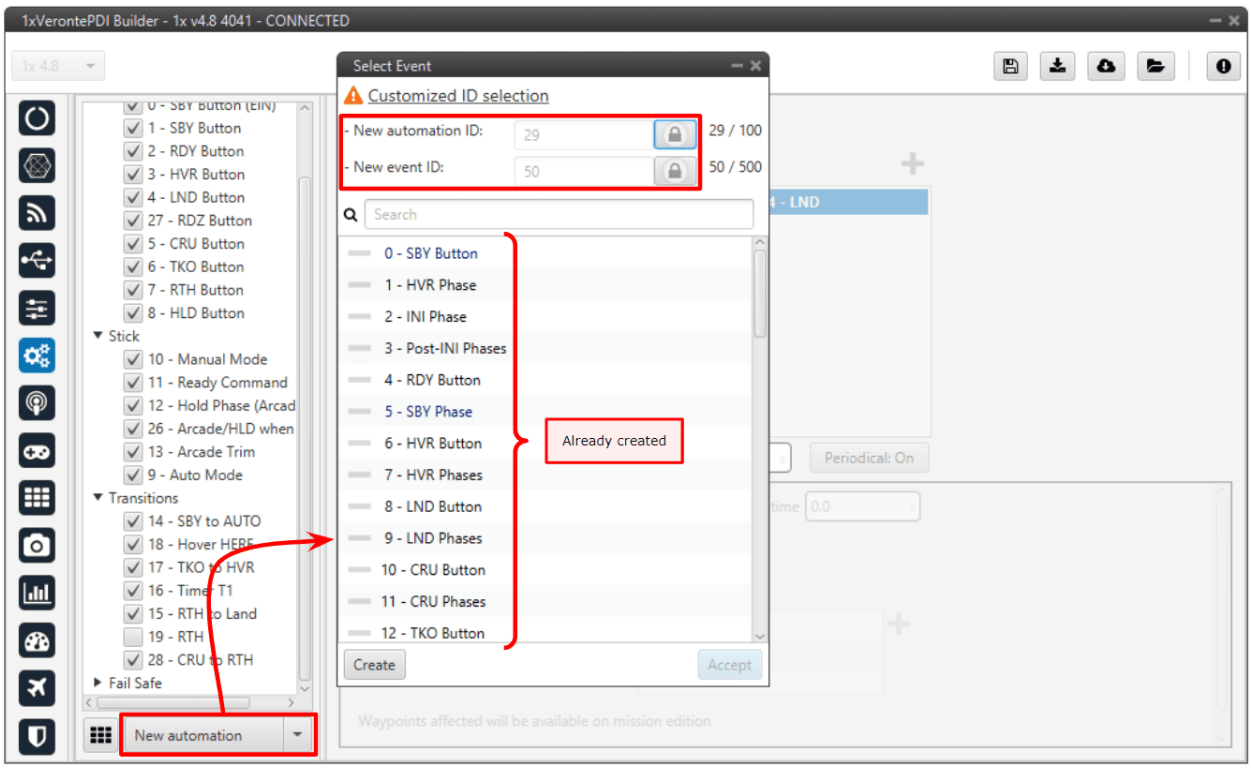


Fig. 148: **Automations menu - New event**

- To add an **event already created**, select the desired event and click **Accept**.
- By default, when a **new automation** is created, it is assigned with the consecutive ID.

  However, if the user wants to create a new automation with a specific ID, the *New automation ID* option can be "unlocked" by holding down  until the process is fully completed (during the unlocking process this icon  is "painted").

  Once unlocked (when it appears as ), the user can assign the desired ID to the new automation.

  Finally, click **Create** and the new automation will be added to the list of automations.

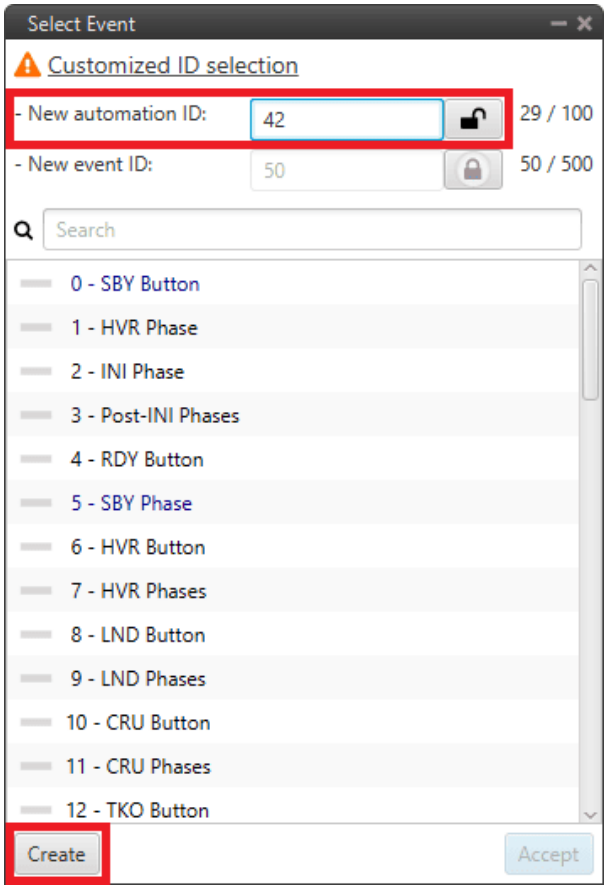


Fig. 149: **Automations menu - New automation ID**

Caution: When the user enters an ID that is already assigned to an automation, the following confirmation message will appear:

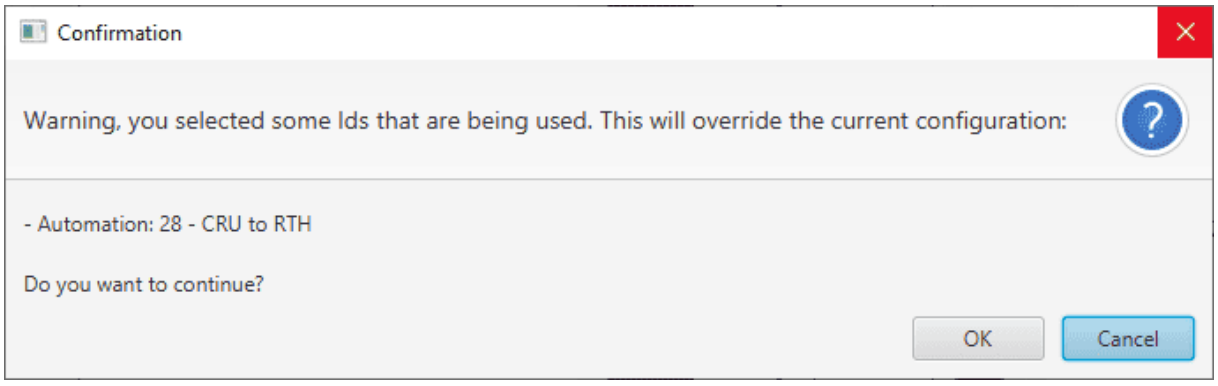


Fig. 150: **Automations menu - New automation confirmation message**

Then, if the user presses 'OK', the existing automation will be removed and a new automation with the entered ID will be added.

To add an action to the automation, press the + icon and a new window will appear. Here, users can choose to either select a **previous action** (if it exists) or **Create** a new one.

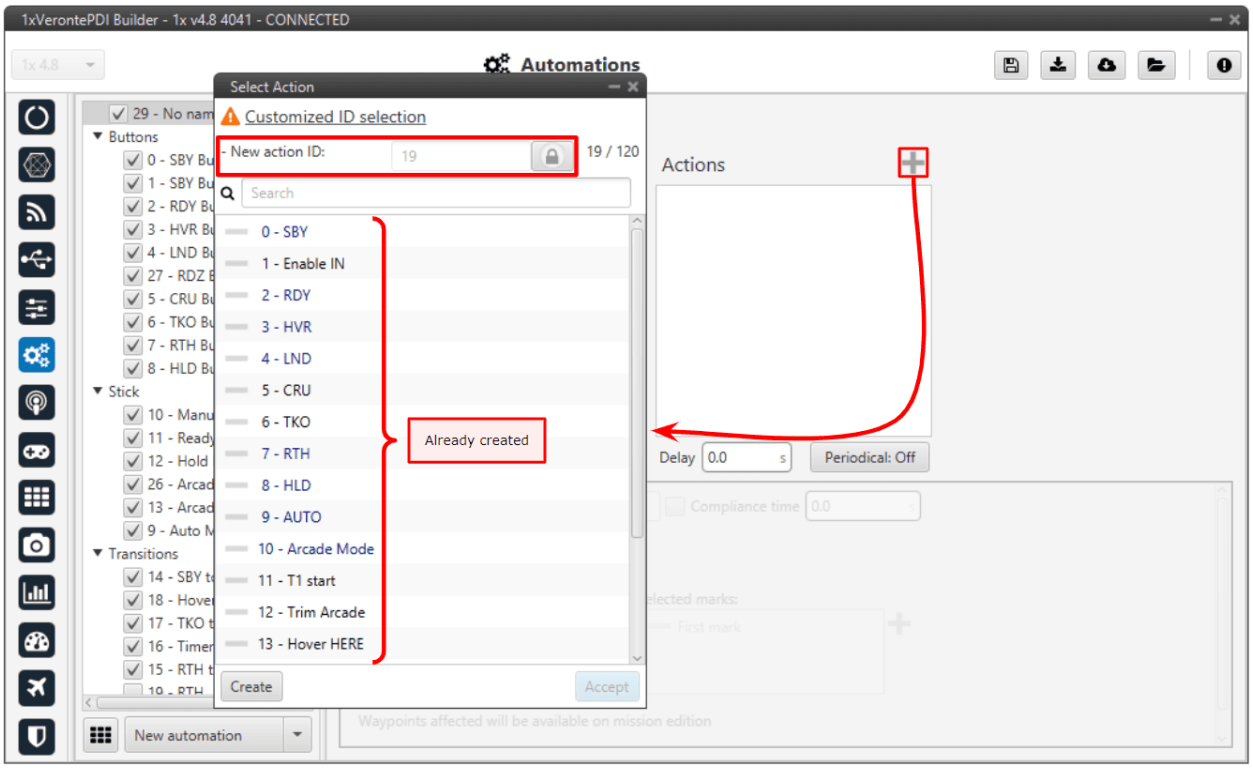


Fig. 151: **Automations menu - New action**

- To add an **action already created**, select the desired event and click **Accept**.
- By default, when a **new action** is created, it is assigned with the consecutive ID.

  However, if the user wants to create a new action with a specific ID, the *New action ID* option can be "unlocked" by holding down  until the process is fully completed (during the unlocking process this icon  is "painted").

  Once unlocked (when it appears as ), the user can assign the desired ID to the new action.

  Finally, click **Create** and the new action will be added.

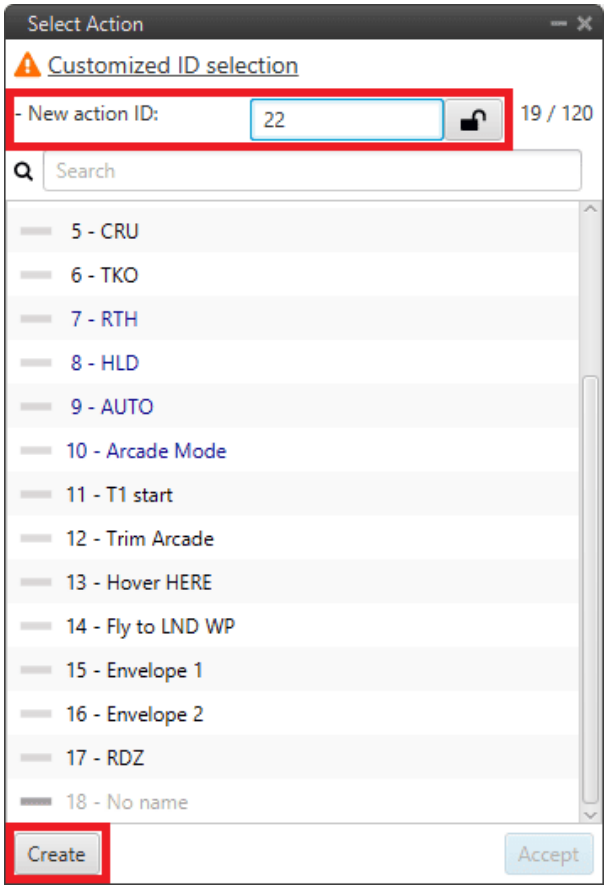


Fig. 152: **Automations menu - New action ID**

Caution: When the user enters an ID that is already assigned to an action, the following confirmation message will appear:

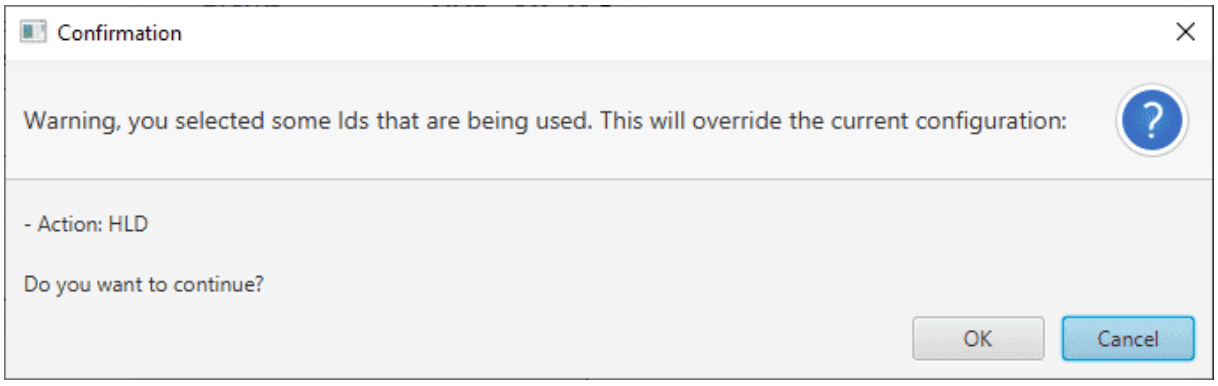


Fig. 153: **Automations menu - New action confirmation message**

Then, if the user presses 'OK', the existing action will be removed and a new action with the entered ID will be added.

When an automation is created, the following options are available:

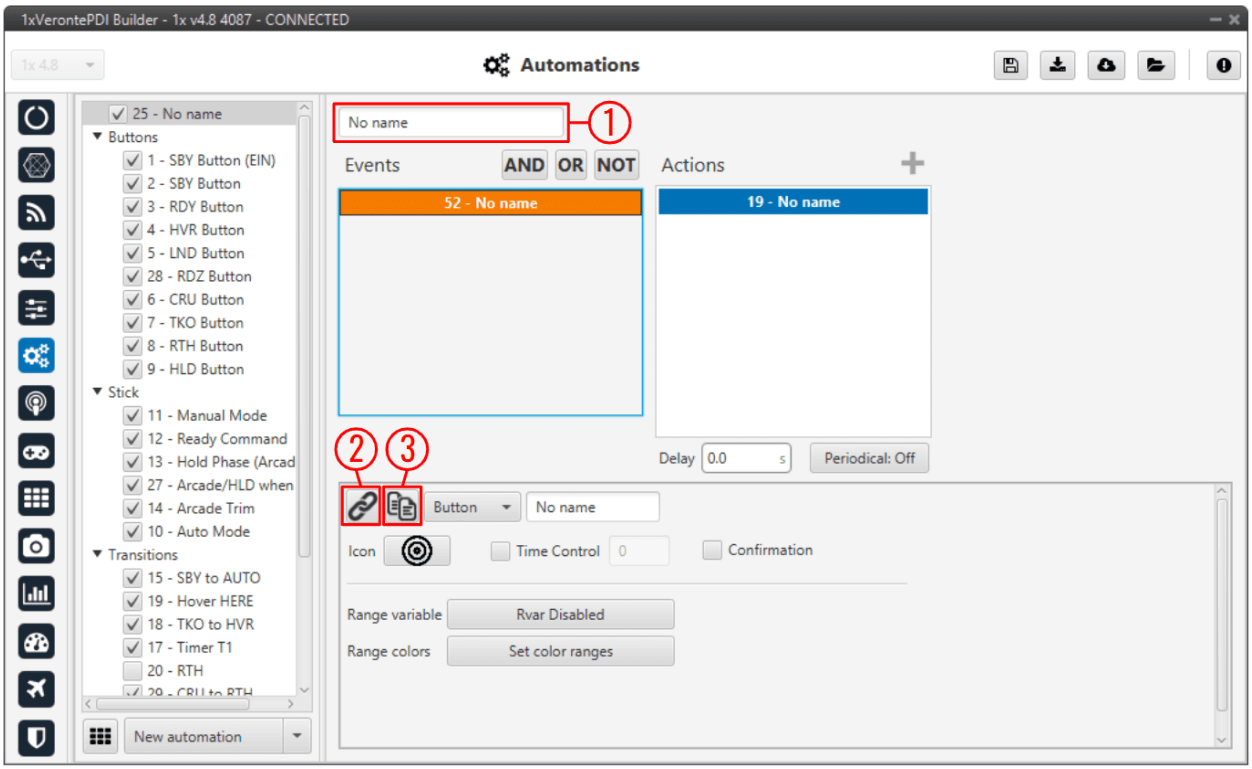


Fig. 154: **Automations menu - Automation options**

1. The user can rename the automation with the name of their choice.
2. **Use existing** button  : Select an action or event from the available in the system. When modifying an action or event it will be **modified in all automations where it is in use**.
3. **Clone** button  : Clone an existing action or event creating a new one with same parameters configured on the start point.

By right clicking on an automation it is possible to **remove** it, **clone** it or **change it of group**. When a group is created, the rest of automations that the user wants to add to the group can be done by drag and drop.

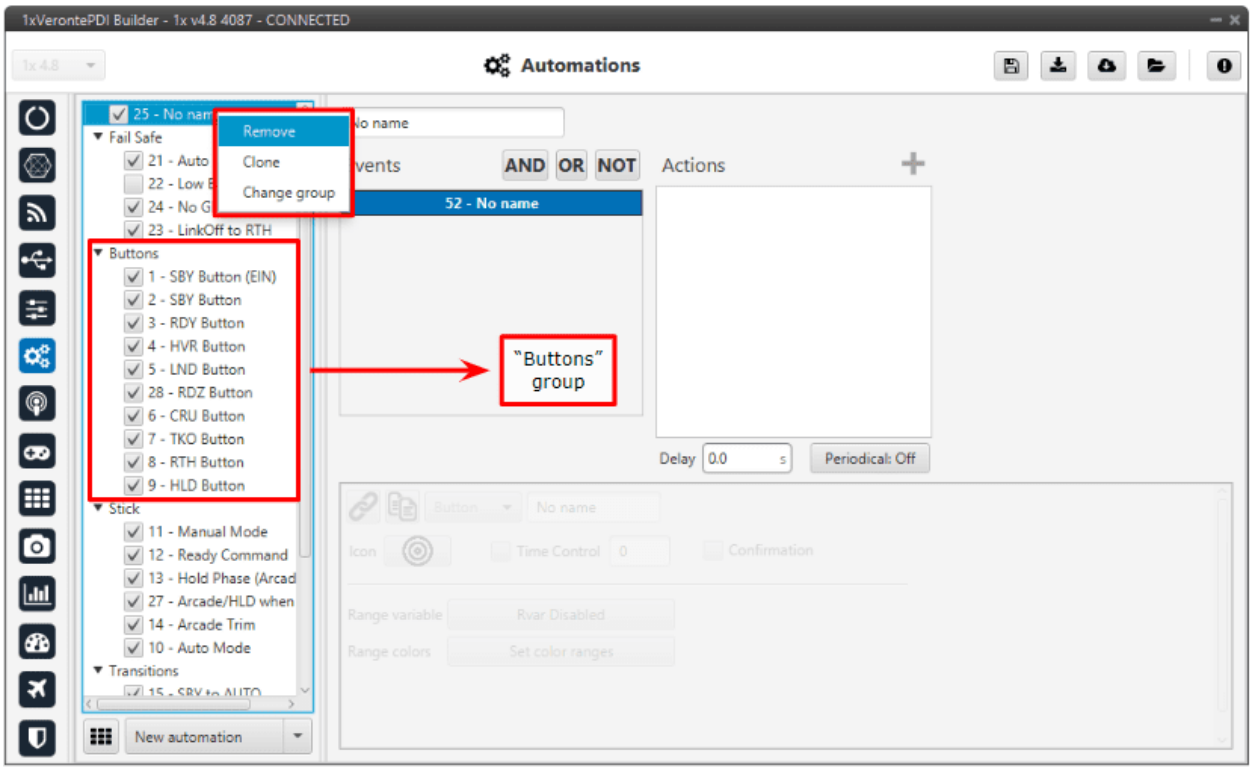


Fig. 155: **Automations menu - Automations groups**

Warning: When creating a clone of one automation, changes made in the **event panel** will be applied to the other one and vice versa, while the actions can be different in each automation.

2.6.2 Other options

In the figure below, the user can see 2 additional options:

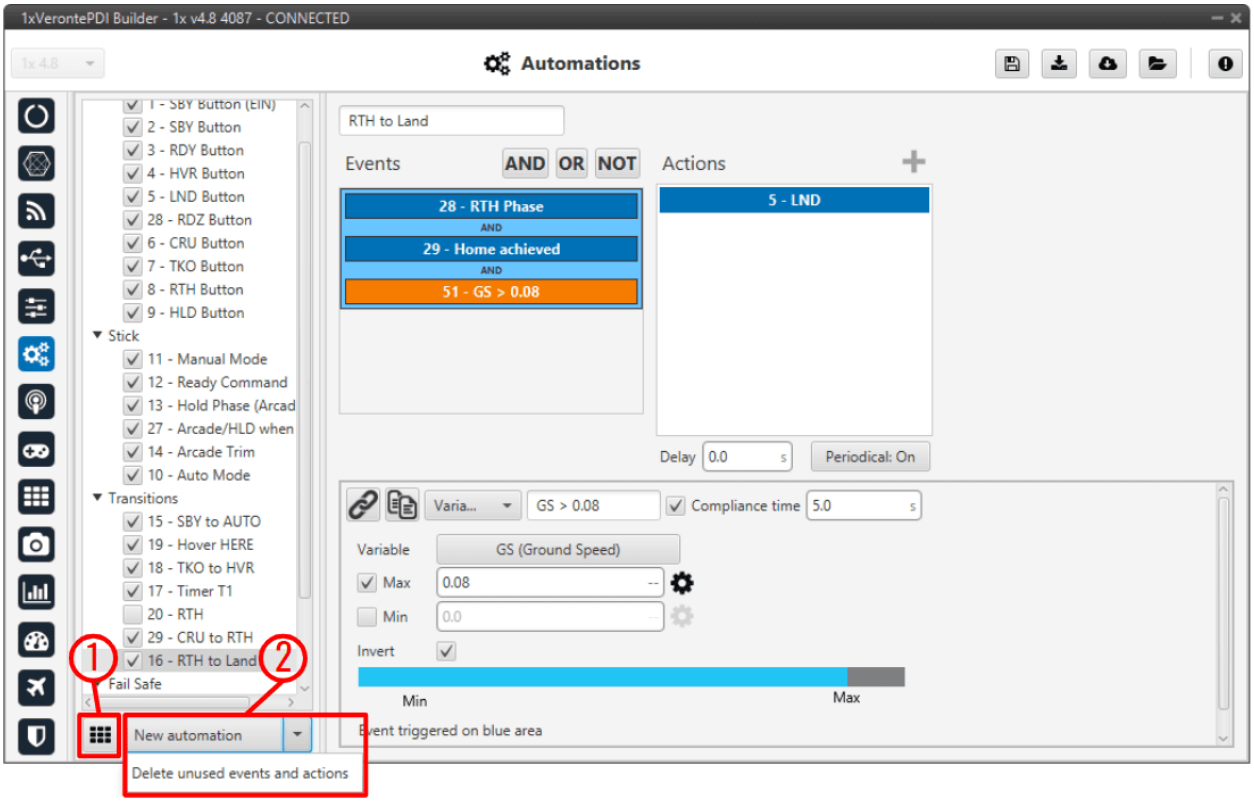


Fig. 156: **Automations menu - Other options**

1. By clicking here, a phase transitions table will appear:

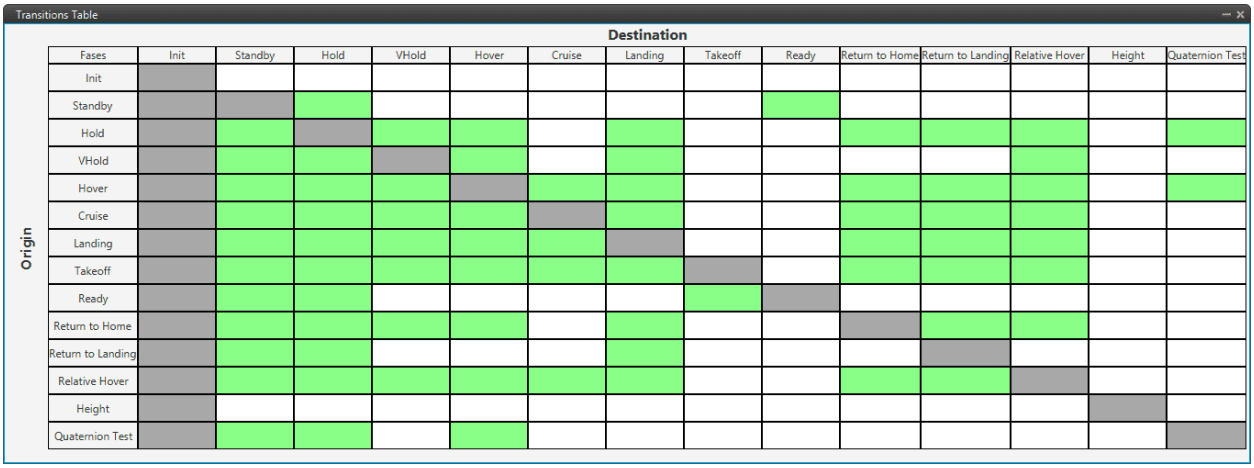


Fig. 157: **Automations menu - Phase transitions table**

In this table, the transitions between each phase can be visualized.

In addition, by clicking on a cell, users can see which automation (and the events) makes the transition between the two phases possible.

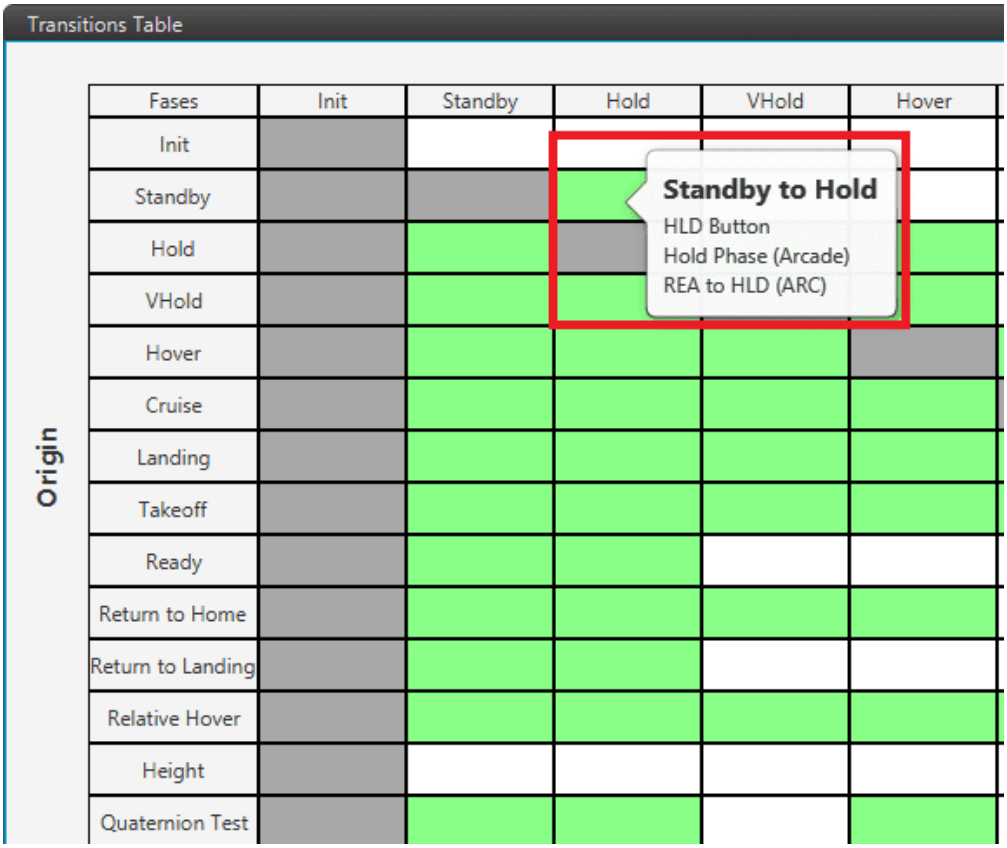


Fig. 158: **Automations menu - Automation example of phase transitions table**

2. **Delete unused events and actions**: This option deletes those created events or actions that are not in use in any automation.

2.6.2.1 Events

An event is a condition, or set of circumstances, that must occur to trigger determined actions. All the events can be combined to create a custom event, using the **boolean operations** provided by the software (**AND**, **OR**, **NOT**).

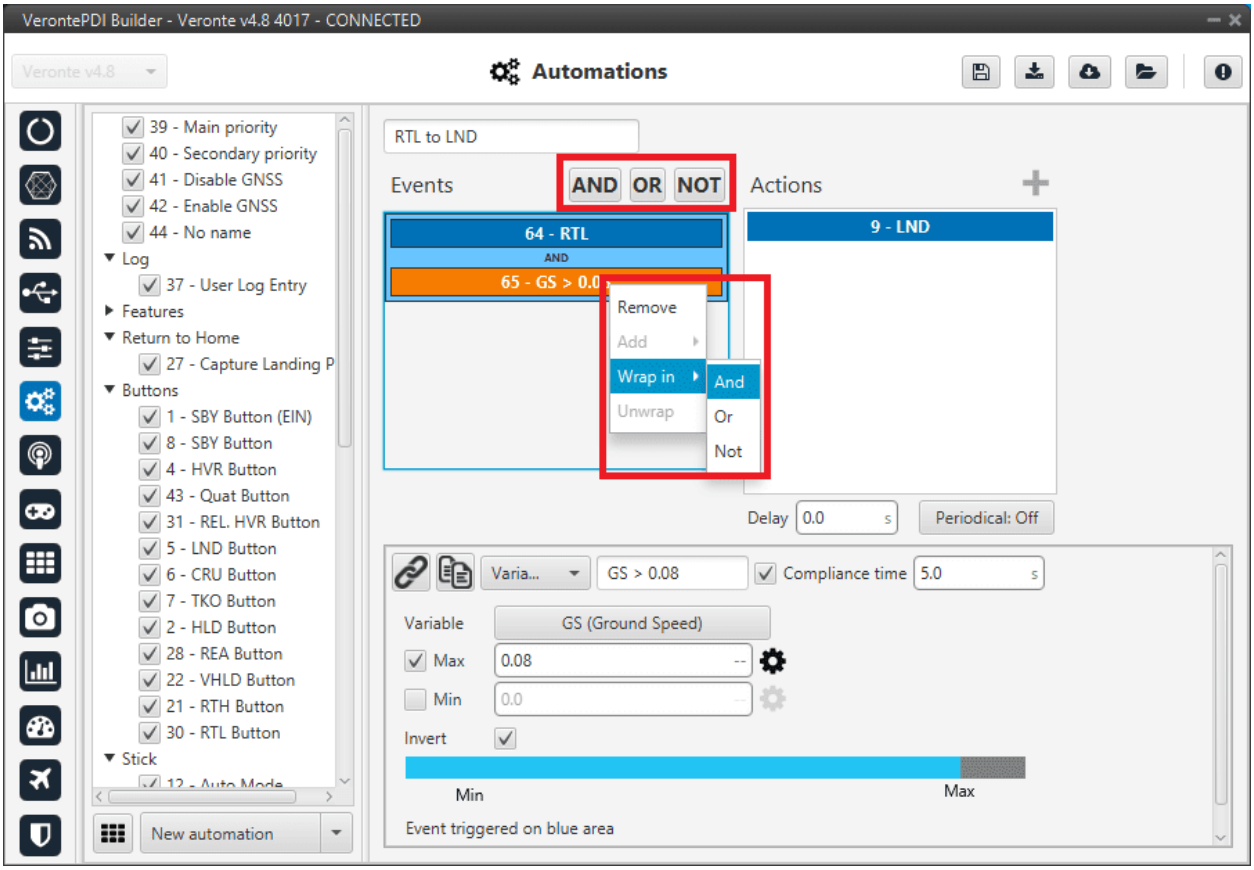


Fig. 159: **Events options**

The following table depicts the meaning of each one of the boolean operators.

Logics	Description
AND	All events grouped on an AND should be accomplished simultaneously in order to activate the automation.
OR	One of the events in the group should be accomplished for activating the automation.
NOT	The event will be active meanwhile the event or event group is not accomplished.

When there is only one event, clicking on the boolean command will create another event linked to the other one according to that operation. By right clicking on an event and selecting Wrap in allows the creation of an operation as if it was inside brackets, i.e it will be evaluated first. Let's consider the following event group as an example.

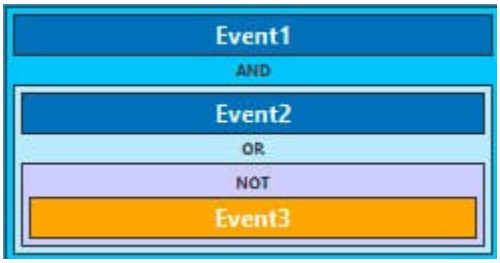


Fig. 160: **Events wrapped**

The first operation that is evaluated is the NOT, then the OR between Event2 and the result of the NOT, and finally the AND between Event1 and the result of the OR.

When creating a new event it is possible to choose from one of the **previously created** on the system or to **create** a **new** one.

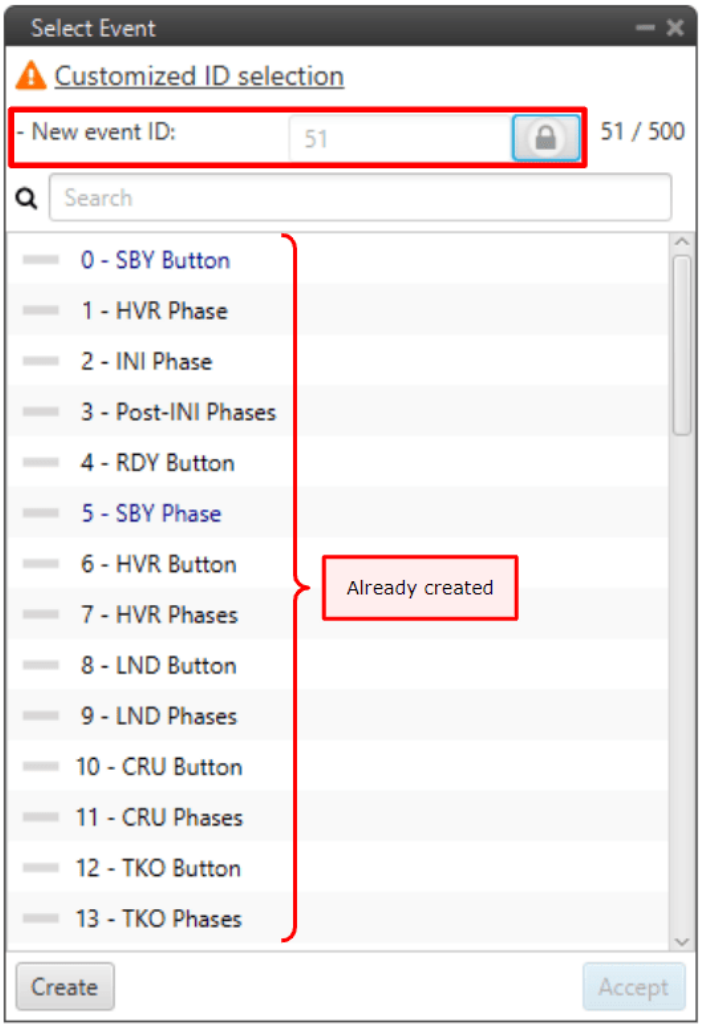


Fig. 161: **New event**

- To add an **event already created**, select the desired event and click **Accept**.
- By default, when a **new event** is created, it is assigned with the consecutive ID.

  However, if the user wants to create a new event with a specific ID, the *New event ID* option can be "unlocked" by holding down  until the process is fully completed (during the unlocking process this icon  is "painted").

  Once unlocked (when it appears as ), the user can assign the desired ID to the new event.

  Finally, click **Create** and the new event will be added.

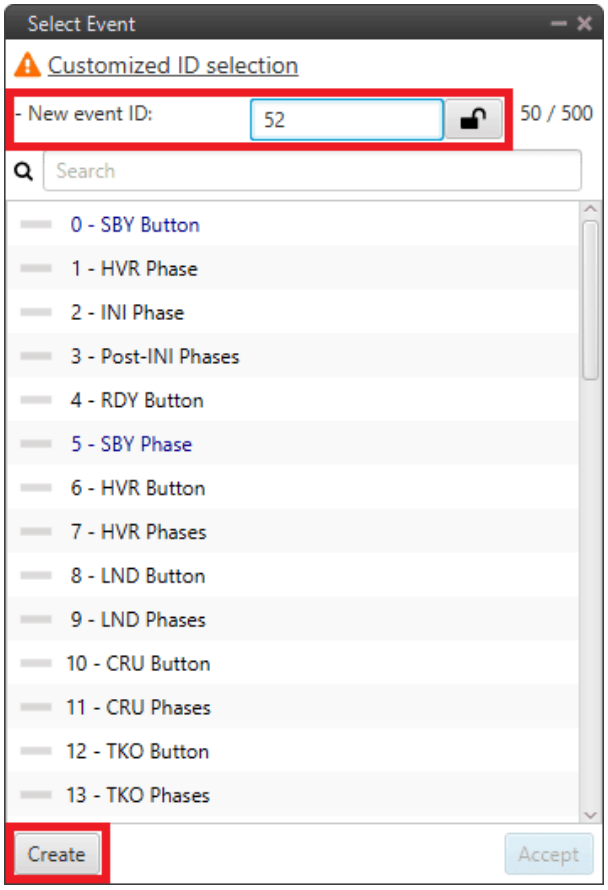


Fig. 162: **Automations menu - New event ID**

Caution: When the user enters an ID that is already assigned to an event, the following confirmation message will appear:

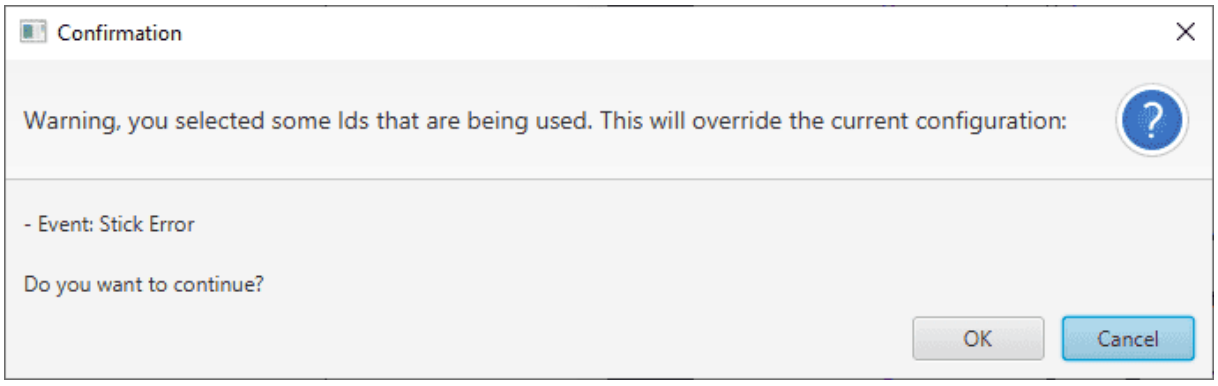


Fig. 163: **Automations menu - New event confirmation message**

Then, if the user presses 'OK', the existing event will be removed and a new event with the entered ID will be added.

The user can also rename the event with the name of his choice.

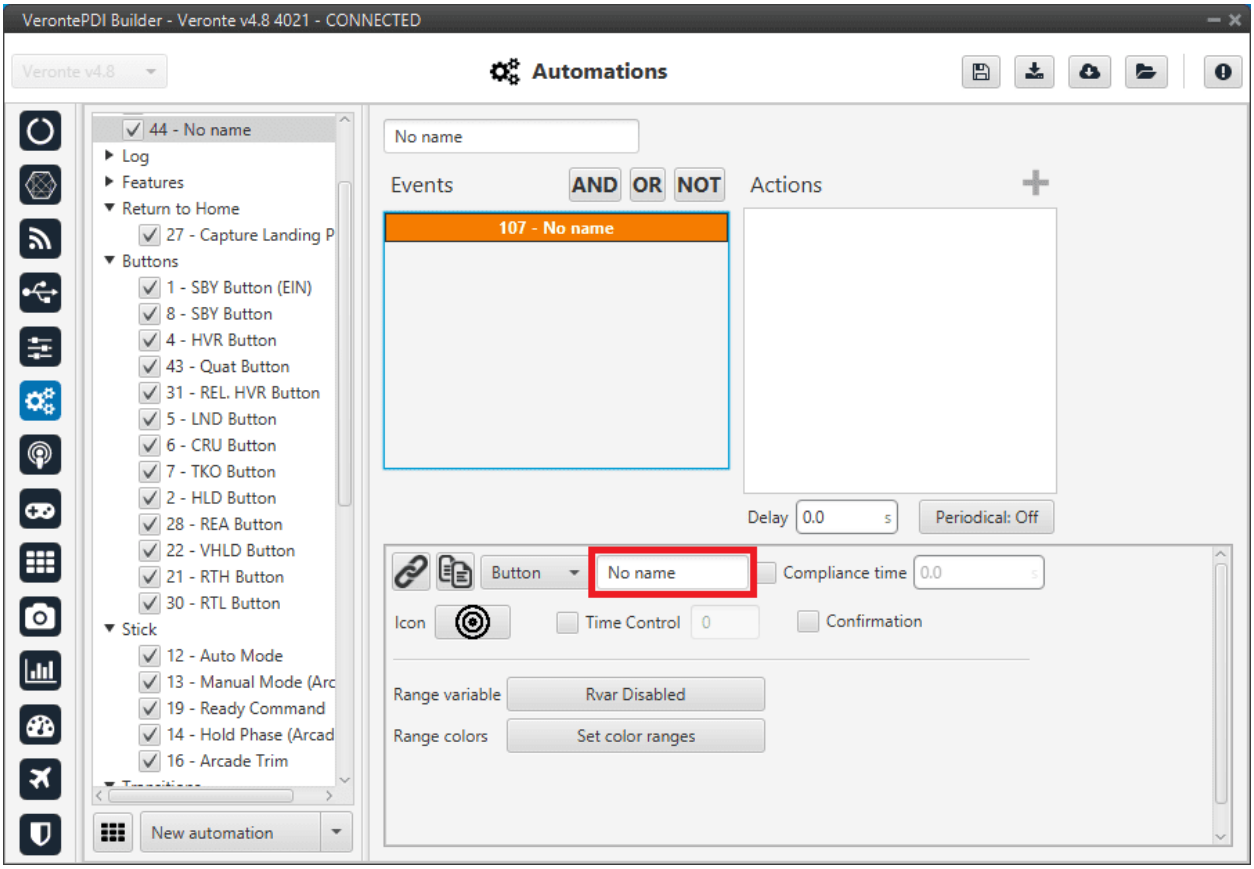


Fig. 164: **New event - name**

The different types of events that can be created are presented below.

2.6.2.1.1 Alarm

This kind of automation allows the user to add any bit of the system as an alarm. Depending on the mode in which it is configured, it will be activated in one way or another.

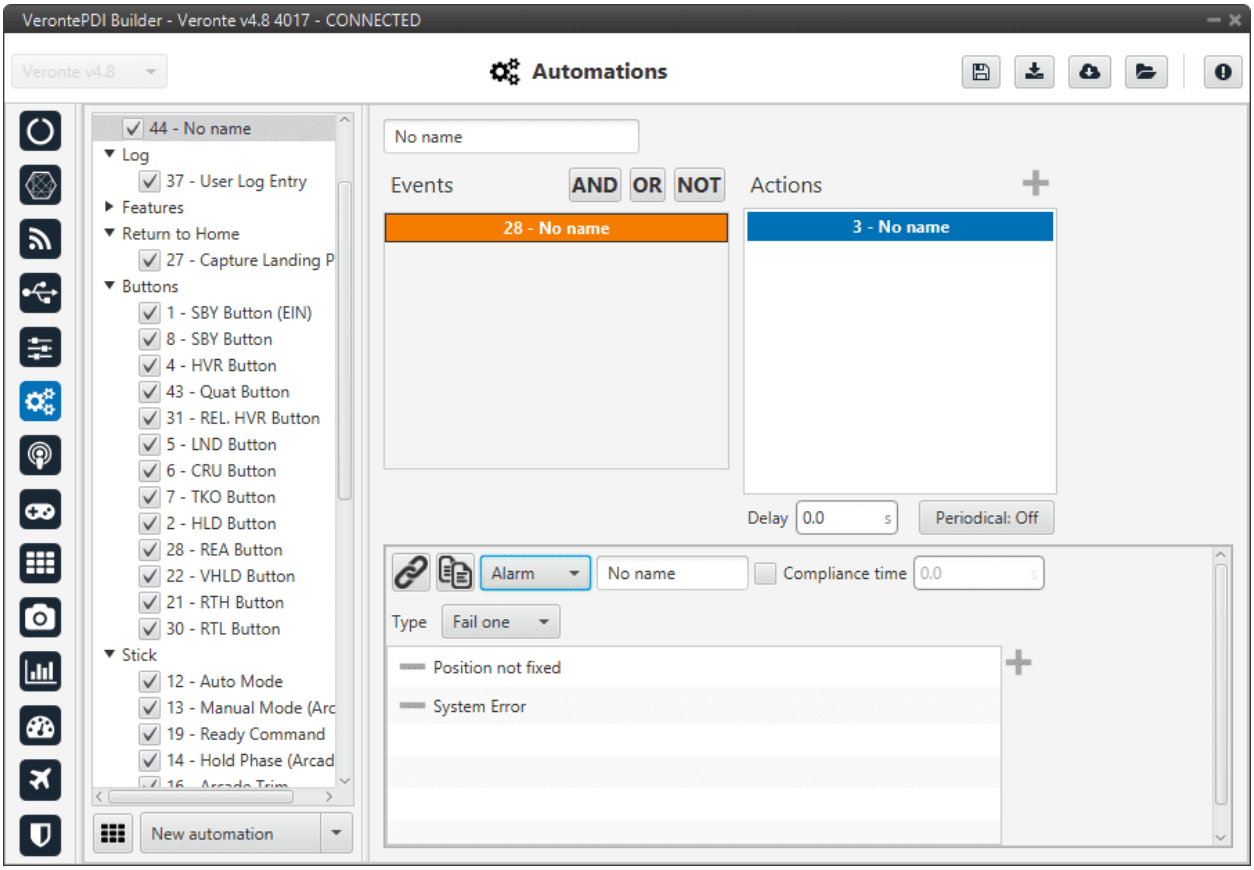


Fig. 165: **Alarm event**

The two possible modes are the following:

- **Fail one**: it is triggered when one of the bits is set to false.
- **All ok**: it is triggered when all bits are set to true.

A common alarm event is the **Position not fixed** in fail one mode, which is triggered when there is not GPS signal in the autopilot.

2.6.2.1.2 Area

The event is triggered when the aircraft is inside or outside an area defined in the mission. For more information on mission creation, take a look at the Veronte Ops manual.

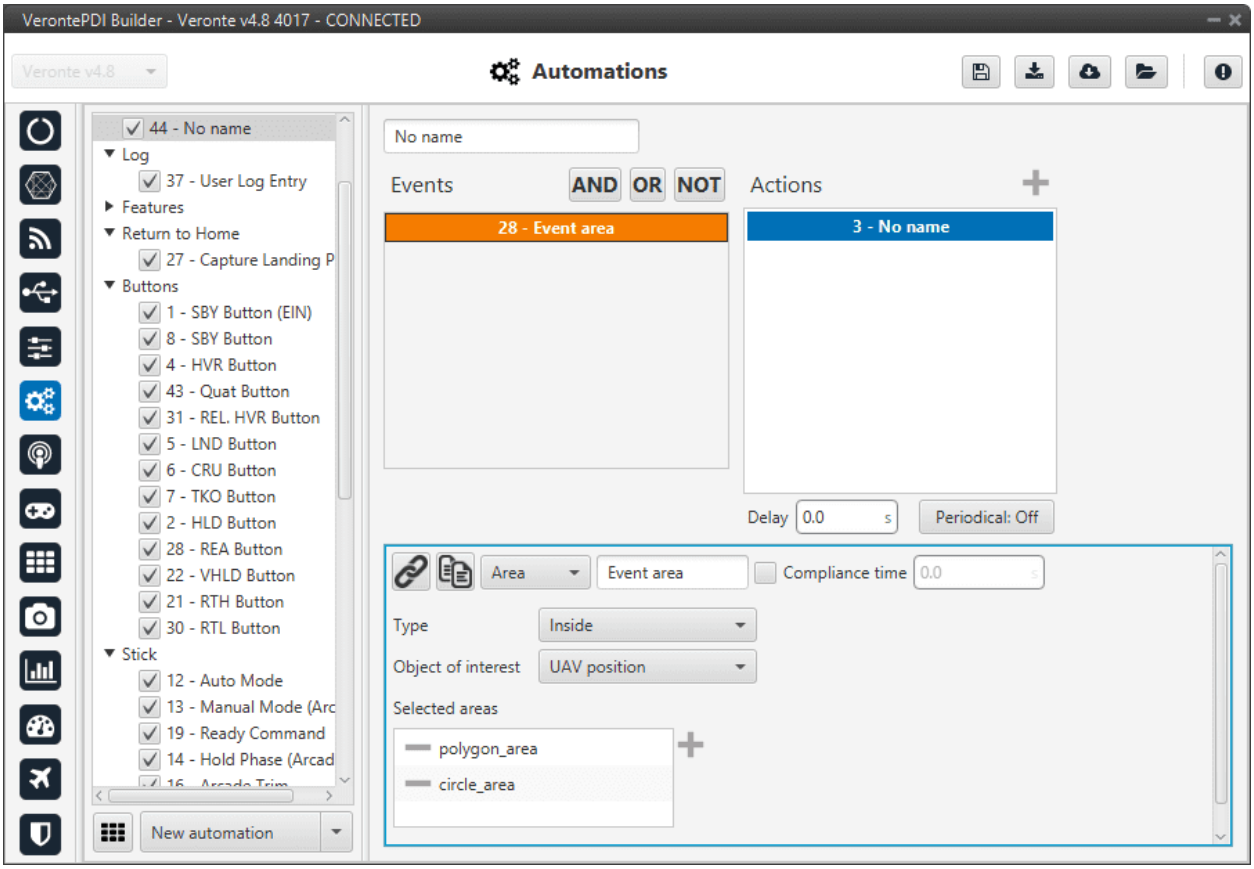


Fig. 166: **Area event**

- **Type**: Inside or Outside.
- **Object of interest**: The user has to select which object is the one that should fulfill the event.
- **Selected areas**: To select an area, first define the desired areas (**prims**, **cylinders** or **spheres**) in the *Operation elements panel* of the **UI menu**.

When the event has been labeled ("Event area" in this case) and saved, it is possible to link it to an area drawn on the map with the **Operation panel** (see more about this at the Veronte Ops manual) .

2.6.2.1.3 Button

This option creates a **button** in **Veronte Ops** software that will trigger the action when it is clicked.

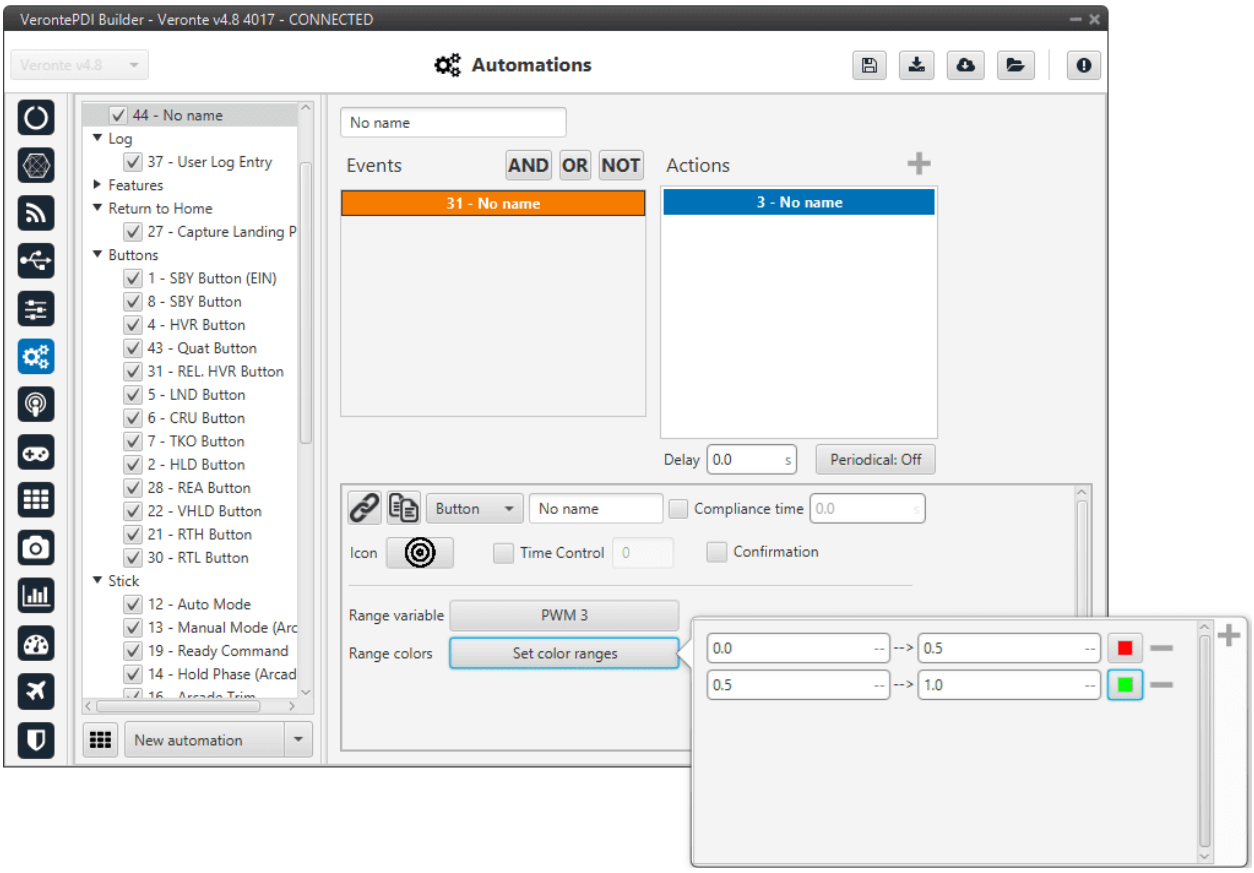


Fig. 167: **Button event**

The following options are available:

- **Icon**: The user can select the most appropriate icon for the event from a list of icons provided by the software.
- **Time Control**: This functionality estabilshes the **time the button must be pressed** to trigger an action.

> **Note:** The time must be specified in **seconds**.

- **Confirmation**: A pop-up window asking for confirmation will be displayed after pushing the button, so it is a safety measure.
- **Range variable** and **range colors** options are used to make the button change its color according to the value of a variable. To do that, select a variable and then indicate as many points as desired, each one with its corresponding value and color.

> **Warning:** For the buttons to be colored, it is necessary that the chosen variables have been added to the **mandatory telemetry**, adding it to the complementary telemetry is not sufficient.

Note:

- If a **button event** triggers an action that consists of a change to a determined phase, the button will be displayed in the Veronte Panel of **Veronte Ops** with the name of that phase on it.
  - To create the button for changing to a determinated phase, it is only needed to link **the button event** to the corresponding '*Phase*' **action**.
- If the **button event** is linked to a different action (servo movement, variable, etc.), it can be displayed both in the **Veronte Panel** and as a independent Action button input.

2.6.2.1.4 Mode

The event is triggered when the aircraft is in one of the modes selected.

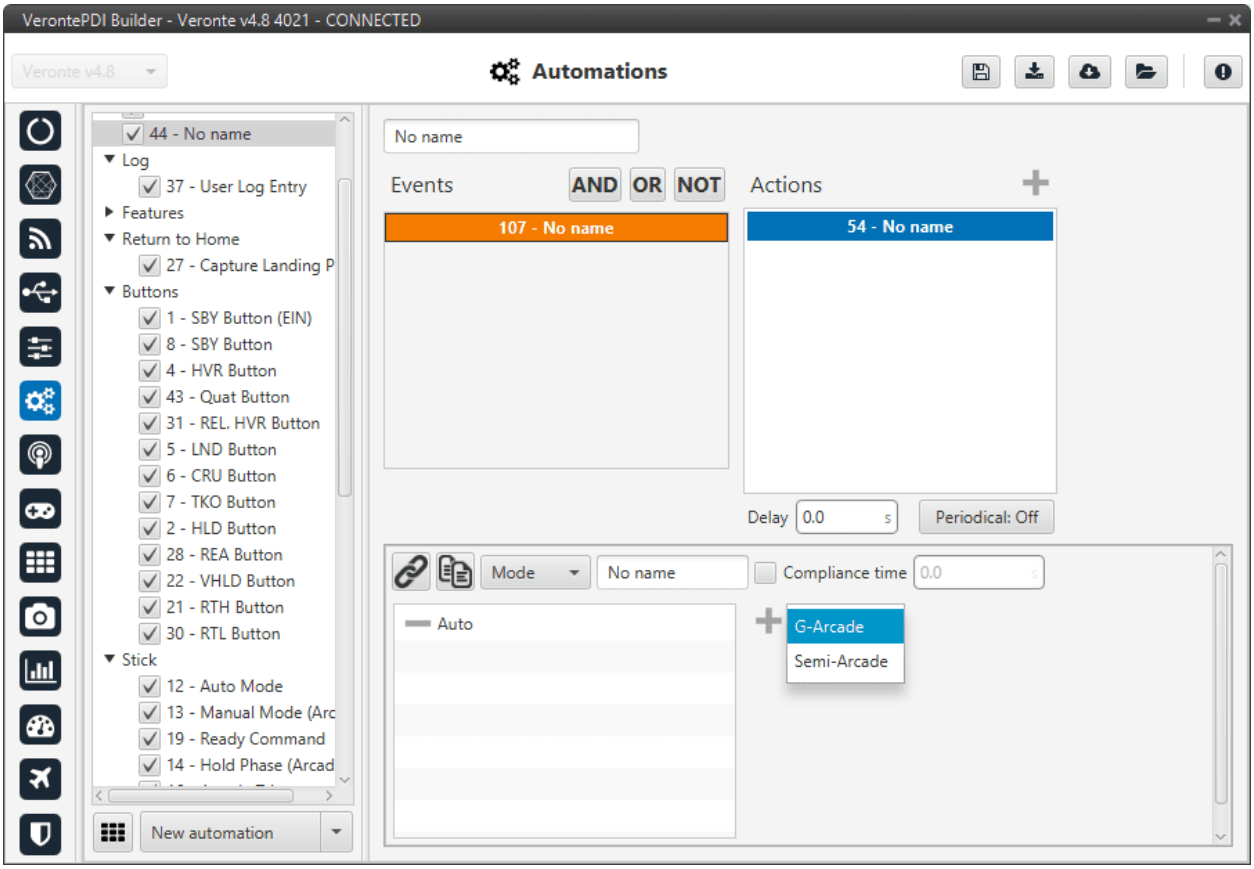


Fig. 168: **Mode event**

These modes have been created previously. See section *Modes*, for more information about creating modes.

The **compliance time** option could be interesting in this type of event.

2.6.2.1.5 Phase

The event is triggered when the aircraft is in the phases selected by clicking on +, being in any of them will trigger the action.

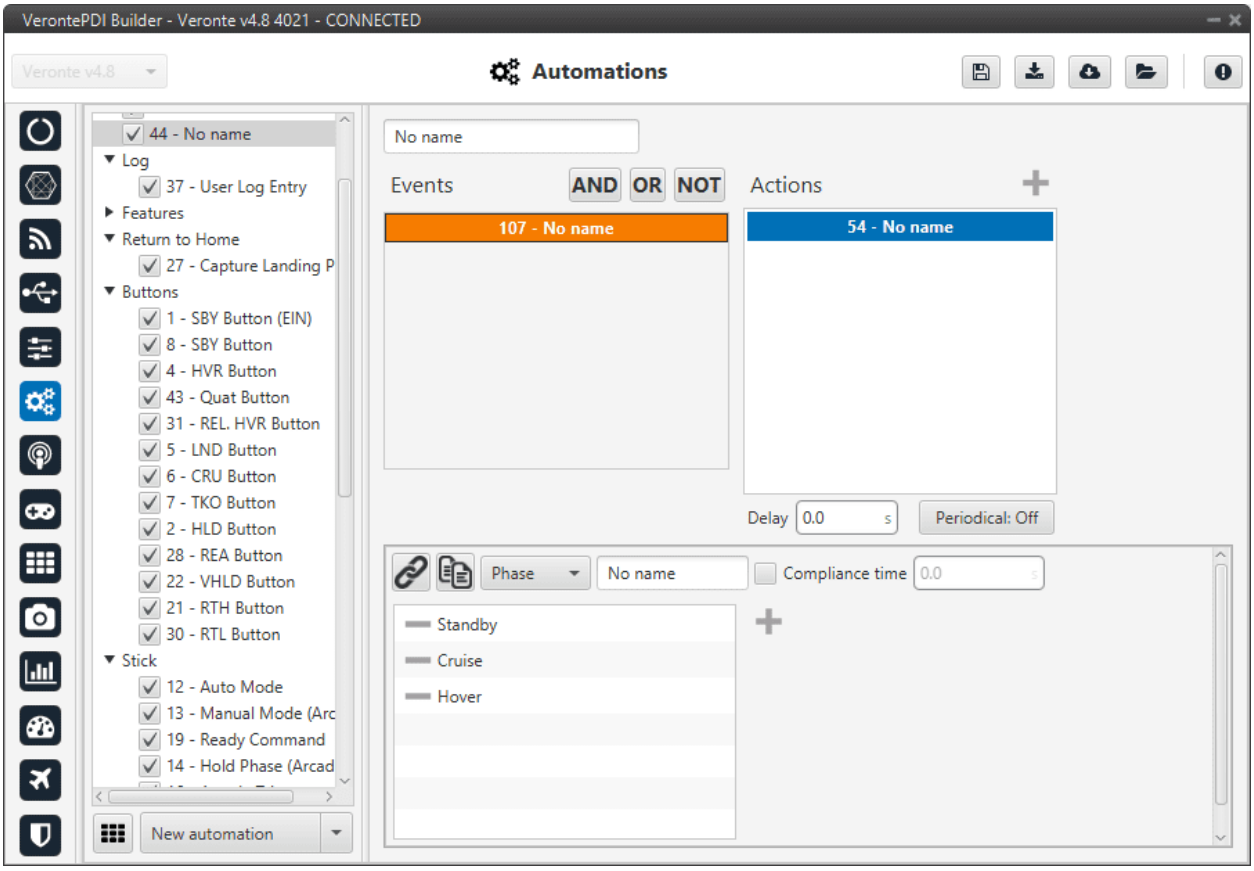


Fig. 169: **Phase event**

These phases have been created previously. See section *Phases*, for more information about creating phases.

2.6.2.1.6 Route

This event is related with the patches and marks defined by the user in the *Operation elements panel* of the **UI menu** and to those created in the mission (in **Veronte Ops**, see more about the creation of marks and patches in the Veronte Ops manual).

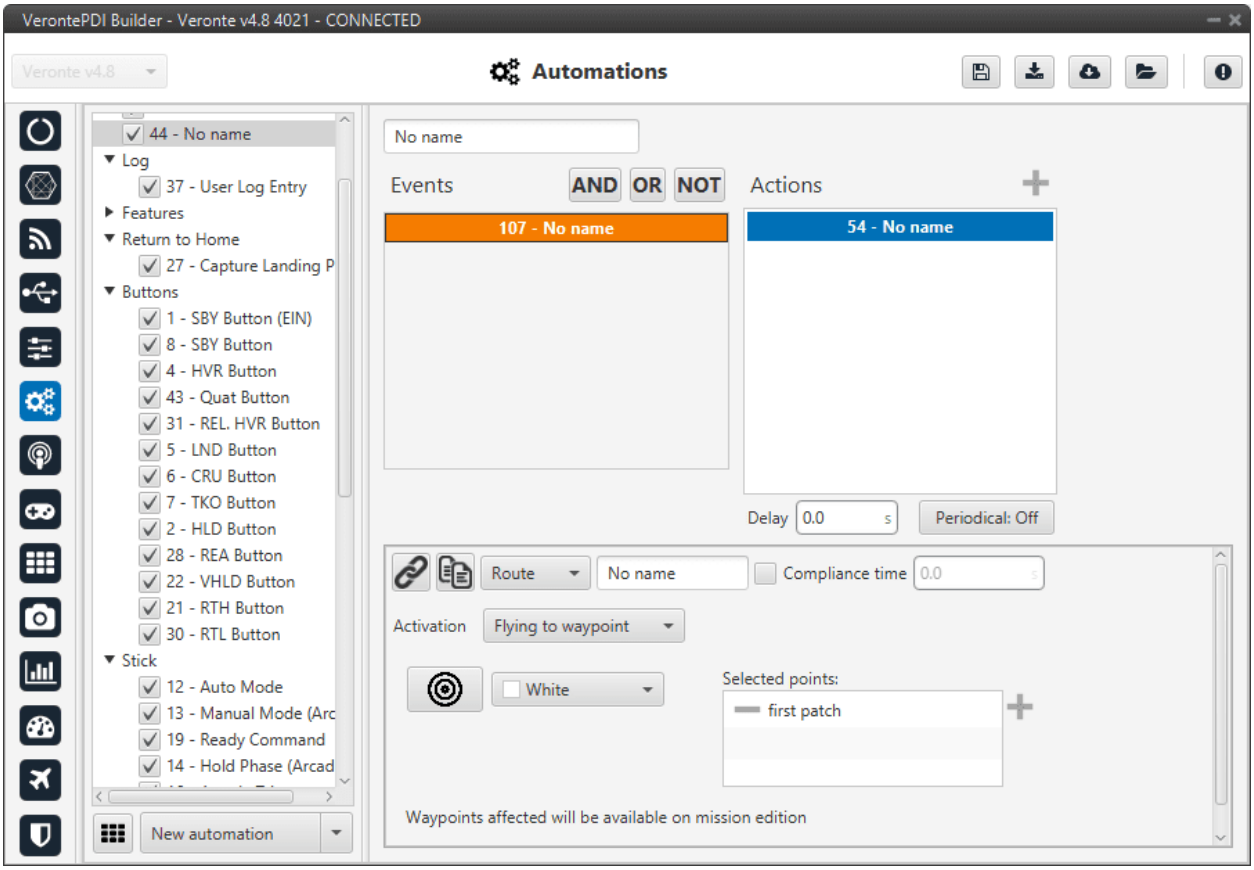


Fig. 170: **Route event**

The following options are available:

- **Activation**: The user can choose between two modes in this event.
  - **Fly to waypoint**: Triggers the action when the platform is flying towards that waypoint (**patch**).
  - **Mark achieved**: Triggers the action when the vehicle has reached the selected mark.
- **Selected marks/points**: To select a mark/waypoint (patch), first define it in the *Operation elements panel* of the **UI menu**.
- **Icon** and **color**: It is possible to change the appearance of the waypoint, selecting an icon from the icon list and a color, so the user can identify easily the waypoint linked to that automation.

2.6.2.1.7 Timer

This event will check the status of the timer selected in the menu. That timer should have been previously configured on the action side of another automation (action type *Periodical*).

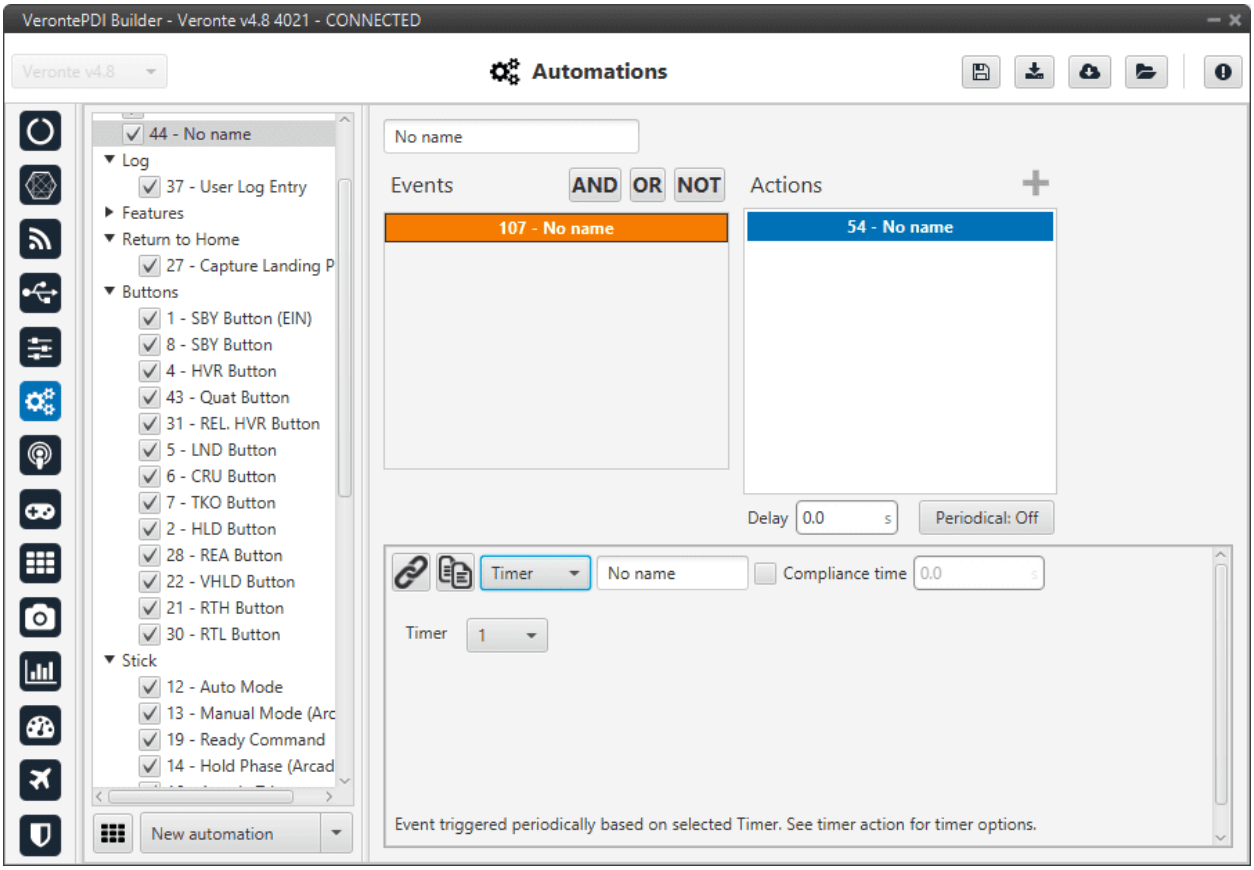


Fig. 171: **Timer event**

In the **Timer** parameter, users must select the number that identifies the timer (previously created with the *periodical action*) that is evaluated in this event.

For example, if it is desired to take a photo 10 seconds after the takeoff, two automations are required:

1. The first automation should have the **event of Phase Take Off**, with the correspondent **Periodical action** that will start a timer that lasts 10 seconds.
2. The second one should have a **Timer event** with the timer previously created and then an **action to take a photo** when the timer event is triggered.

2.6.2.1.8 Variable

This event is triggered when a variable selected is between a range established.

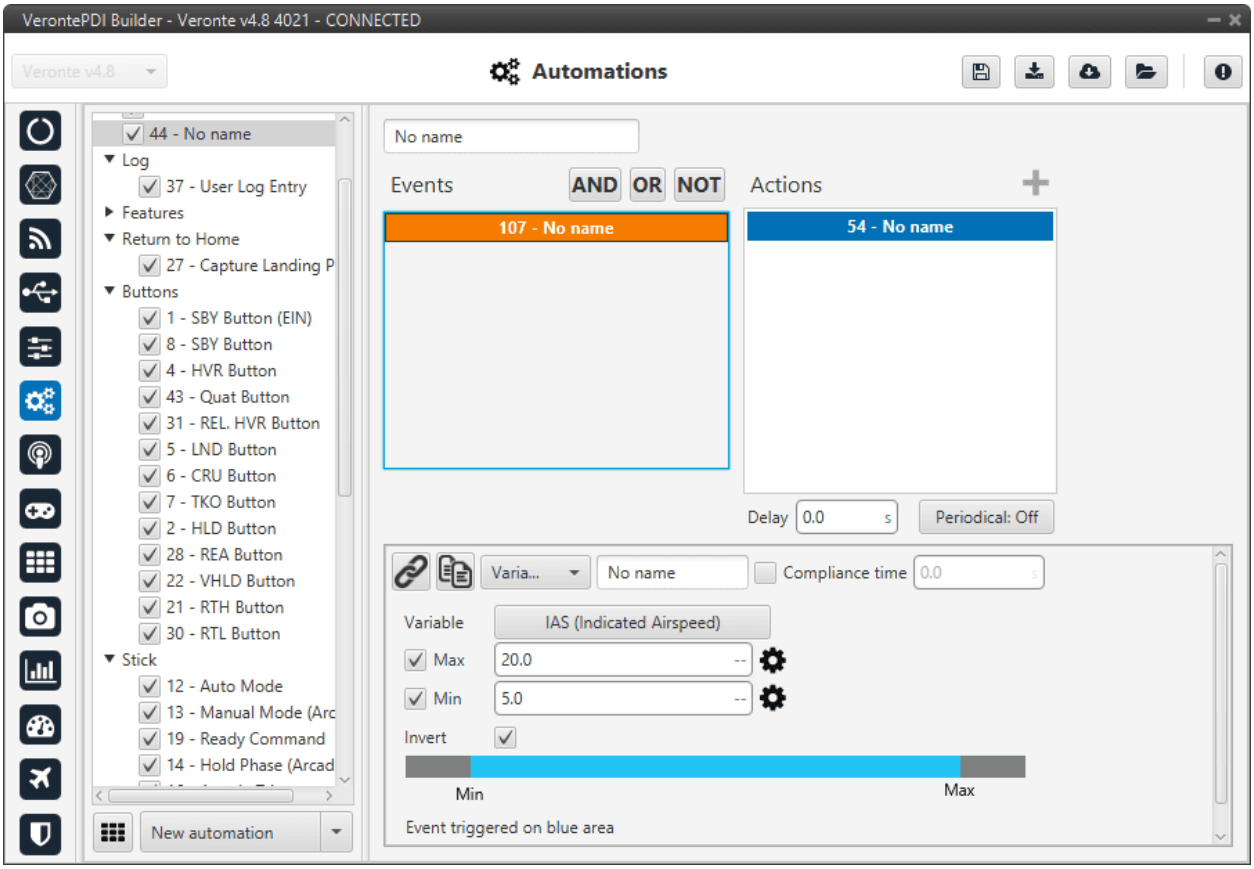


Fig. 172: **Variable event**

- **Variable**: The user can select the variable to be evaluated.
- **Max/Min**: Maximum and minimum values of the threshold are established here. Custom threshold can be established by clicking on the ⚙ icon.
- **Invert range**: This option will change the interval (the blue area will be gray, and the gray one will be blue).

As an example consider the event of the figure. With that parameters, the event is triggered when the IAS is between 5 and 20 meters per second. If the invert range option is unchecked, the event will be triggered when the IAS is lower that 5 m/s or greater than 20 m/s.

2.6.2.2 Actions

An action is a specific task, operation, or set of activities that will be performed when the event (or group of events) has been accomplished. The actions box contains all the created actions.

The user can also rename the action with the name of their choice.

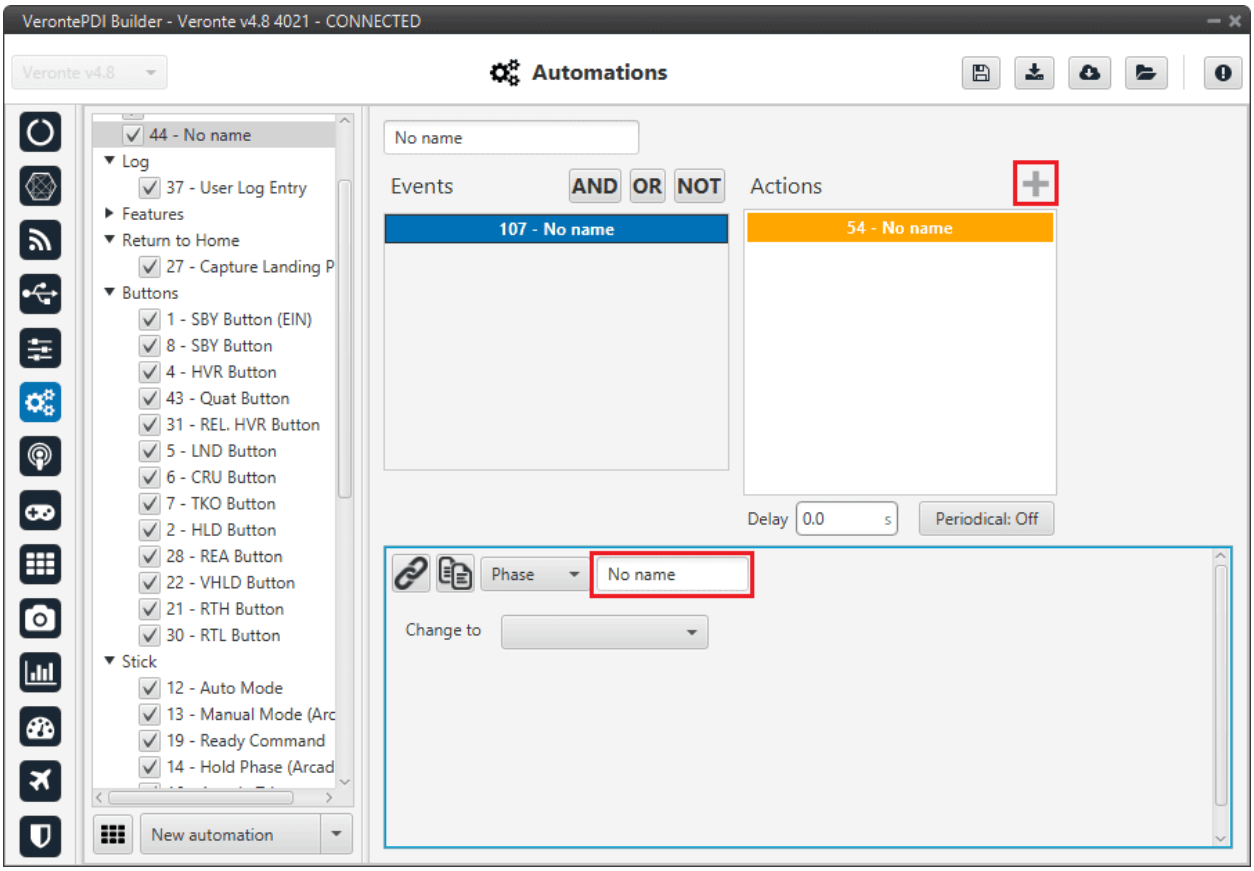


Fig. 173: **Actions menu**

When creating a new action it is possible to choose from one of the **previously created** on the system or to **create** a **new** one.

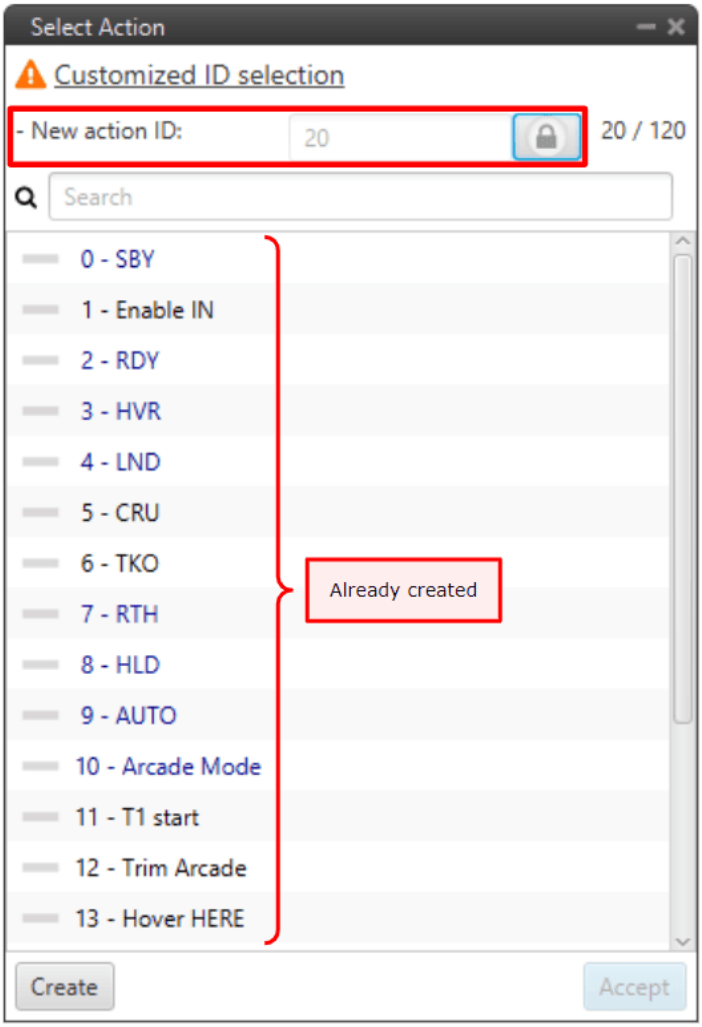


Fig. 174: **New action**

- To add an **action already created**, select the desired event and click **Accept**.
- By default, when a **new action** is created, it is assigned with the consecutive ID.

  However, if the user wants to create a new action with a specific ID, the *New action ID* option can be "unlocked" by holding down  until the process is fully completed (during the unlocking process this icon  is "painted").

  Once unlocked (when it appears as ), the user can assign the desired ID to the new action.

  Finally, click **Create** and the new action will be added.

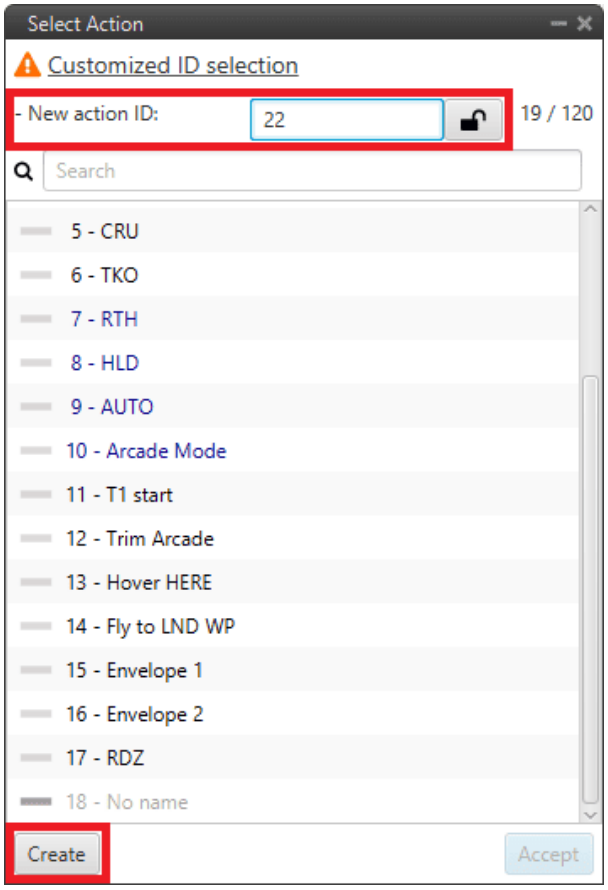


Fig. 175: **Automations menu - New action ID**

Caution: When the user enters an ID that is already assigned to an action, the following confirmation message will appear:

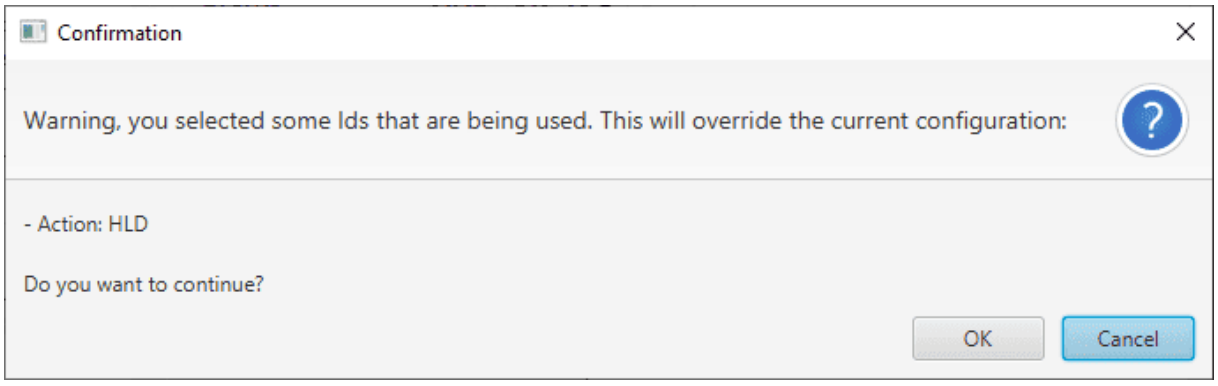


Fig. 176: **Automations menu - New action confirmation message**

Then, if the user presses 'OK', the existing action will be removed and a new action with the entered ID will be added.

The different types of actions that can be created are presented below.

2.6.2.2.1 Atmosphere calibration

This action allows the atmosphere calibration in the same way as shown in the **Operational panel of Veronte Ops** (for more information about atmosphere calibration, see Veronte Ops manual).

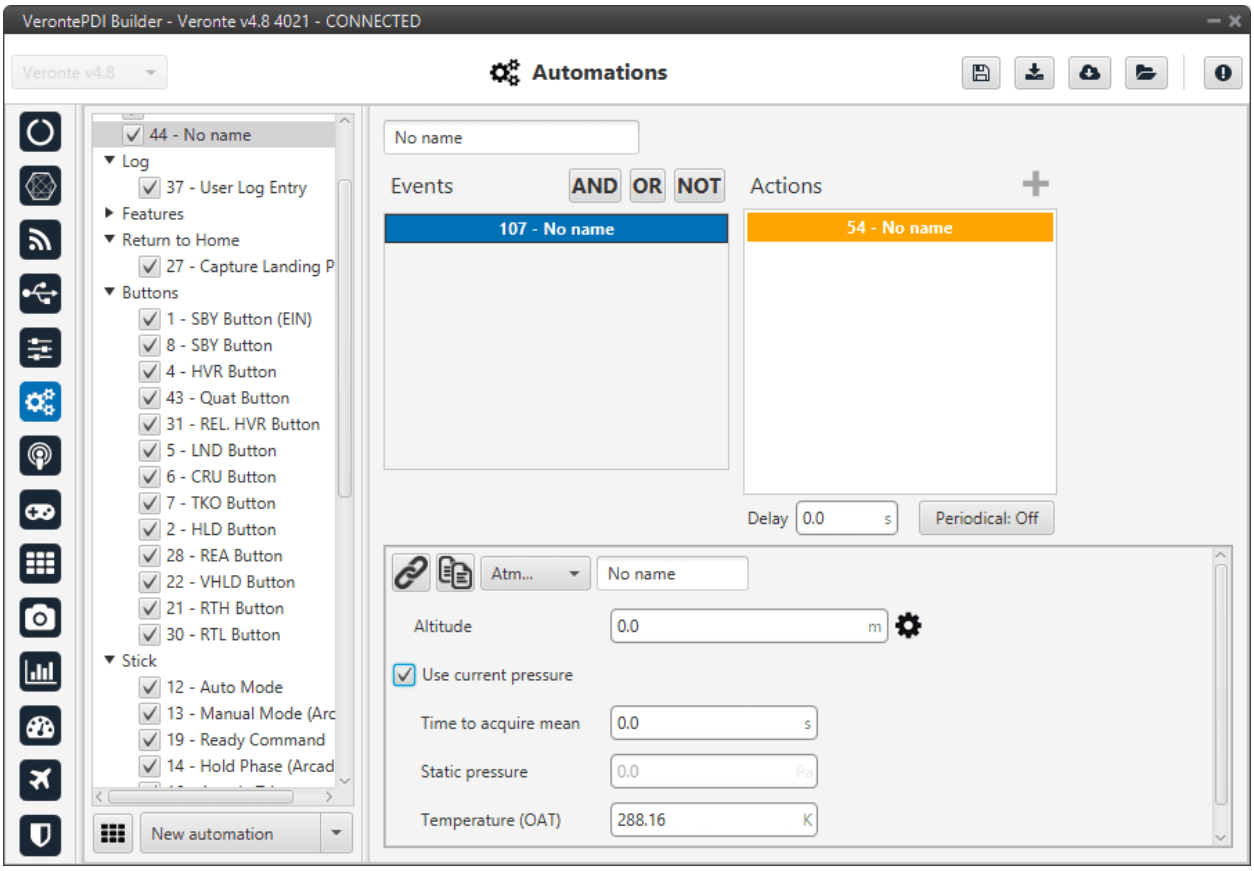


Fig. 177: **Atmosphere calibration action**

The following options can be configured:

- **Altitude**: Actual MSL altitude. The user must choose between entering this value manually or selecting a system variable.
- **User current pressure**: By enabling it, the static pressure will be read from the static pressure sensor during the specified time (**Time to acquire mean**).
- **Static pressure**: If the above option is not enabled, the actual static pressure should be specified manually.
- **Temperature (OAT)**: Outside air temperature.

2.6.2.2.2 Change active sensor

This option allows changing the current selected and default sensors used as IMU (accelerometer and gyroscope) and dynamic pressure.

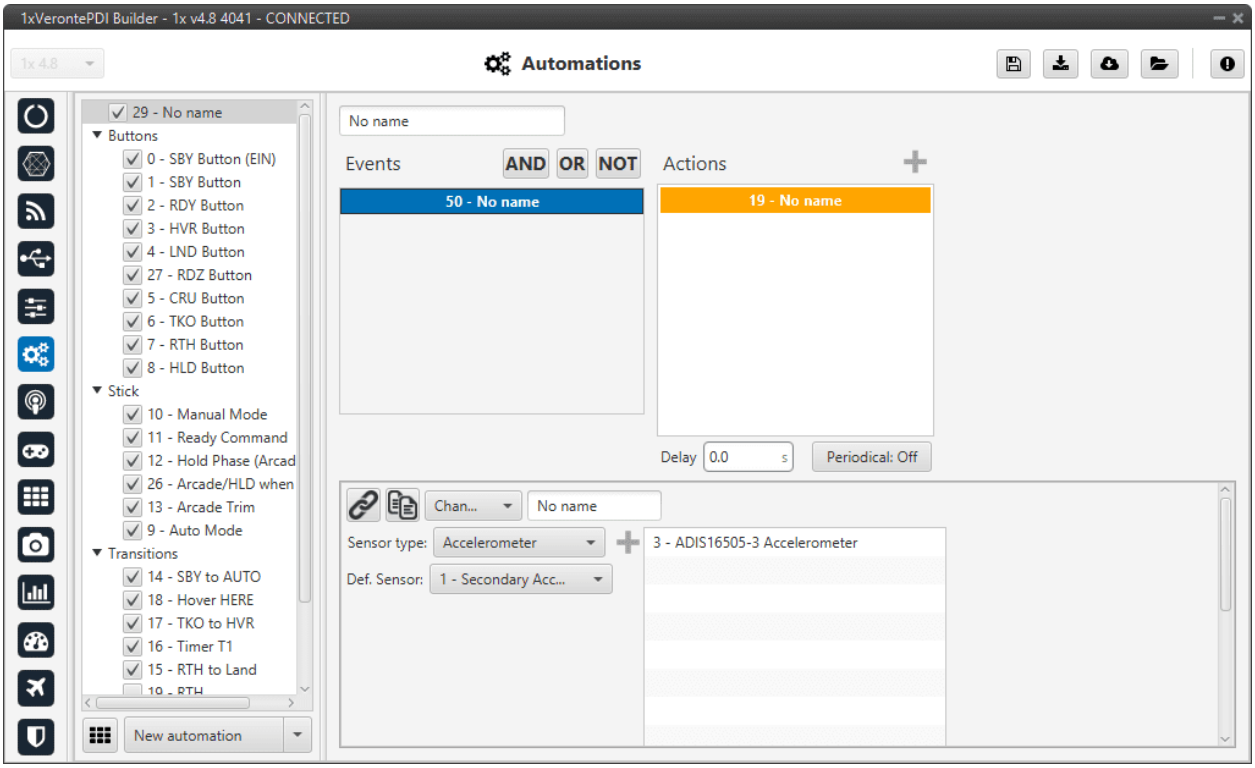


Fig. 178: **Change active sensor action**

- **Sensor type**: Select the type of sensor to be changed with this action: accelerometer, gyroscope or dynamic pressure.
- **Def. Sensor**: A default sensor must be chosen. If all selected sensors fail, the measurement value will be that of the default sensor.
- : By clicking here, users can add the selected sensors of their choice.

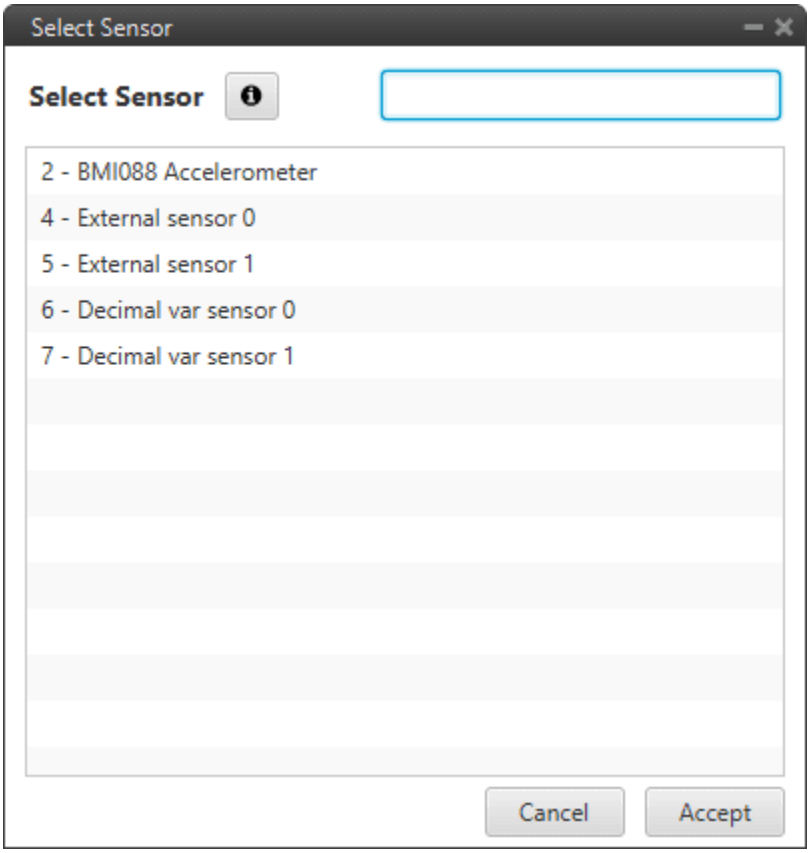


Fig. 179: **Change active sensor action - Select sensor**

Note: The behavior of the default and selected sensors is the same as described in the *Sensors* section of this manual.

2.6.2.2.3 Command block

This action allows the user to configure a gimbal or to trim a radio controller.

Note: This action is disabled by default when Autopilot 1x is started. To activate it, the user have to create a gimbal block or an arctrim block (see more information about it in the *Block Programs* section of this manual).

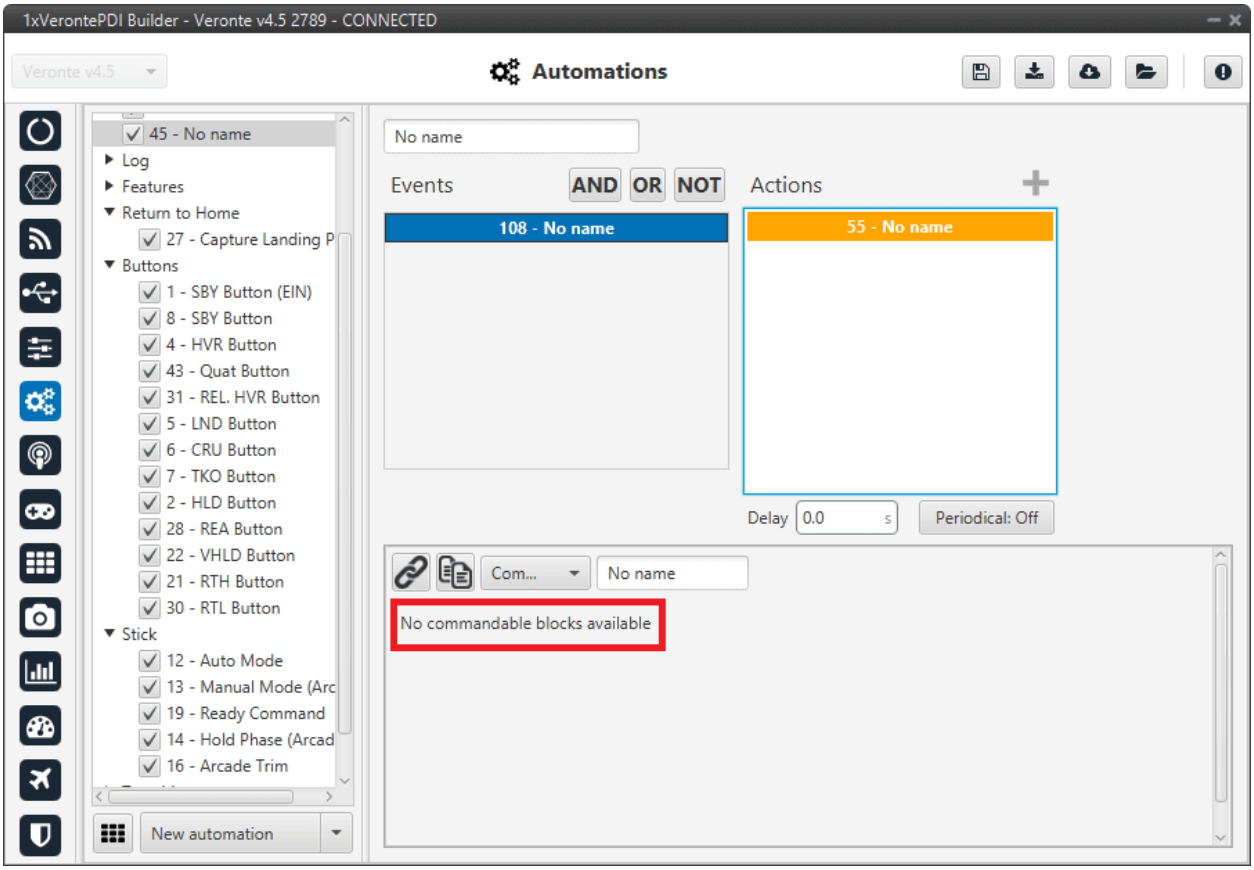


Fig. 180: **Command block action**

Gimbal

When this action is triggered, the gimbal control is enabled. There are several control modes that are explained below.

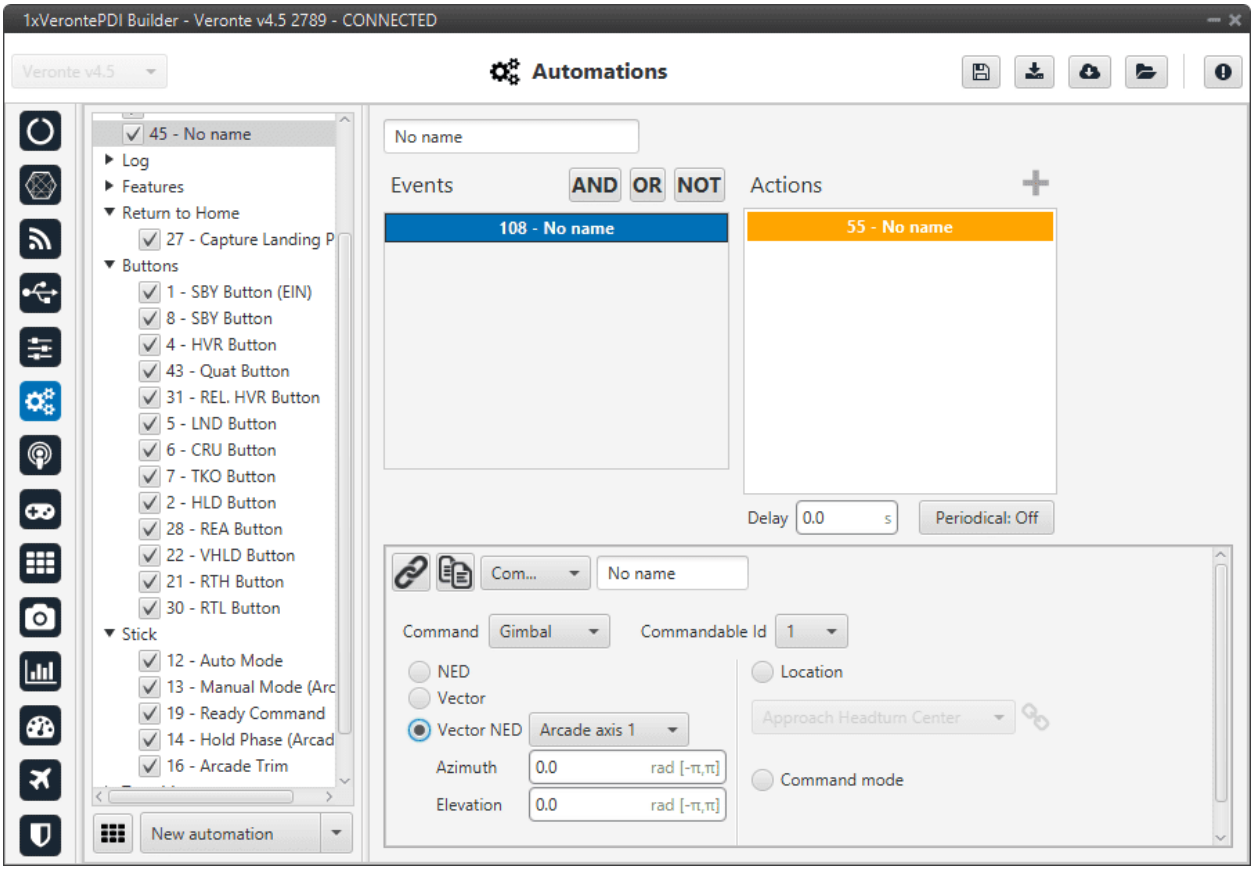


Fig. 181: **Command block action - Gimbal**

- **Commandable Id**: Id of the commandable Gimbal block (users can look up this Id in the block to be commanded, in the *Gimbal - Devices blocks* of the **Block Programs menu**).
- **NED**: Control using NED axis, defining the initial position through azimuth and elevation.
- **Vector**: This control uses aircraft body axis. The initial position is defined through roll and tilt.
- **Vector NED**: In this case the axis should have been defined in *Arcade Axis* of the **Control menu**.
- **Location**: The gimbal will point towards the projection on the ground at the specified point.
- **Command mode**: Gimbal control is done externally, e.g. via VCP commands.

ArcTrim

This action trims the radio controller, i.e sets as zero the current sticks positions.

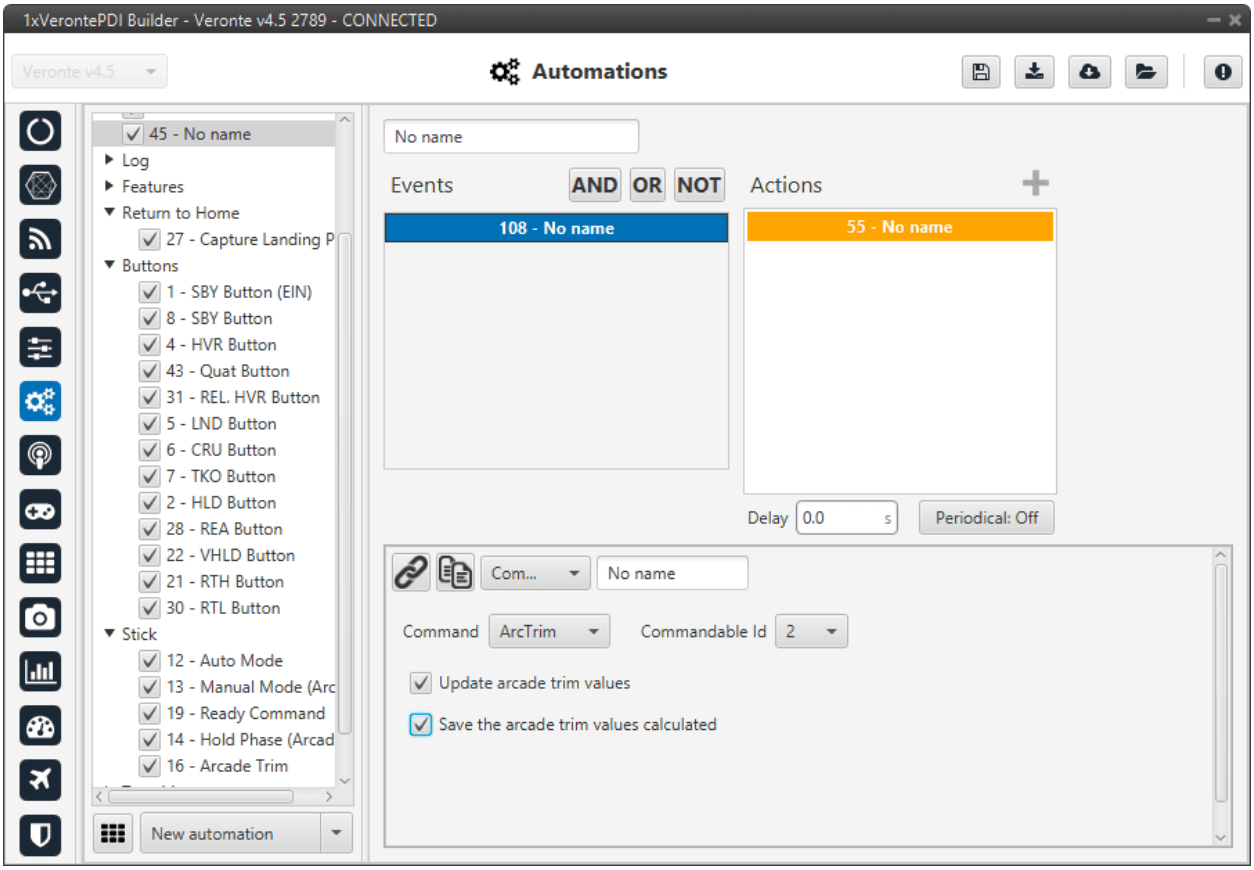


Fig. 182: **Command block action - ArcTrim**

- **Commandable Id**: Id of the commandable ArcTrim block (users can look up this Id in the block to be commanded, in *Arc Trim - Servos blocks* of the **Block Programs menu**).
- **Update the arcade trim values**: If this option si enabled, the stick is trimmed but not saved in the configuration. This means that **if Autopilot 1x is restarted the trimming is lost**.
- **Save the arcade trim values calculated**: Trim values are **stored** for future flights.

2.6.2.2.4 Custom CAN TX

When this action is triggered, a previously configured CAN message is sent through the CAN bus. The message has to be configured in *CAN Custom messages panel* of the **Input/Output menu**.

> **Warning:** As this automation is used to send a single message on demand, in its configuration in **Custom Messages**, the user has to set its **period to -1**. This way, this **message will only be sent when this action is triggered**.

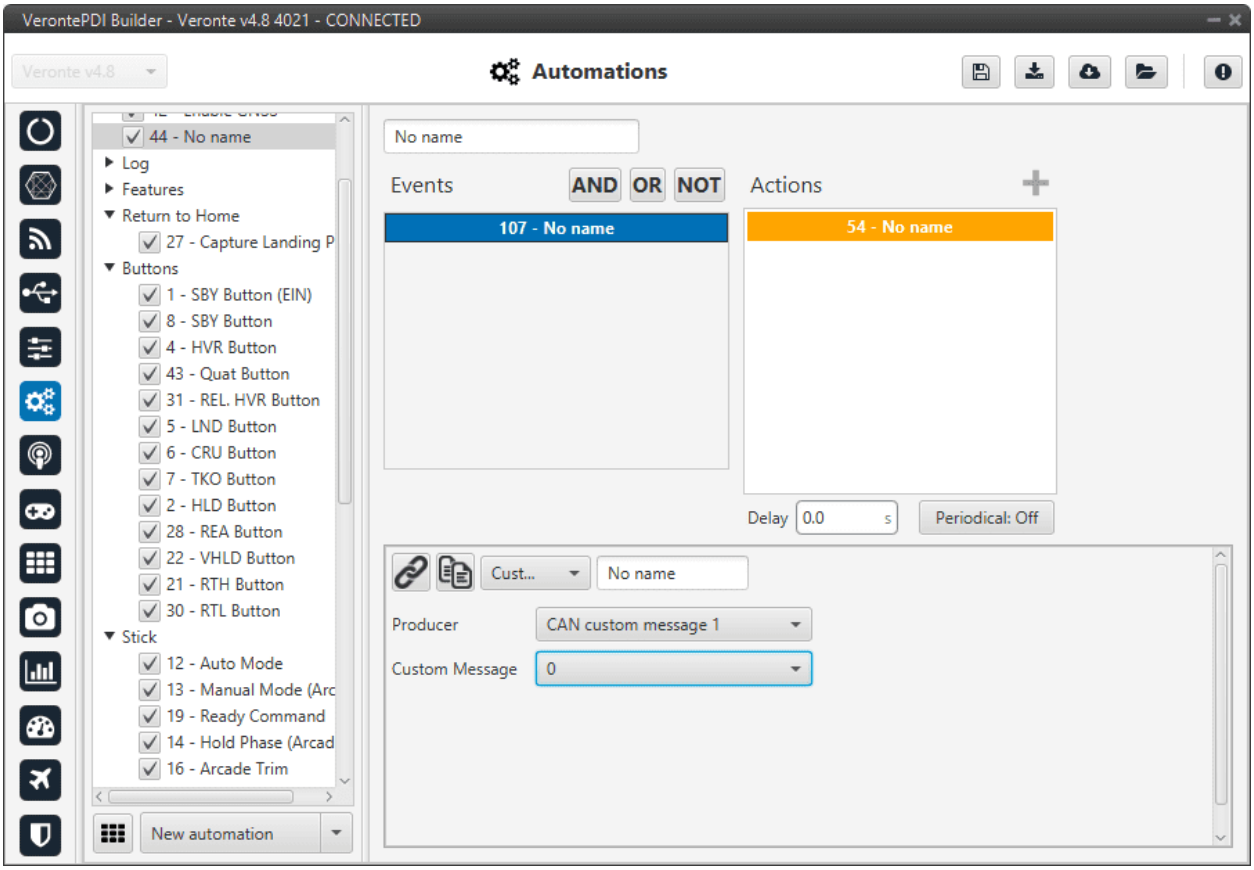


Fig. 183: **Custom CAN TX action**

The two parameters to configure in this action are:

- **Producer**: The user has to specify where the custom message is located: CAN custom message 0, 1 or 2.
- **Custom Message**: The number of the custom messages that will be sent.

2.6.2.2.5 Custom Serial TX

When this action is triggered, a previously configured serial message is sent through the serial port (RS232 or RS485). The message has to be configured in *Serial custom messages panel* of the **Input/Output menu**.

> **Warning:** As this automation is used to send a single message on demand, in its configuration in **Custom Messages**, the user has to set its **period to -1**. This way, this **message will only be sent when this action is triggered**.

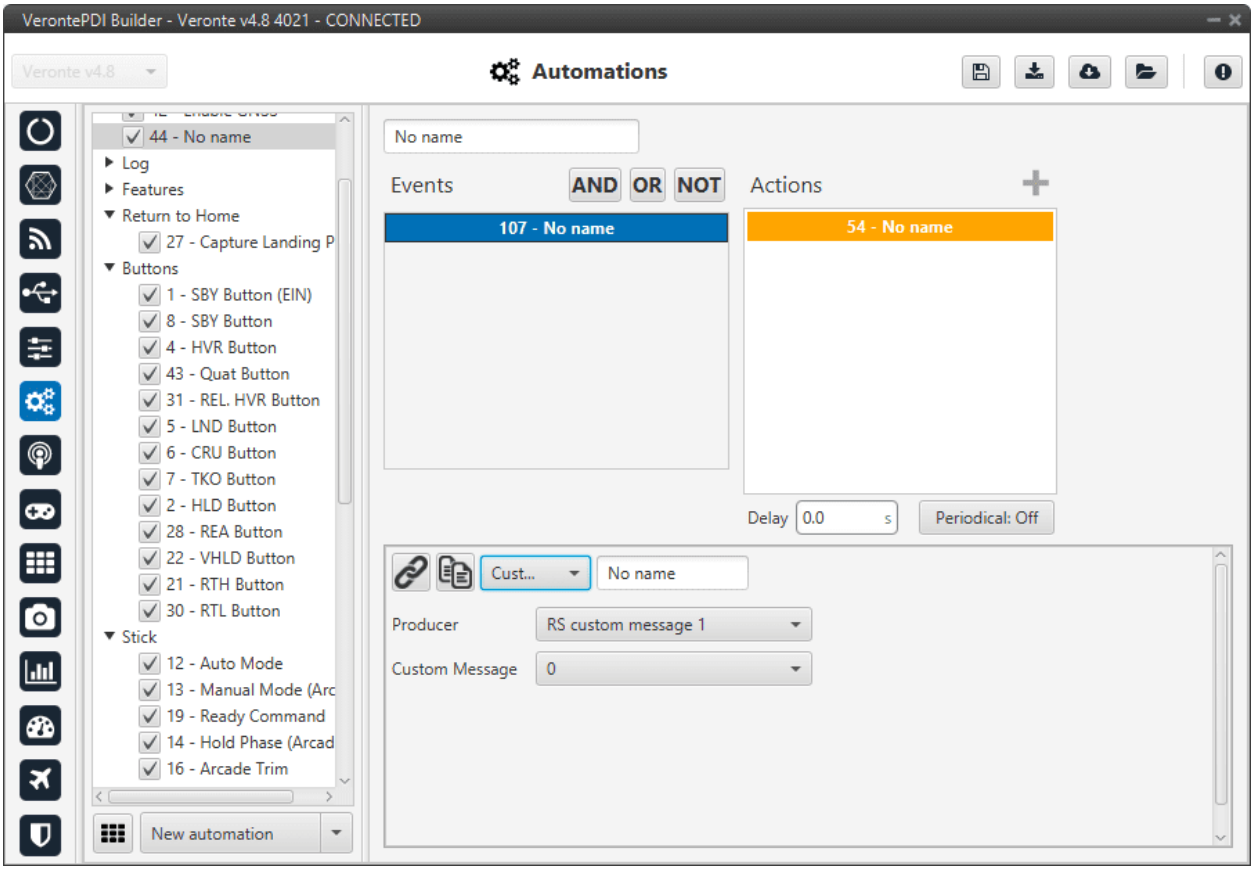


Fig. 184: **Custom Serial TX action**

The two parameters to configure in this action are

- **Producer**: The user has to specify where the custom message is located: RS custom message 0, 1 or 2.
- **Custom Message**: The number of the custom messages that will be sent.

2.6.2.2.6 DEM calibration

This option allows the calibration of the digital elevation model by setting the **actual AGL value** in the same way as shown in the **Operational panel of Veronte Ops** (for more information about DEM calibration click Veronte Ops manual).

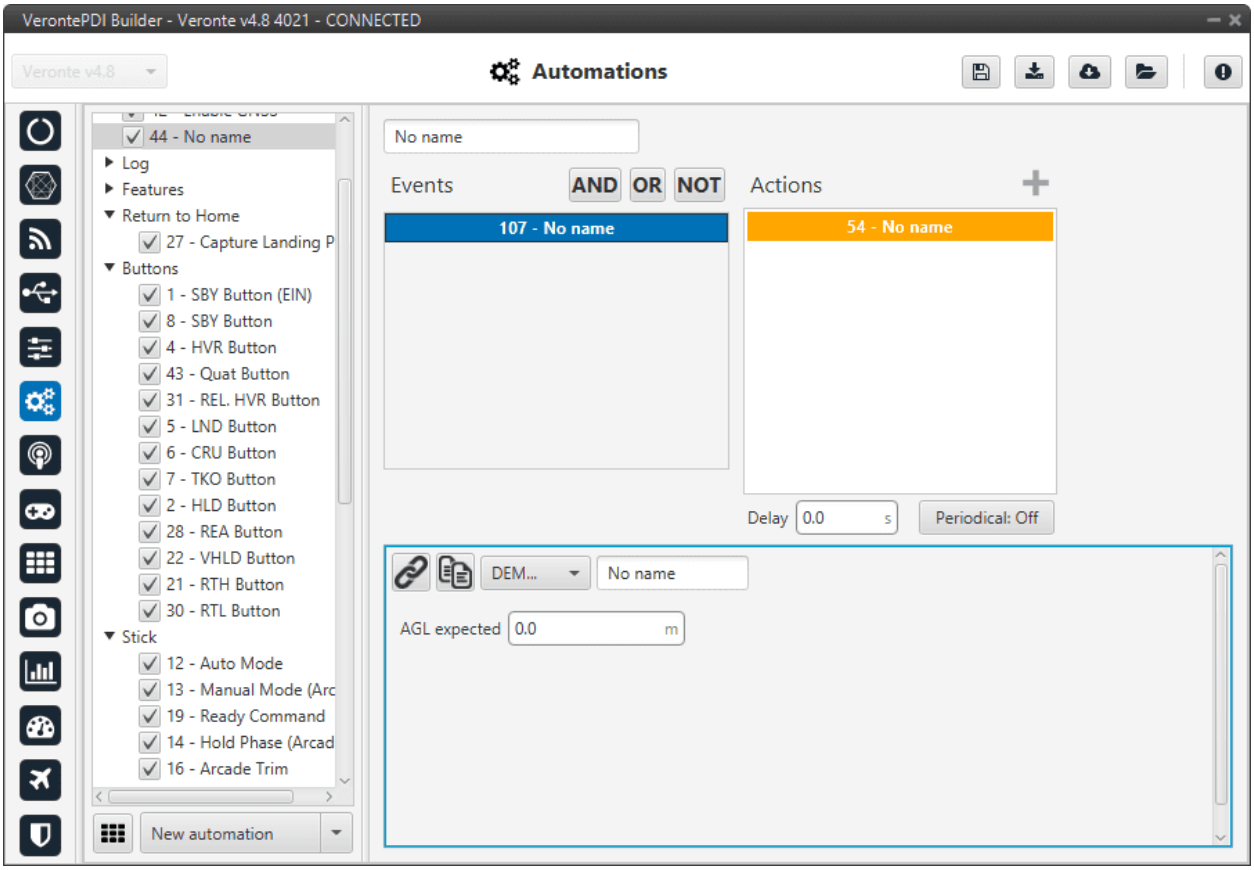


Fig. 185: **DEM calibration action**

2.6.2.2.7 Enable/Disable Wind Estimation

This action allows the user to enable or disable the wind estimation.

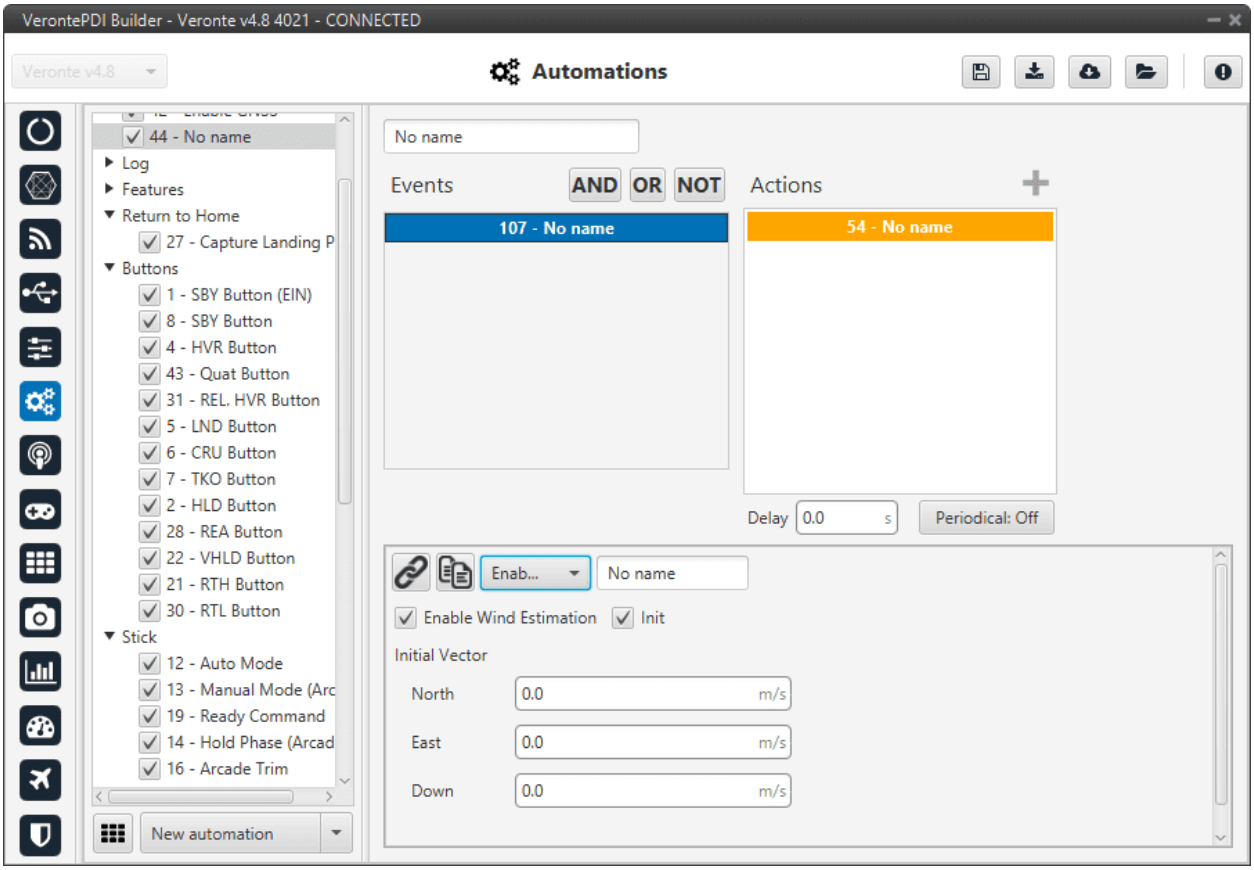


Fig. 186: **Enable/Disable Wind Estimation action**

The following parameters can be configured:

- **Enable Wind Estimation**: Enabled/Disabled.
- **Init**: By enabling it, an initial wind vector can be set to a faster convergence of the estimation.
  - **North, East, Down**: Inital wind vector.

2.6.2.2.8 FTS-Activation

This action activate the flight termination system (FTS) bit.

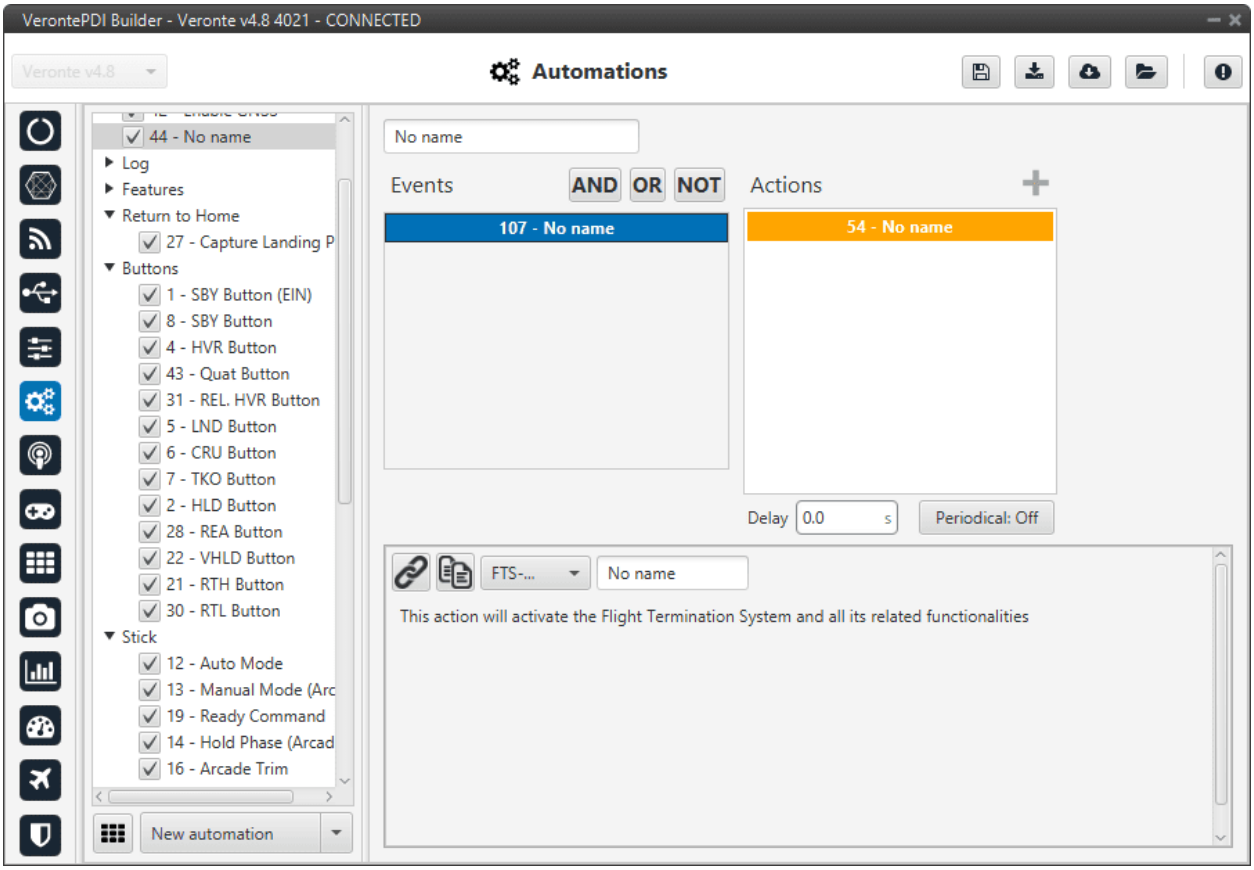


Fig. 187: **FTS-Activation action**

In an Autopilot 4x, when two or more autopilots activate their FTS the arbiter can activate a safe system such as a parachute.

2.6.2.2.9 Feature

When this action is triggered, a position is stored in the desired variable. This position can be absolute or relative (in the figure below the current position of the aircraft would be saved):

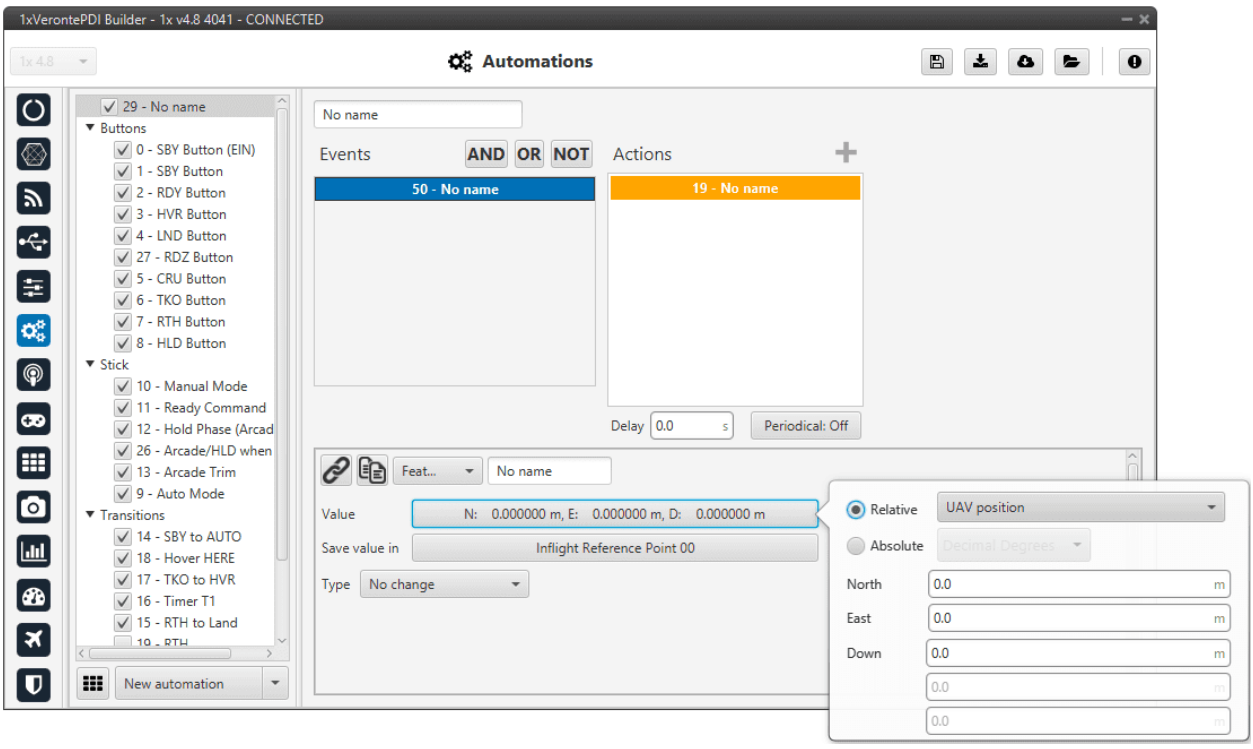


Fig. 188: **Feature action**

The following options should be configured:

- **Value**: Specified the position to be stored, this position can be **absolute** or **relative**.
  - **Absolute**: The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress º ” ‘. They are indicated through the latitude, longitude and altitude (being possible to define this last one with respect to the ellipsoid, WGS84, to the sea level, MSL or to the ground, AGL).
  - **Relative**: In this case, the position of the point is relative to another point. That point could be any platform fitted with an Autopilot 1x.
- **Save value in**: The position has to be saved in an ‘Inflight Reference Point’.
- **Type**: There are 2 types of features:
  - **Fixed**: Once the point has been generated it remains fixed.
  - **No change**: If the **point** has been created **relative**, it remains relative all the time.

> **Note:** This option only appears when the position has been previously defined as **Relative**.

This action is very useful for storing the take-off point for later landing at the same place.

2.6.2.2.10 Format SD

Warning: This action will have irreversible effects on your Autopilot 1x. Formatting the SD card will delete important and mandatory files for the correct functioning of Autopilot 1x.

In order to recover a formatted, please contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).

This action will format the SD card, deleting the configuration and flight logs from it.

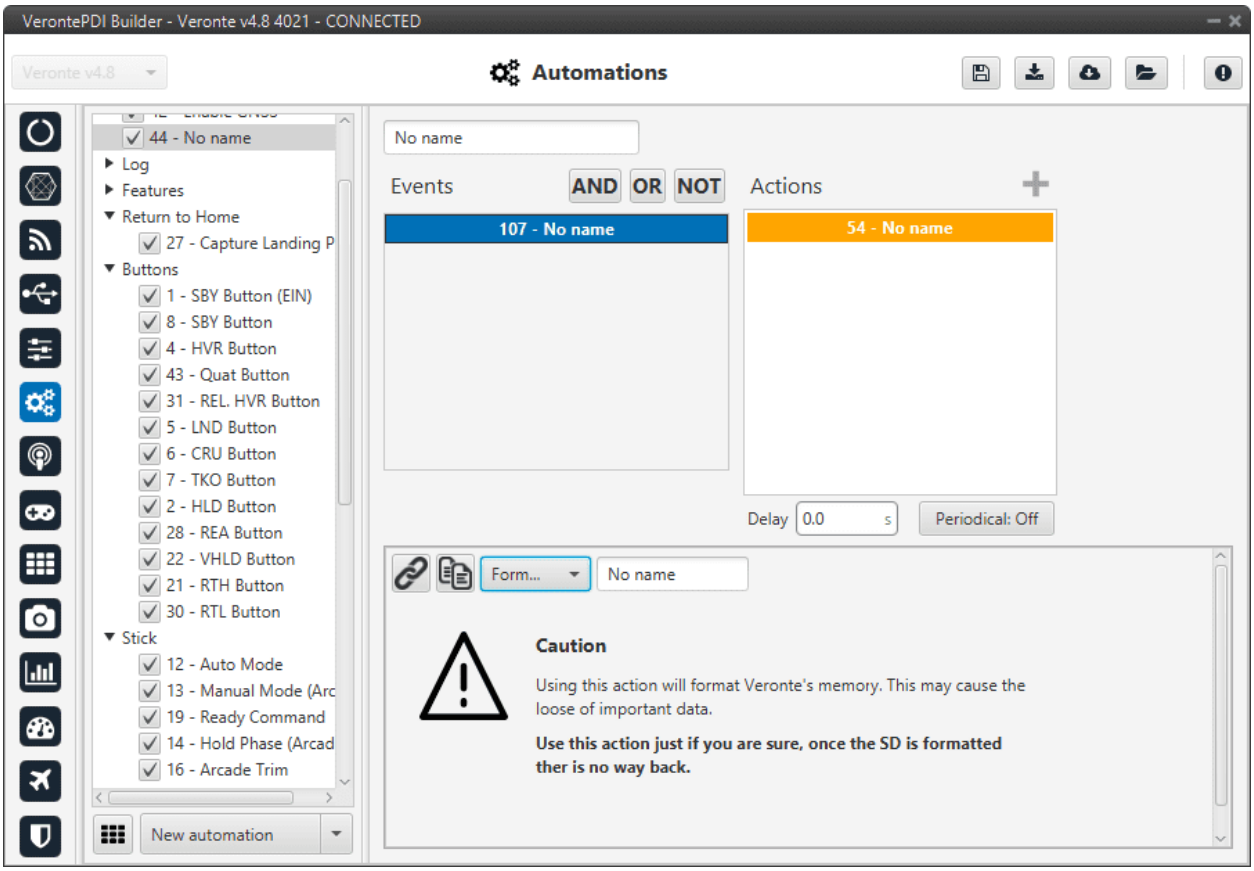


Fig. 189: **Format SD action**

2.6.2.2.11 Go to

This action is used to make the aircraft go to a patch created by the user with the mission toolbar of **Veronte Ops**. For more information about the mission toolbar, take a look at the Veronte Ops manual.

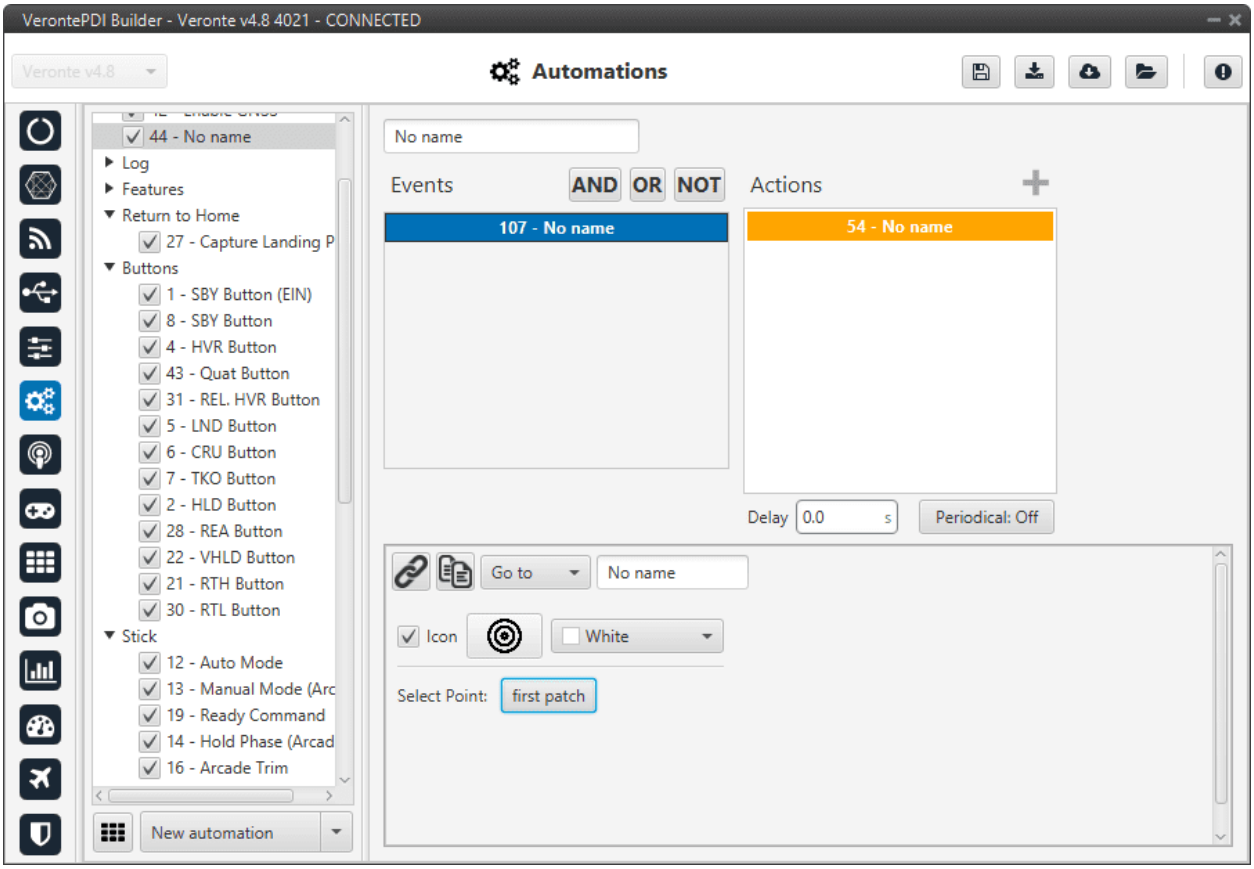


Fig. 190: **Go to action**

In this action two parameters can be configured:

- **Select Point**: To select point (**patch**), first define it in the *Operation elements panel* of the **UI menu**.
- **Icon** and **color**: It is possible to change the appearance of the point, selecting an icon from the icon list and a color, so the user can identify easily the point linked to that automation.

Once the action is triggered, the vehicle will go to that patch. If the patch is on a route, the vehicle will follow the selected patch and then it will continue the route going to its adjacent.

2.6.2.2.12 Mode

The flight mode is changed to the one specified in this option.

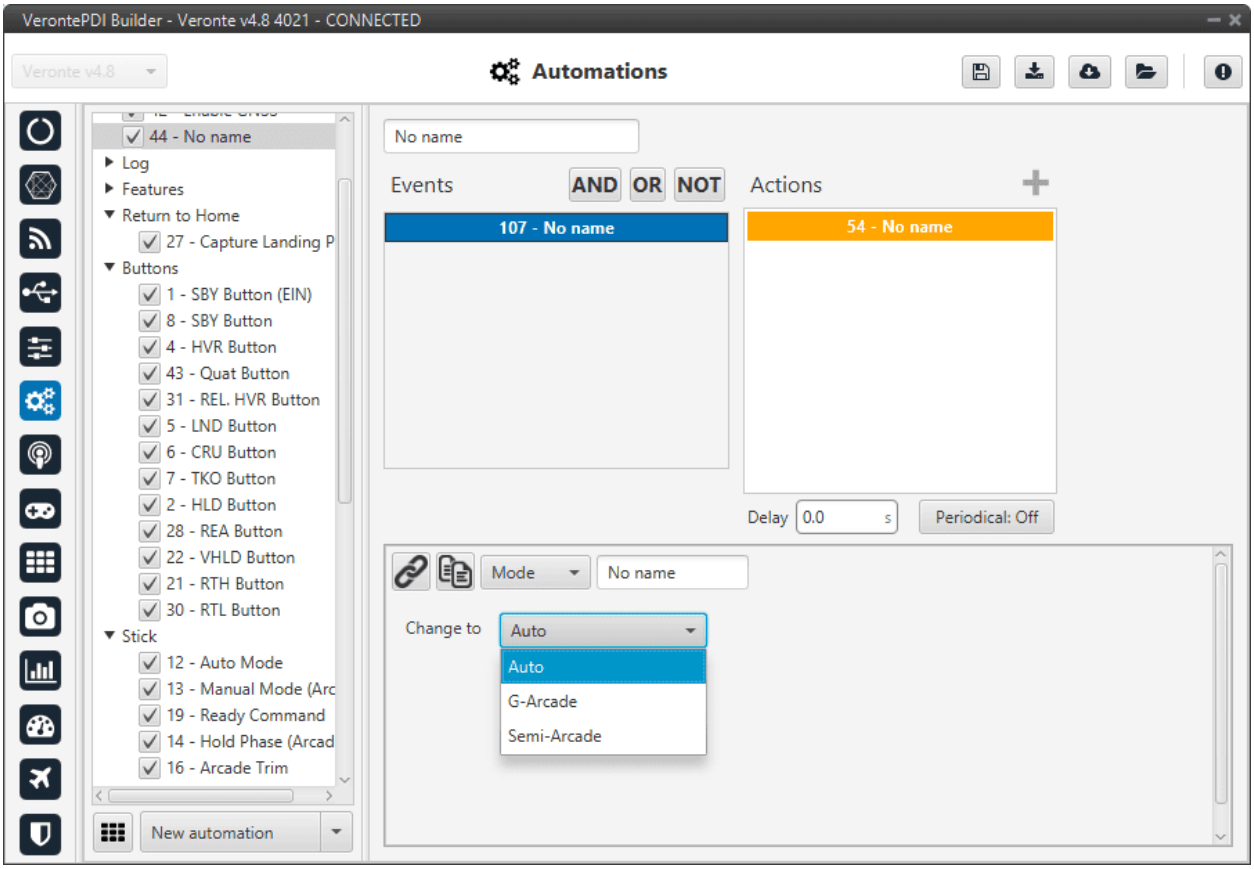


Fig. 191: **Mode action**

These modes have been created previously. See *Modes - Control* section, for more information about creating modes.

2.6.2.2.13 Navigation

This action is used to change the navigation mode used by the aircraft.

By default, the UAV uses an Internal sensor fusion algorithm, but for example, if the GPS fails, the Autopilot 1x switches to **inertial navigation**. Since in this type of navigation, the estimation of position and velocity diverges over time, if that happens, it is advisable to switch to another type of navigation (External).

Note: This **behavior** is not specific to Veronte Autopilot 1x, it is **common to all inertial navigations**.

The navigation without GPS will make the aircraft fly stable but it will not be possible to command a path to follow during that time, so this action can be used as a safety mode to avoid a malfunction of the system when the GPS signal is lost.

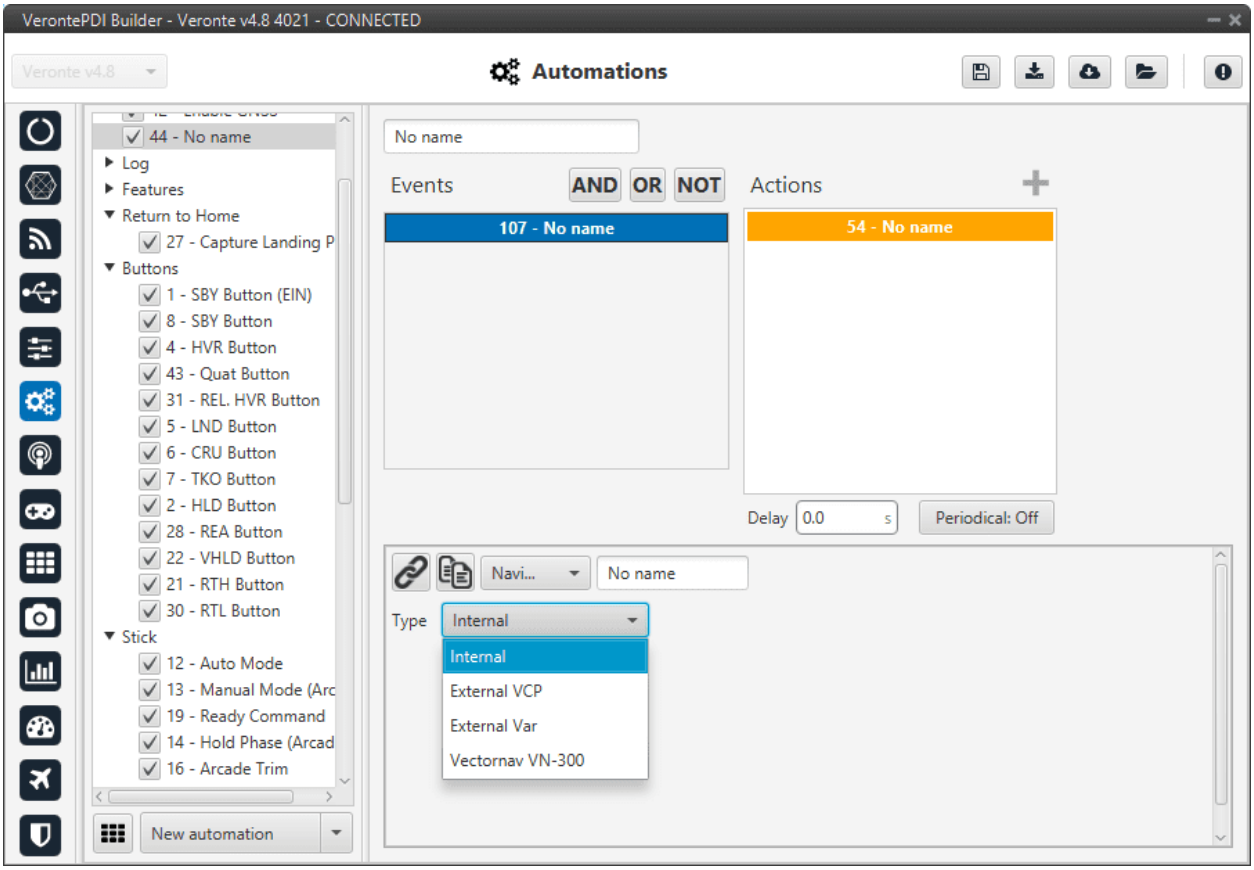


Fig. 192: **Navigation action**

The options available are:

- **Internal**: Uses internal data for navigation. Data (position, attitude, etc.) is processed into 1x unit from sensor measures.
- **External VCP**: Uses external data for navigation. Data (position, attitude, etc.) is provided by Veronte Communication Protocol (VCP).
- **External Var**: Uses external data for navigation.
  - It takes directly the **attitude, velocity and acceleration data** of the following real variables from the memory:
    - **ID 259**: External yaw
    - **ID 260**: External pitch
    - **ID 261**: External roll
    - **ID 262**: External roll rate
    - **ID 263**: External pitch rate
    - **ID 264**: External yaw rate
    - **ID 265**: External velocitiy north
    - **ID 266**: External velocity east
    - **ID 267**: External velocity down

    - **ID 268**: External acceleration x body axis
    - **ID 269**: External acceleration y body axis
    - **ID 270**: External acceleration z body axis
    - **ID 271**: External GPS Time of Week
  - **Position data** is read from the **Moving Feature 00**.
- **Vectornav VN-300**: Uses external data for navigation. Data (position, attitude, etc.) is provided by Vectornav VN-300. For more information, see the *Vectornav VN-300 - Integration examples* section of this manual.

2.6.2.2.14 Obstacle avoidance

This action enables avoidance of any obstacle previously created in **Veronte Ops** (for more information, see Veronte Ops manual).

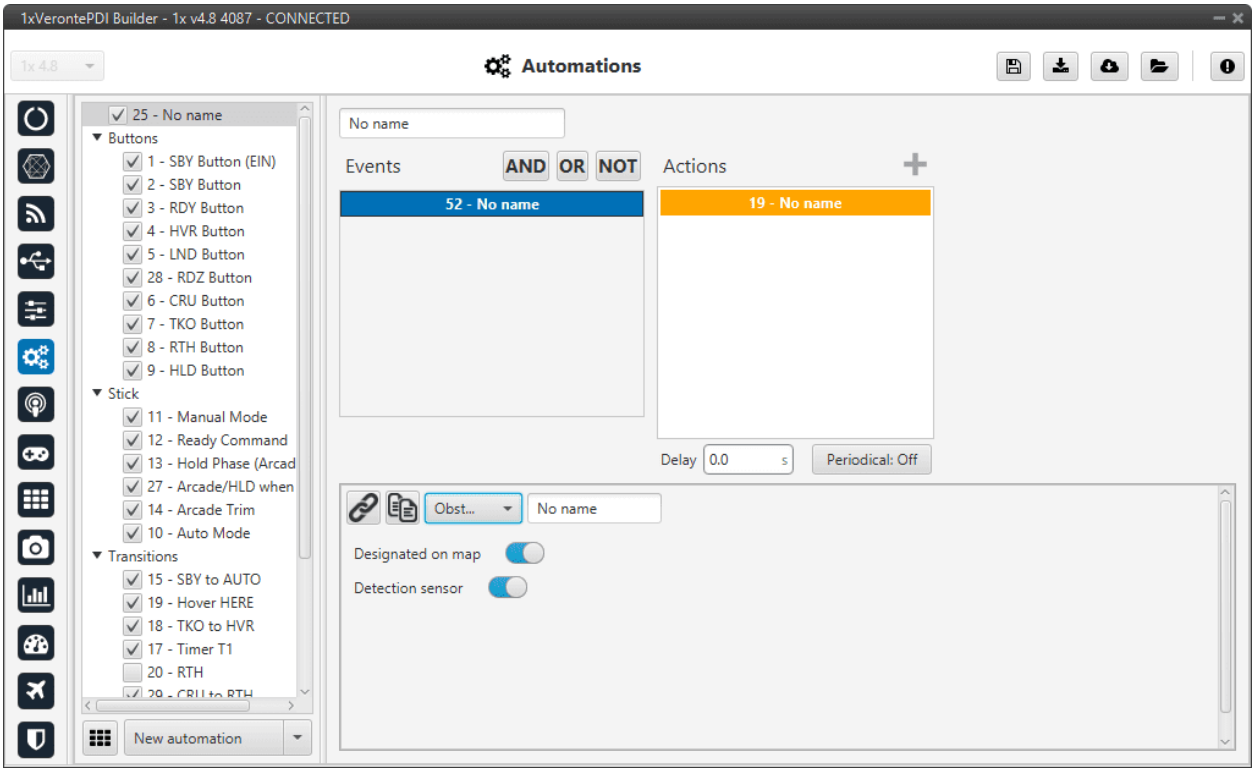


Fig. 193: **Obstacle avoidance action**

2 types of obstacles can be enabled:

- **Designated on map**: By activating this option Veronte Autopilot 1x attempts to avoid all obstacles on the map which may include prisms, cylinders, etc., as well as aircraft received by ADS-B.
- **Detection sensor**: Obstacles previously defined in **Veronte Ops**.

2.6.2.2.15 Output

This action is used to set an output value in a GPIO pin. The output pin must have been configured as a **GPIO output** (visit *GPIO - Connections* section of this manual).

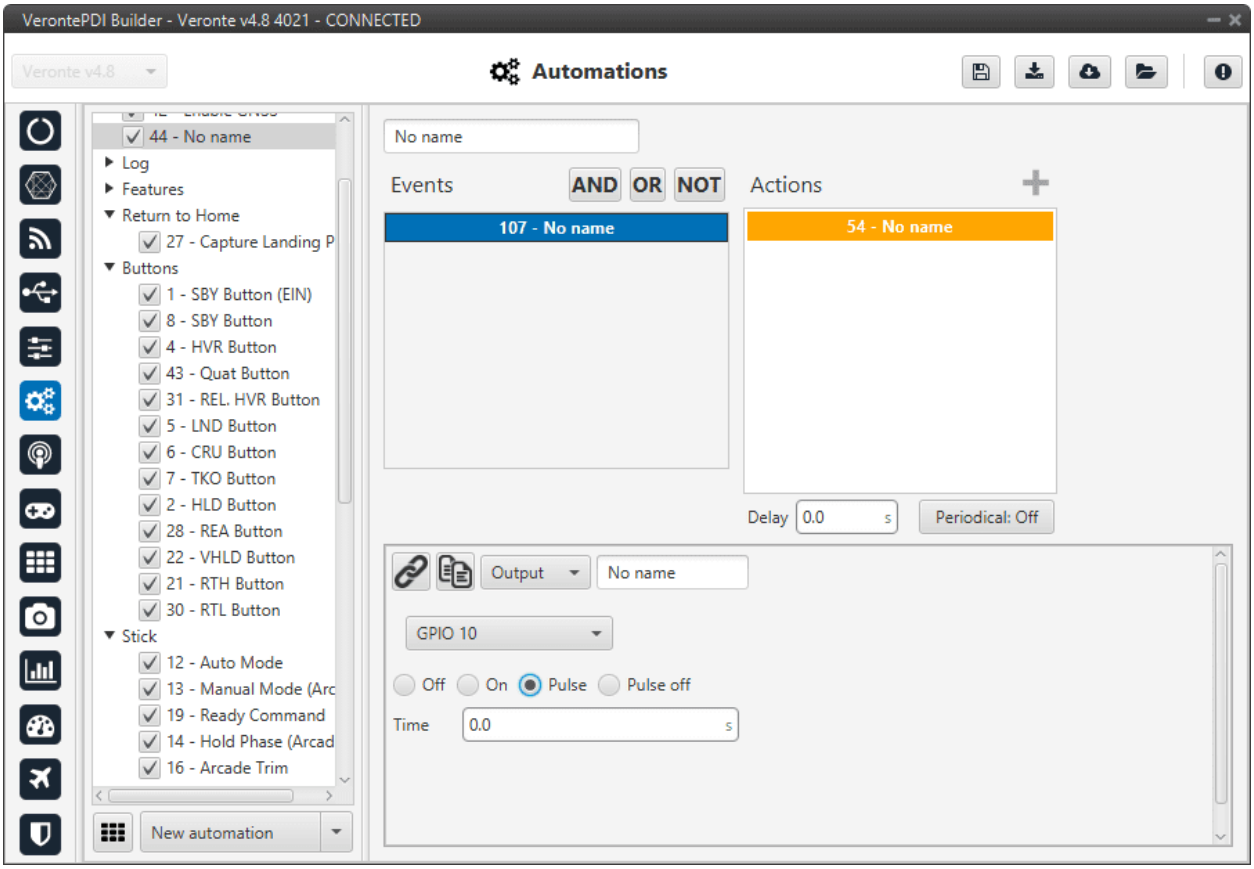


Fig. 194: **Output action**

The user can select, in the drop-down menu, between a GPIO output or a virtual output. The virtual option works like a normal GPIO output, but physically this output is not in the autopilot. It is used, for example, with a CEX.

There are four possible output signals:

- **Off**: Provides continuous 0V output.
- **On**: Provides continuous 3.3V output.
- **Pulse**: Provides 3.3V for the specified time and after that 0V.
- **Pulse off**: Provides 0V for the specified time and after that 3.3V.

2.6.2.2.16 Periodical

This action is used to set a timer during a flight operation.

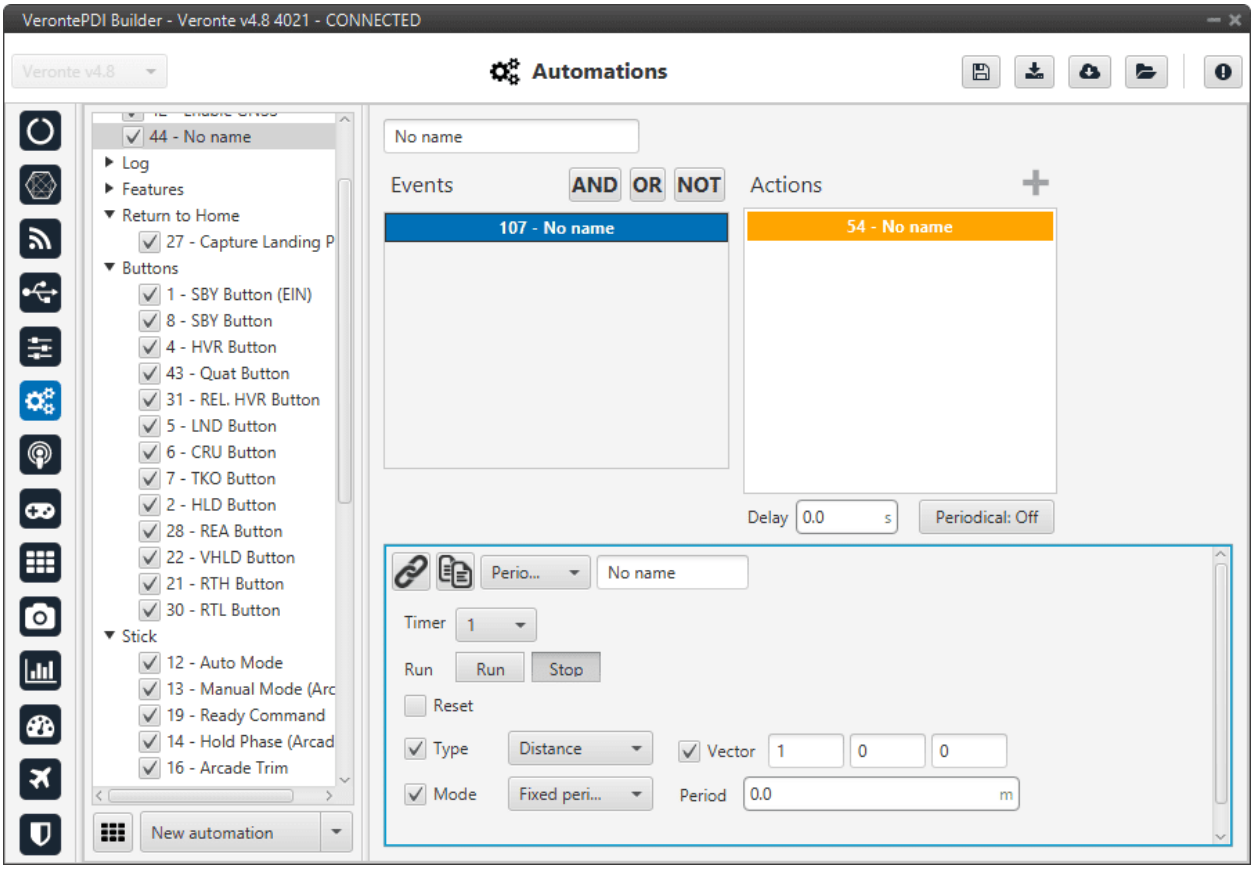


Fig. 195: **Periodical action**

The following parameters are configured:

- **Timer**: This parameter is an identifier for the timer, so it can be used in an event for another automation.
- **Run**: Sets the timer status:
  - **Run**: The timer will start.
  - **Stop**: The timer will be stopped. Another automation should be created to run it again.
- **Reset**: When this action is active, the timer is reset to zero before starting to measure.
  - **Stop + Reset**: The timer will be stopped and set back to zero.
- **Type**: These available options have been previously explained in *Automations*.
- **Mode**: The difference between fixed delay and fixed period has been previously explained in *Automations*.

For a better understanding of this action, a set of examples are detailed below with possible combinations of the different options.

- **Run + Distance/Time + Continuous**: When the action is triggered, the timer will be started and will measure distance/time from that instant until the moment when the autopilot is turned off (or until another automation acts on the same timer).

- **Run + Distance/Time + Fixed Delay/Period**: Once the action has been triggered, the timer will start to measure a distance/time. Each time the value indicated in Period is reached, the event linked to this timer (in another automation) will be triggered.

  For example, if the user wants to take a photo each 25 meters, in a first automation, the timer should have Distance in the Type option and 25 meters in Period, then in the second automation, an event of type Timer is created (and linked with the timer before created), so each time the timer reaches 25 meters the event will be triggered and the action will be carried out.

- **Distance + Vector**: The distance is measured in the direction indicated by the vector.

2.6.2.2.17 Phase

The flight phase is changed to the one selected in this action.

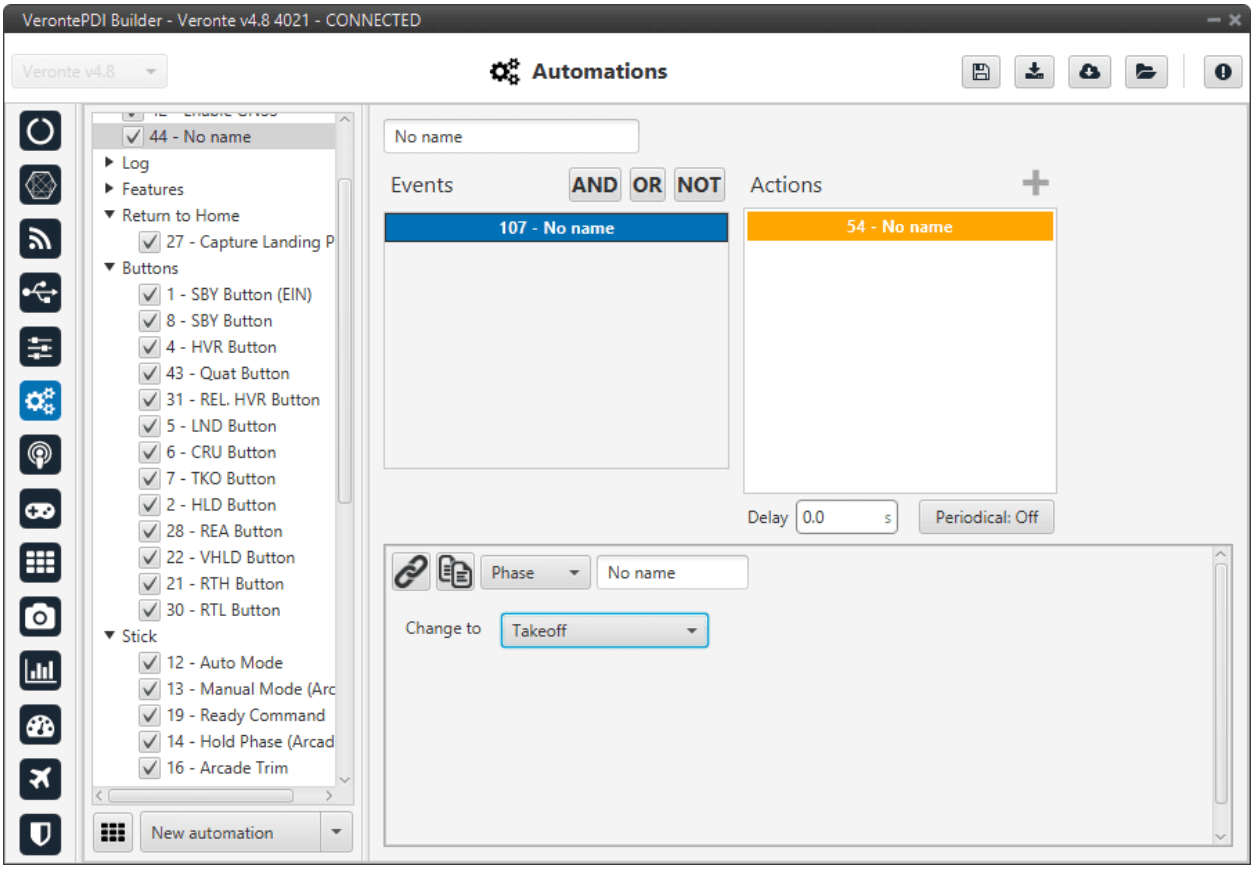


Fig. 196: **Phase action**

These phases have been created previously. See *Phases - Control* section, for more information about creating phases.

2.6.2.2.18 Ports

This action allows the user to switch between 4 pre-set configurations defined in the *Ports panel* of the **Communications menu**.

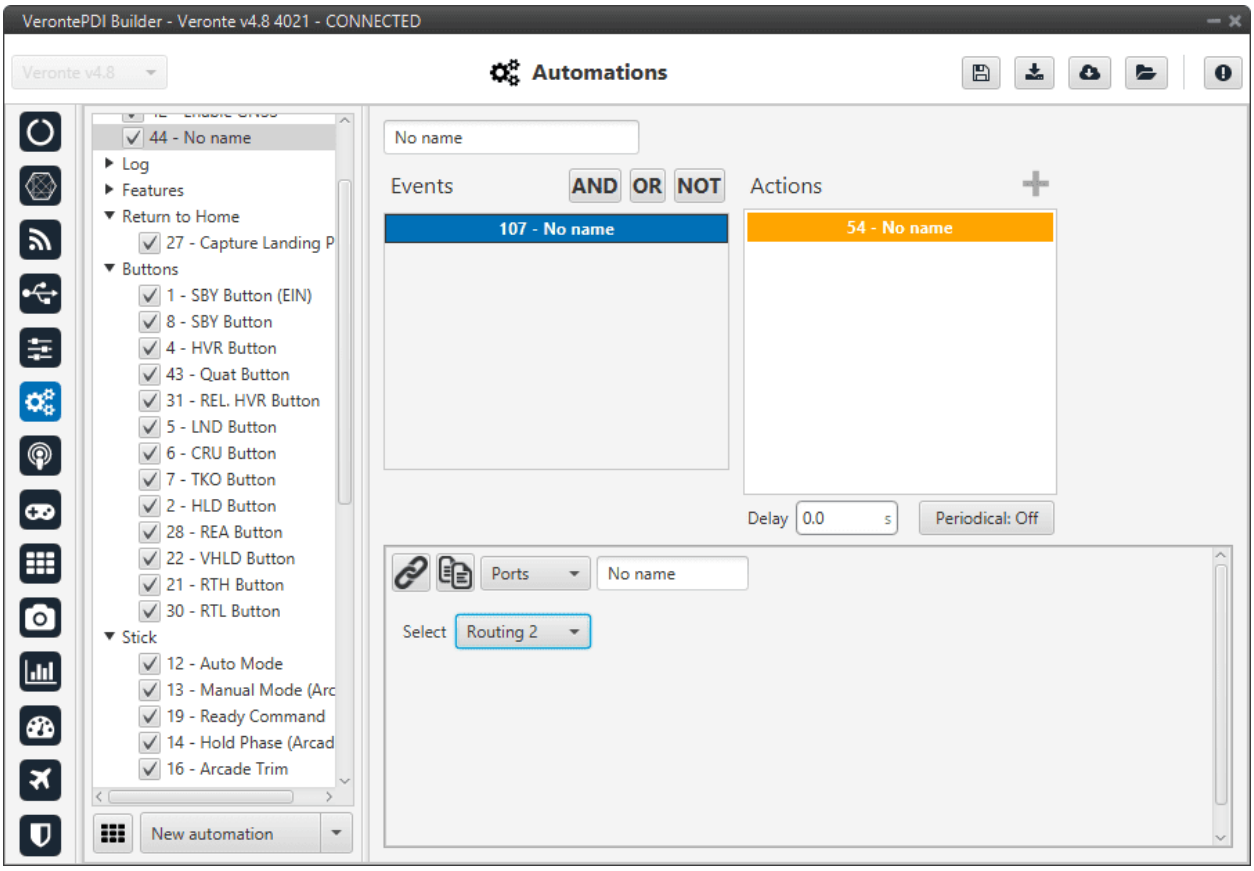


Fig. 197: **Ports action**

2.6.2.2.19 Run block program

When this action is triggered, the block program specified in the "Execute" label is executed.

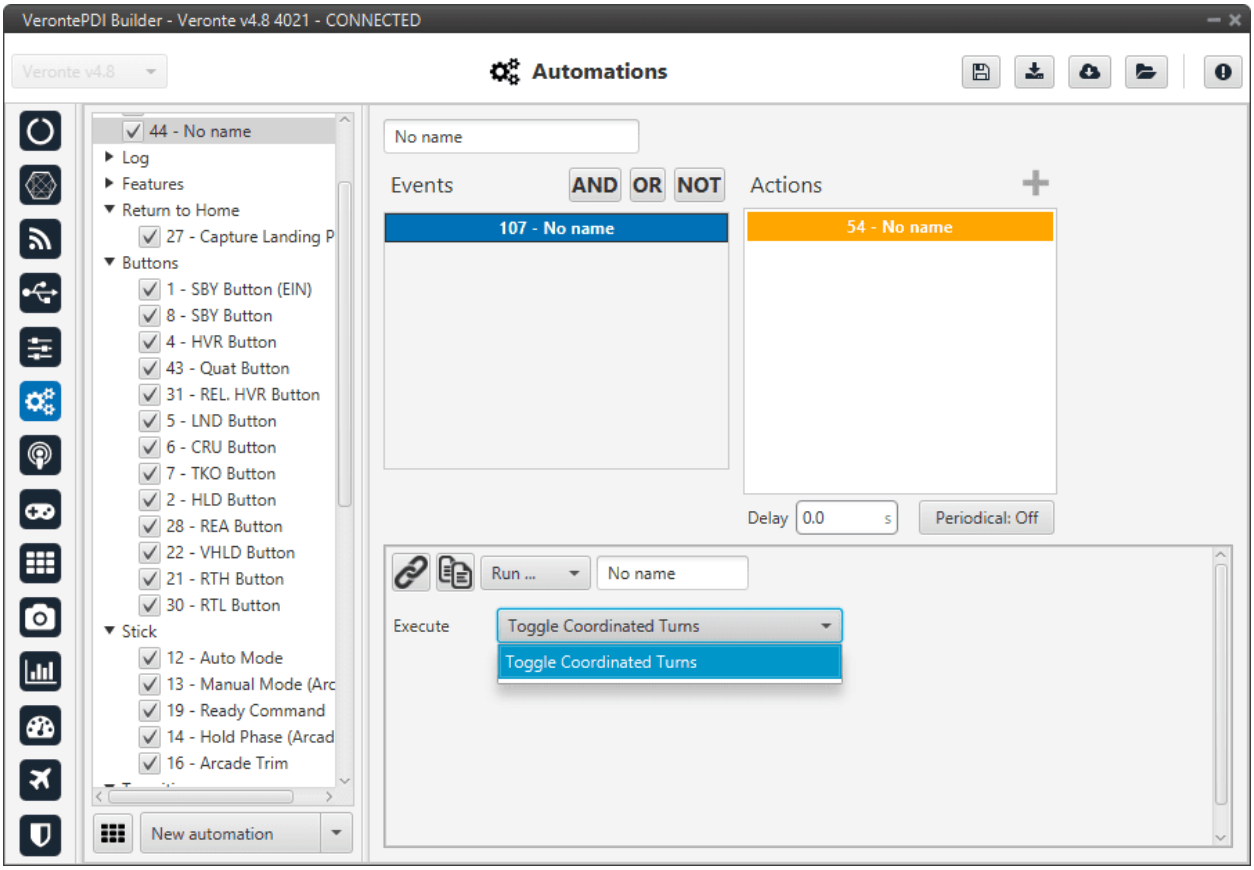


Fig. 198: **Run block program action**

This action can run those programs that have the lightning icon in grey color as those programs with the lightning icon in black color run continously. For more information about programs, see section *Block Programs*.

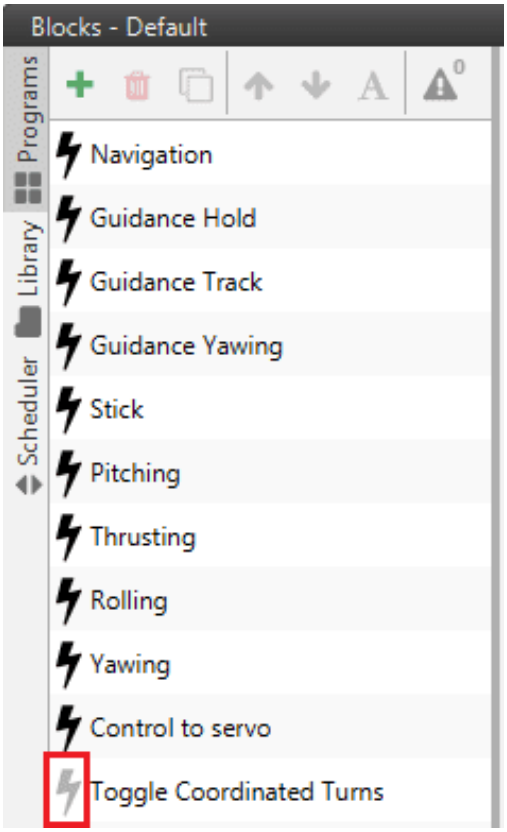


Fig. 199: **Grey lightning icon**

2.6.2.2.20 Safety Bits

This action selects a predefined safety bits list.

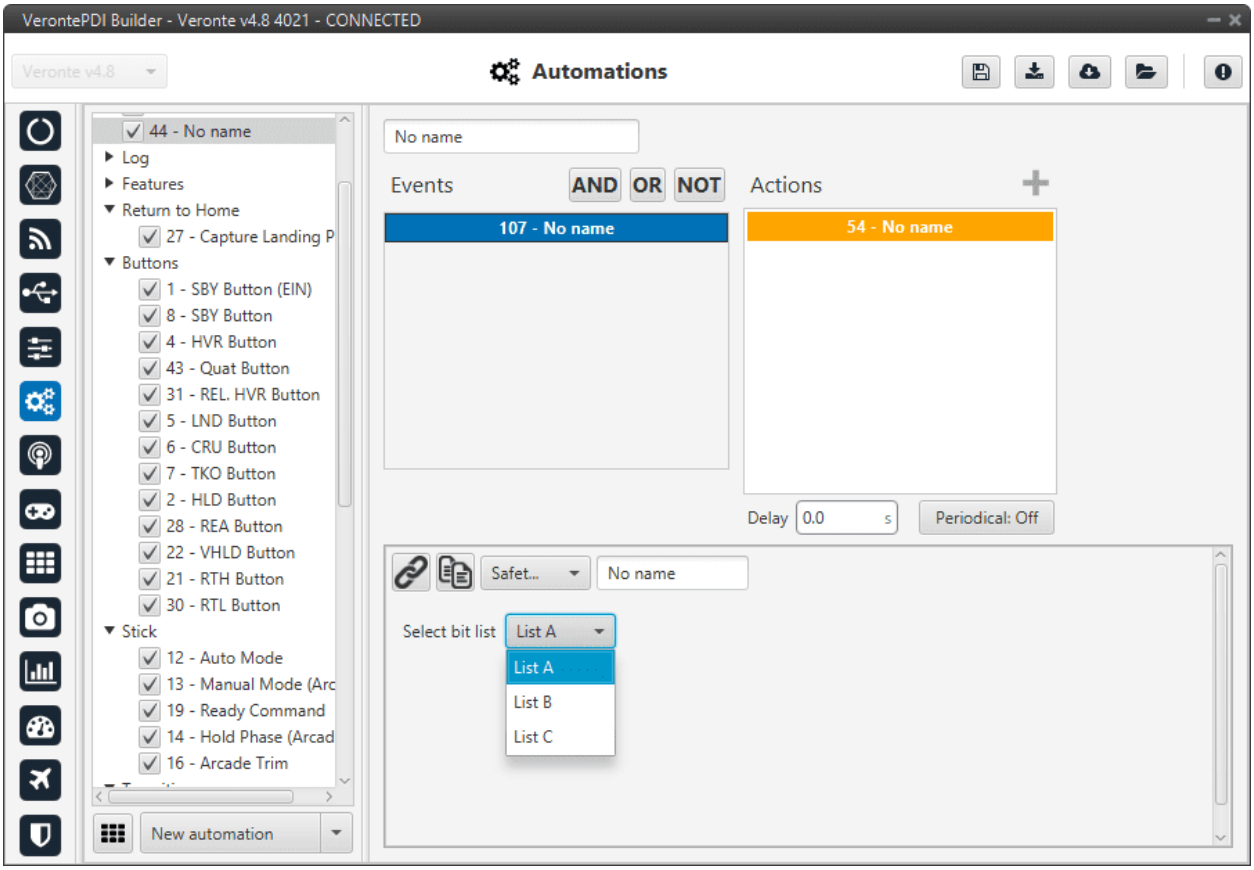


Fig. 200: **Safety Bits action**

This list must be previously configured. For more information about this, see *Safety bits - Safety* section of this manual.

2.6.2.2.21 Select Arcade axis

The axes system of the aircraft is changed to one that has been previously created in the *Arcade Axis panel* of the **Control menu**.

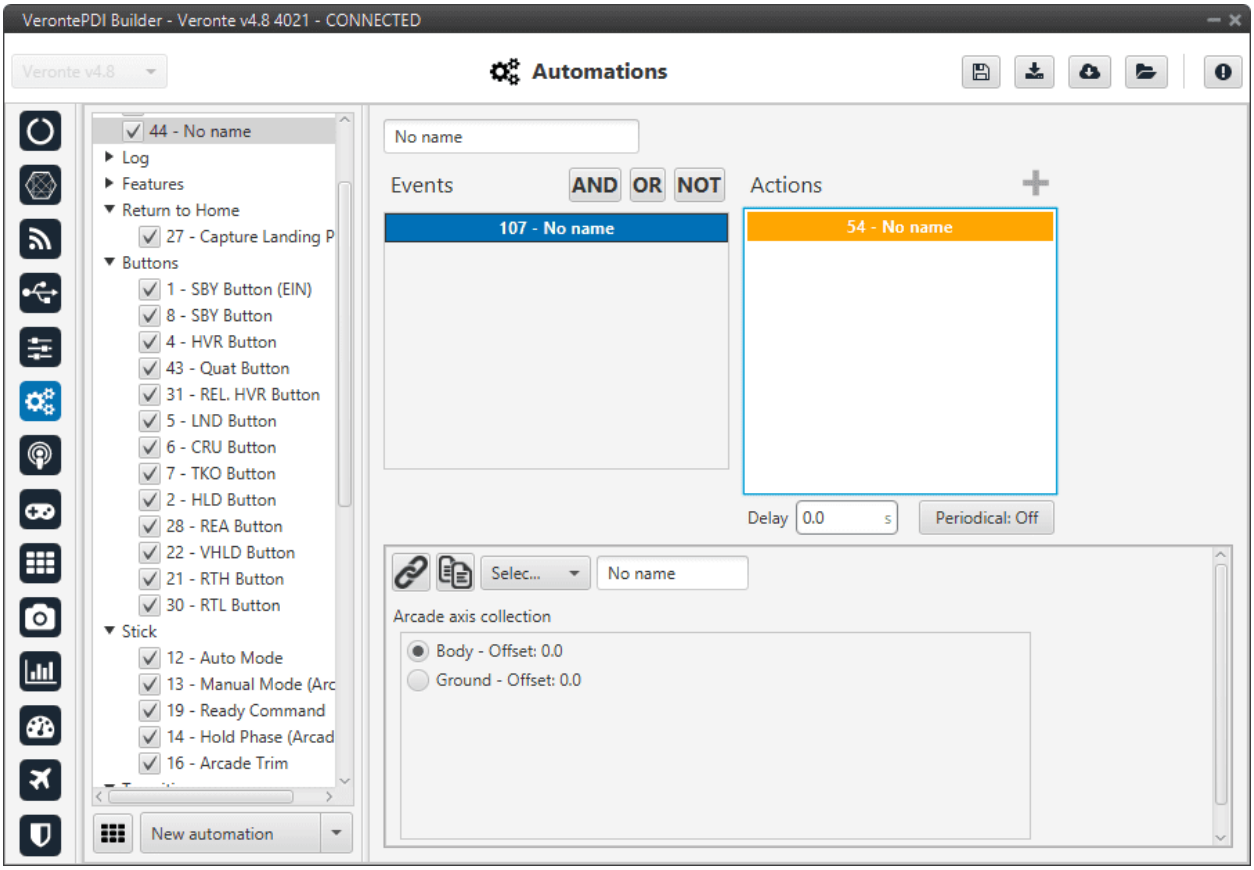


Fig. 201: **Select Arcade axis action**

2.6.2.2.22 Stick priority

The user can switch between the two priority tables of the **Stick block** (for more information about the stick block, see *Block Programs* section of this manual). By default priority table 0 is selected when Autopilot 1x starts.

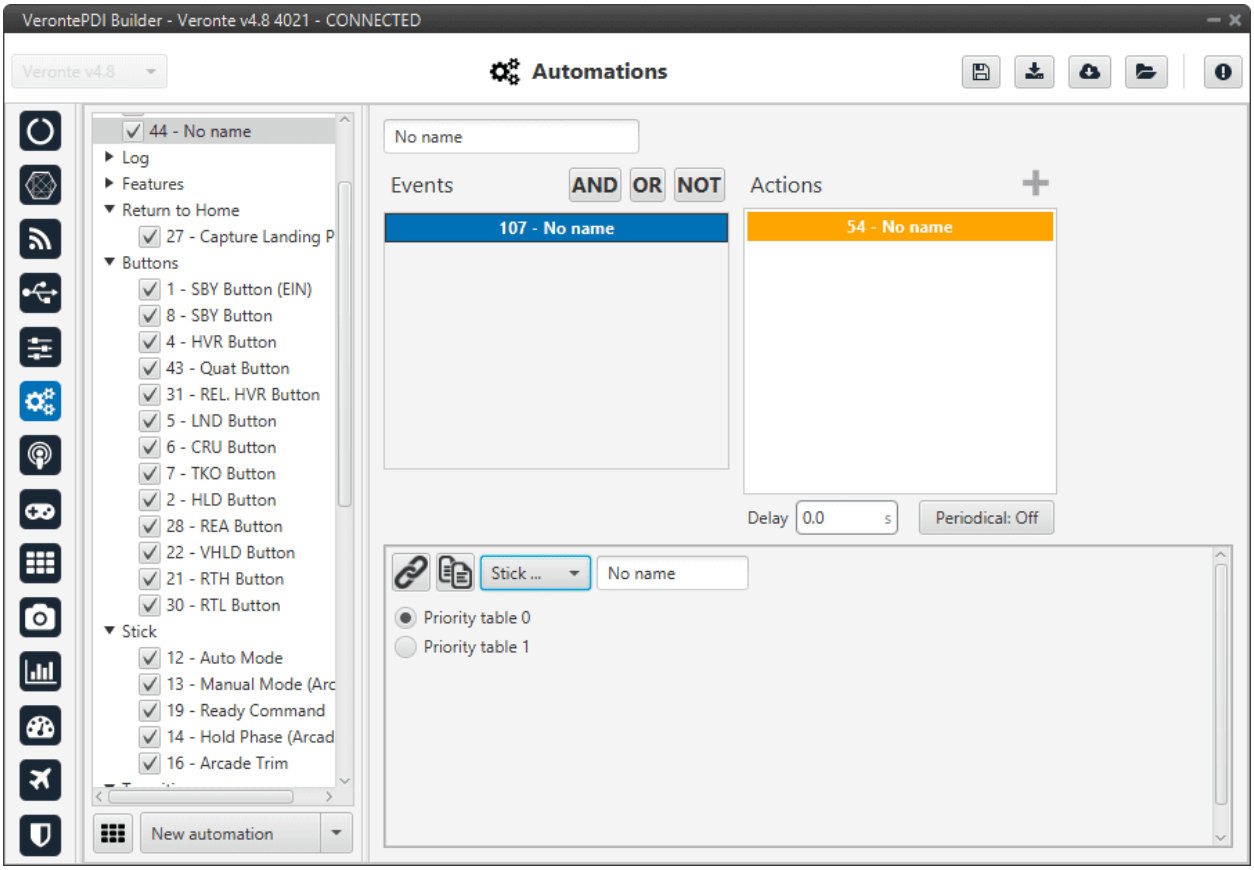


Fig. 202: **Stick priority action**

2.6.2.2.23 Terrain obstacle

This action is used to cause the aircraft to climb when it is reaching the ground (normally an altitude of zero metres), acting in the same way as an obstacle, with a **vertical repulsion**.

Caution:

- This only prevents "vertical" collisions.
- This option cannot be activated all the time because it will not allow the aircraft to land.

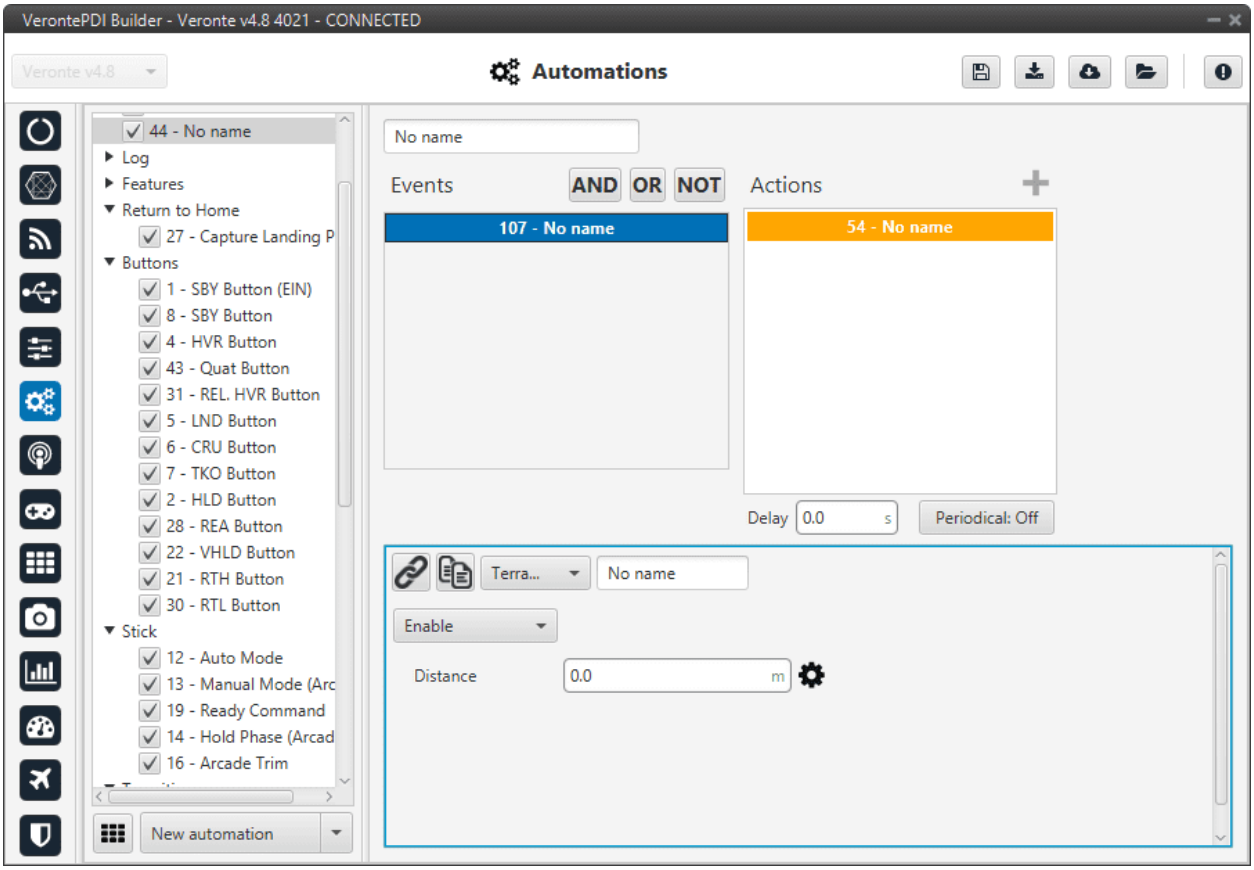


Fig. 203: **Terrain obstacle action**

- **Enable/Disable**: When this option is enabled, the ground + an offset (Distance) is considered an obstacle, in the same way as those that can be defined from **Veronte Ops**. For more information on obstacles, please refer to the Set Obstacle (Sphere item) - Mission section of the **Veronte Ops** user manual.
- **Distance**: Establishes the distance between the ground and the edge of the imaginary obstacle. The repulsion generated will be inversely proportional to the distance of the uav to this edge, being maximum when the uav is directly over the edge.

Important: The distance at which the repulsion starts will be inversely proportional to the acceleration set in the "**Obstacles/Geofencing**" configuration tab of the *Envelope block*.

2.6.2.2.24 Track

This action is used to configure a hover/loiter route (depending if it is a multicopter or an airplane) for the platform. Besides, there exists an option to follow a moving object.

There are 3 different options for the Track action, selecting **Disabled** no action will have effect on the guidance. The others are explained below.

Position

The aircraft will loiter/hover in a selected point.

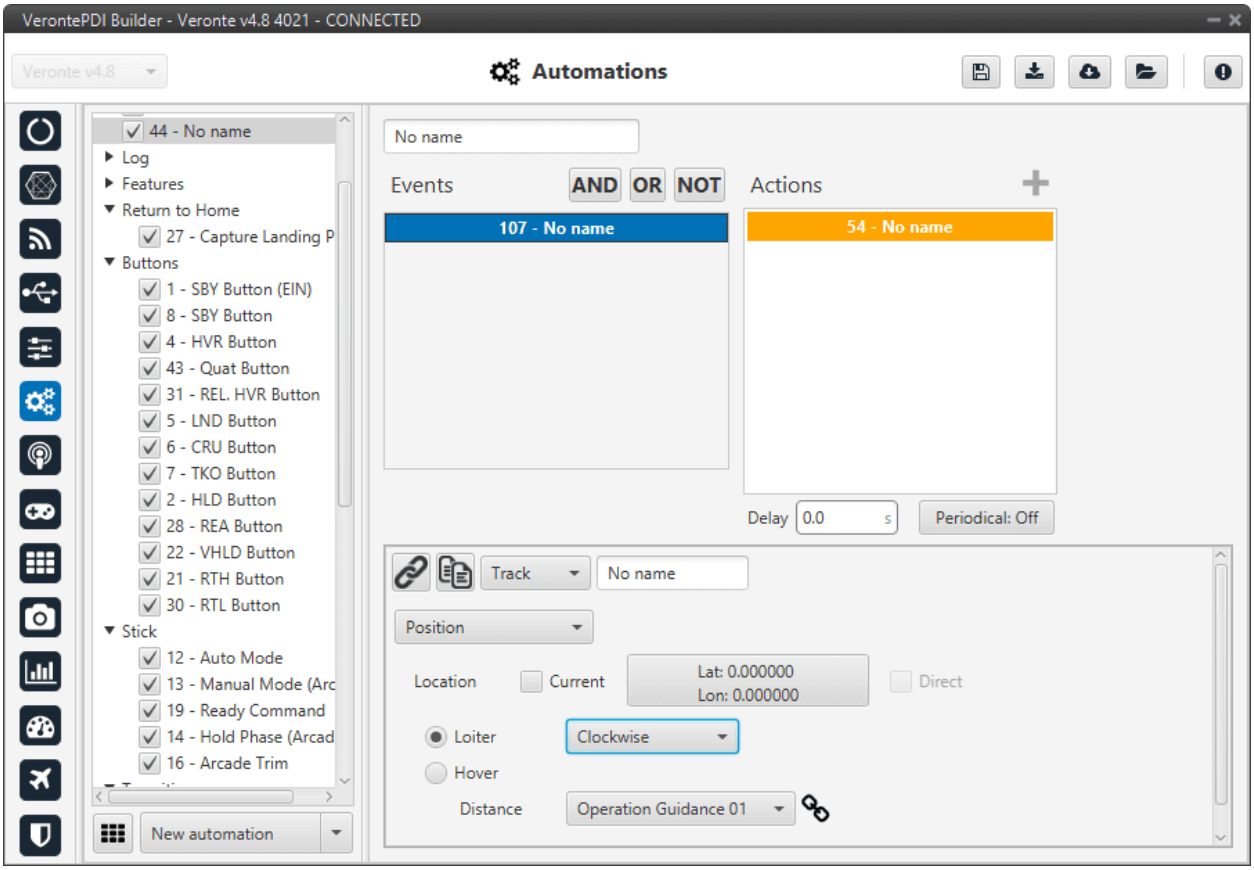


Fig. 204: **Track action - Position**

The following options are available:

- **Location**:
  - Selecting **Current** will make the platform to **hover** over the position that the vehicle has when this action is triggered, or **loiter** around that point in a circular route.
  - The box **Longitude**, **Latitude** allows the user to select the point where the hover/loiter will be performed.
- **Loiter**: It is also possible to select the direction of the loiter (**Auto**, **Clockwise** and **Anticlockwise**).
- When **Hover** is selected the option **Direct** can be enabled.
  - If direct is **enabled**, the autopilot will calculate the control actions to reach the desired point based on the position error with that point.
  - If direct is **disabled**, the autopilot will trace a path to the desired point and calculate the necessary control actions for this 'new route'.
- **Distance**:
  - **Distance + Loiter**: In this case, distance indicate the radius of the loiter circular route.
  - **Distance + Hover**: This option allows the user to define an acceptance radius around the position of the hover center. If the UAV position is inside this circle, then Autopilot 1x considers it is hovering correctly and will keep the position. If the center of the hover changes its position and the UAV position is out of the hovering area, 1x will fly to the hovering center, and once it is inside the circle, the hover will start.

Follow Leader

The platform will follow a moving object.

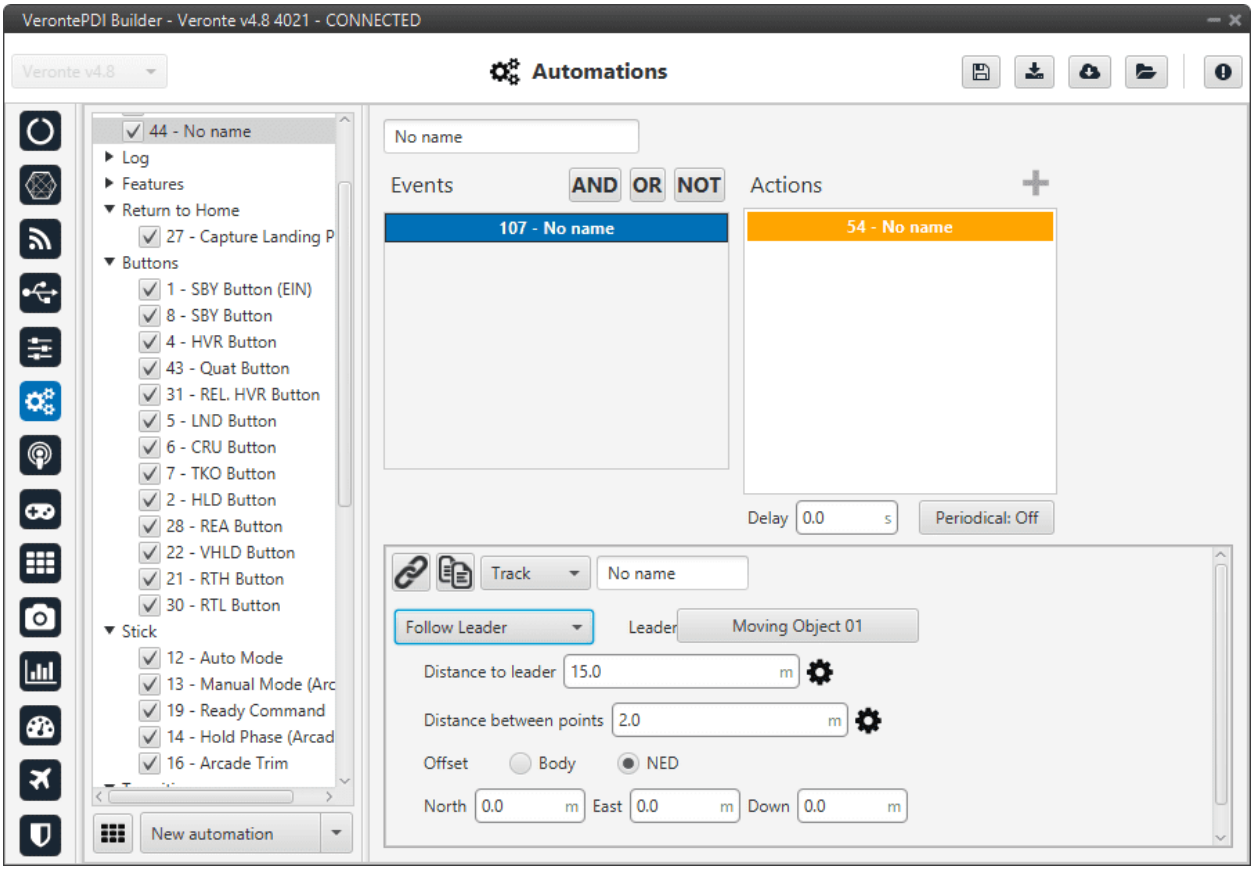


Fig. 205: **Track action - Follow Leader**

In this action the following parameters can be configured:

- **Leader**: Here is selected the object to follow, e.g. Moving Object.
- **Distance to leader**: Distance to leader over trajectory.
- **Distance between points**: Leader route is generated by points separated by the distance specified here.
- **Offset**: User can establish offset parameters related to trajectory in Body or NED coordinates.

Note: To configure correctly this automation, the user has to follow the next steps:

- Configure Telemetry in Air and Ground units.
- Configure the automation as desired.

2.6.2.2.25 User Log

An entry, previously configured in the *User Log panel* of the **Telemetry menu**, is added to the on-board log.

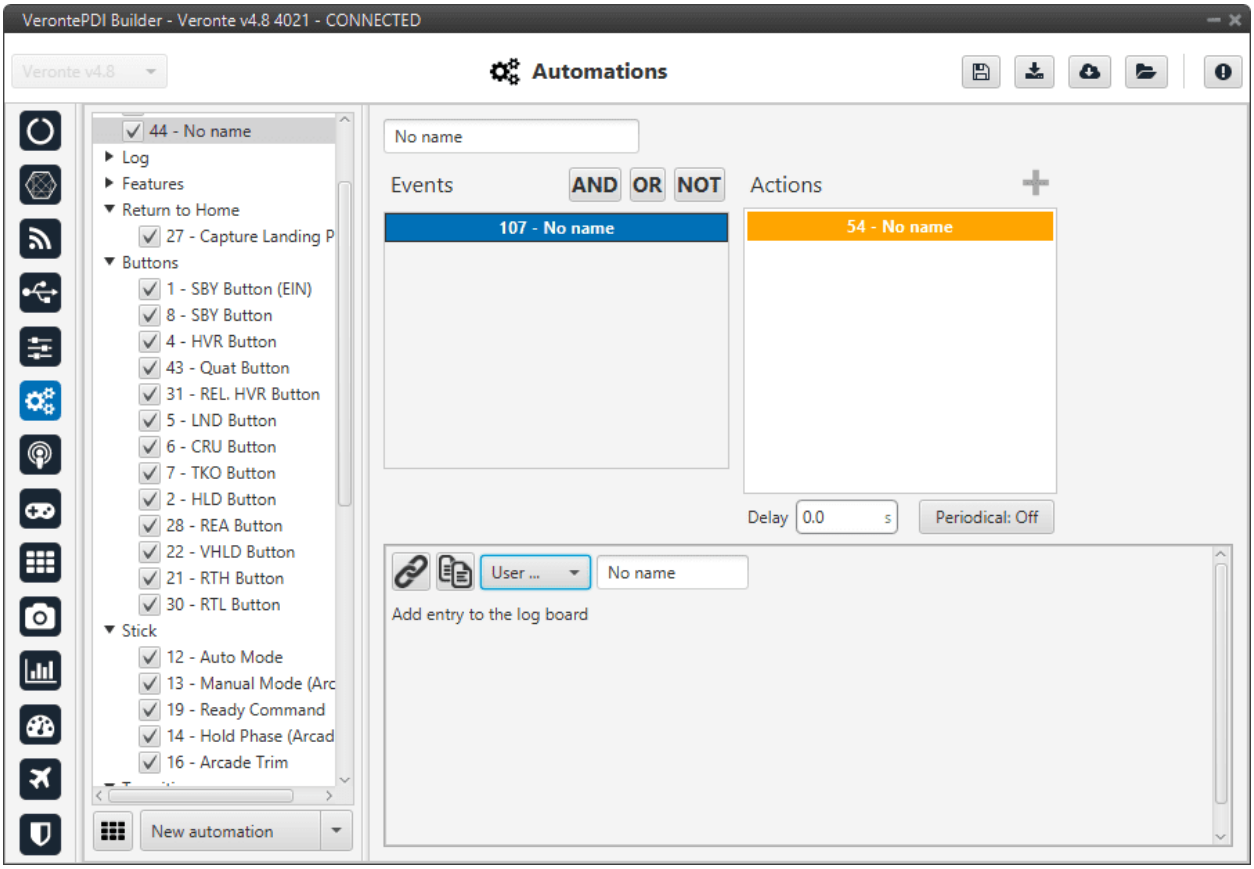


Fig. 206: **User Log action**

2.6.2.2.26 Variable

This action allows the user to select variables or enter values and store them in user variables.

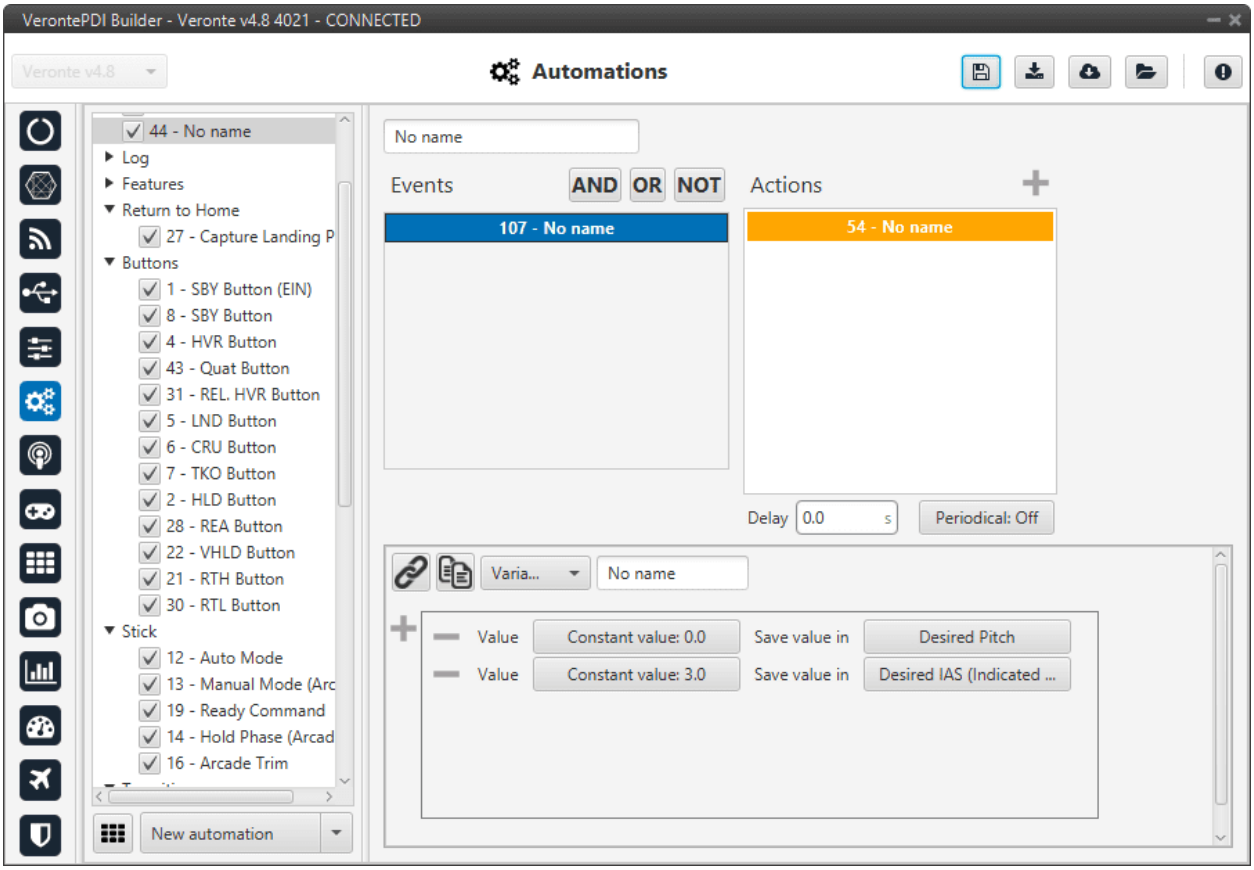


Fig. 207: **Variable action**

2.6.2.2.27 Yaw

When this action is triggered, the current yaw will be set to the one configured here.

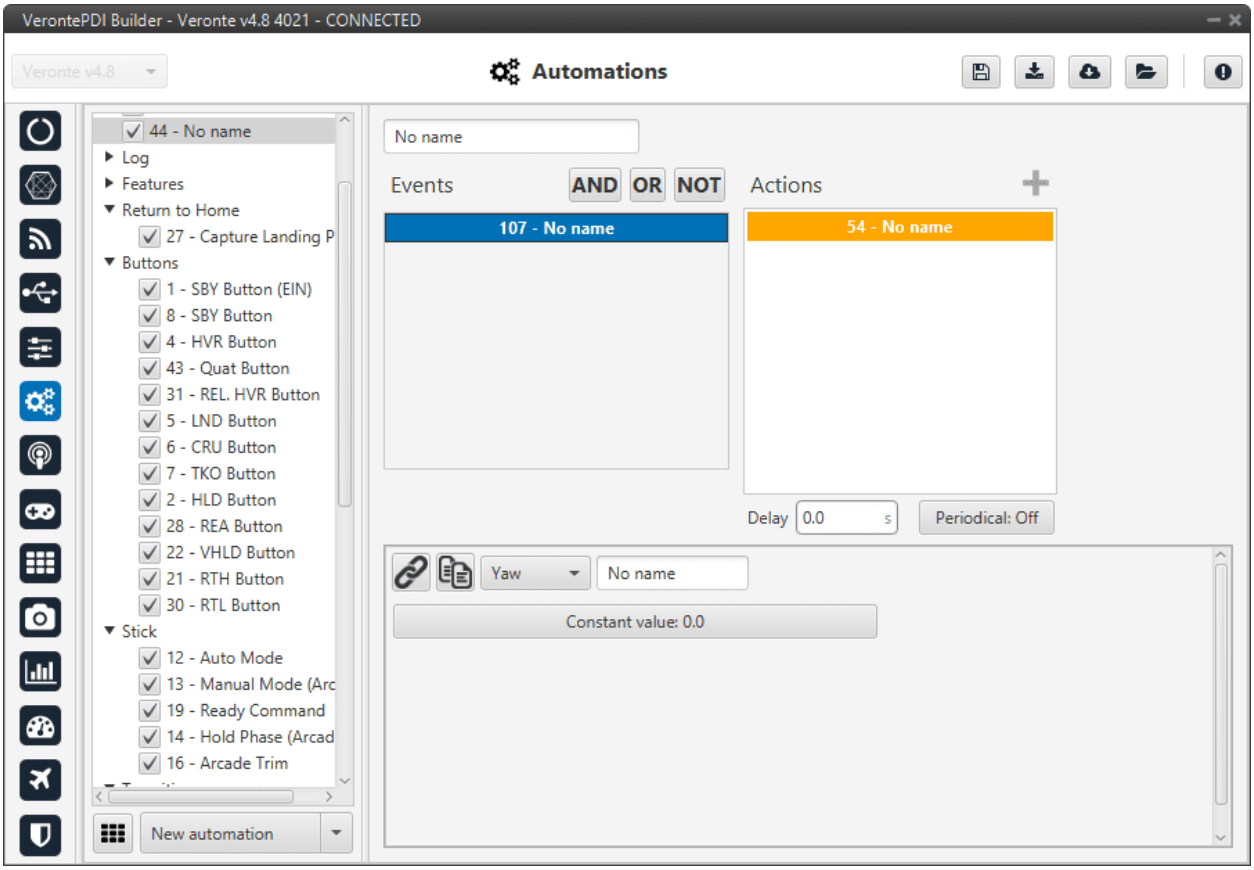


Fig. 208: **Yaw action**

This action is useful when flying without a magnetometer, as the user can establish the current yaw value when it is known.

2.7 Communications

2.7.1 Ports

Ports configuration allows the user to configure which communication ports (Commgr Ports in *I/O Setup*) will be used for communication. When using the **Route** feature, Autopilot 1x can be configured to **route** VCP messages for an external Veronte device with a known address (ID) through a given port.

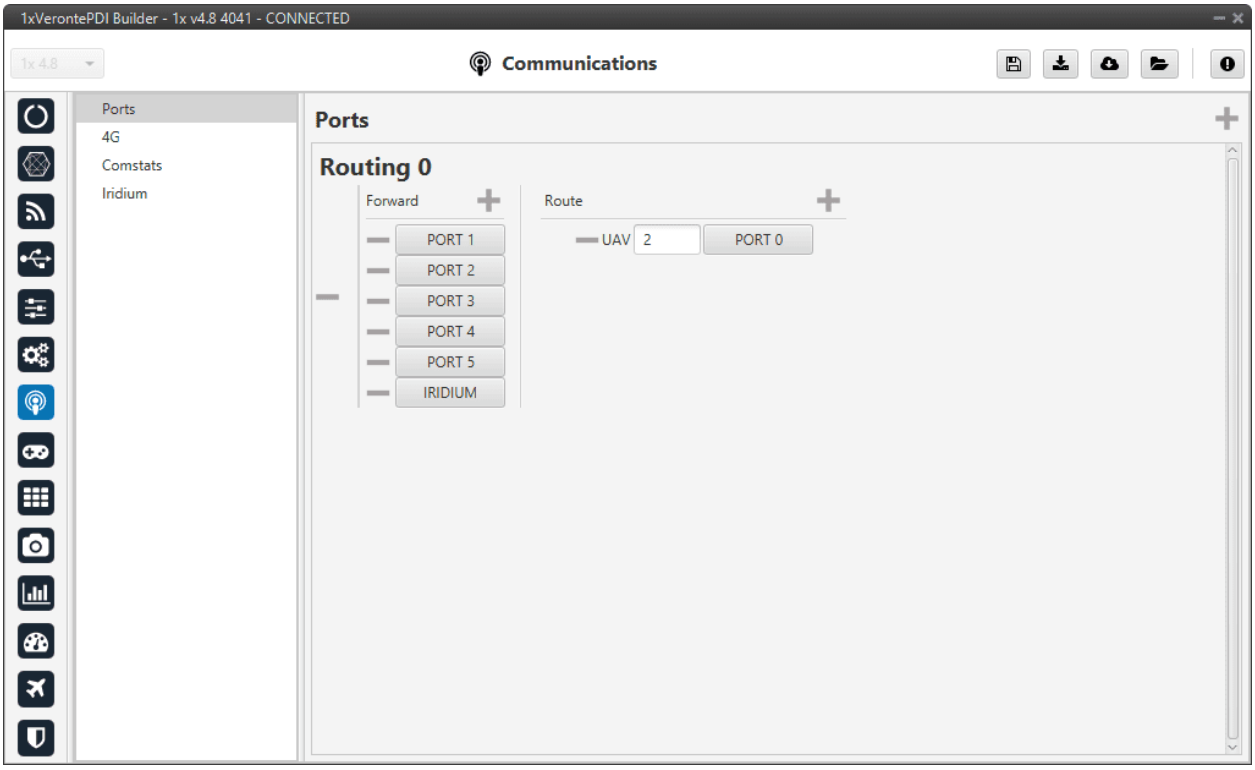


Fig. 209: **Ports panel**

Each of the different ports can be configured as either of the following options:

- **Forward**: Any messages generated by this unit (i.e. Telemetry or response messages to certain commands) will be sent through these ports.
- **Route**: Any messages received at any Commgr Port with the defined address will be re-sent through the defined port. It is possible to route several addresses through the same port, but is **not possible** to route the same address through several ports. Only the first configured port will be used. Routing also applies to messages generated by the unit for the defined address.

Note: The same **port** cannot be used as **Forward** and **Route** at the same time.

It is possible to define up to 4 routing setups, which can be switched unsing the *Ports action* of the **Automations menu**. Routing 1 will always be selected by default when booting Autopilot 1x.

Application example

A practical example of the use of this menu are 1x ground unit configurations. These configurations should have configured a Routing of **Address 2** (**Veronte applications**, in this case Veronte Ops) through the Commgr port to which the **USB consumer** is connected (to allow connection to the PC). In this example, **Port 0 producer** is the one connected to USB consumer.

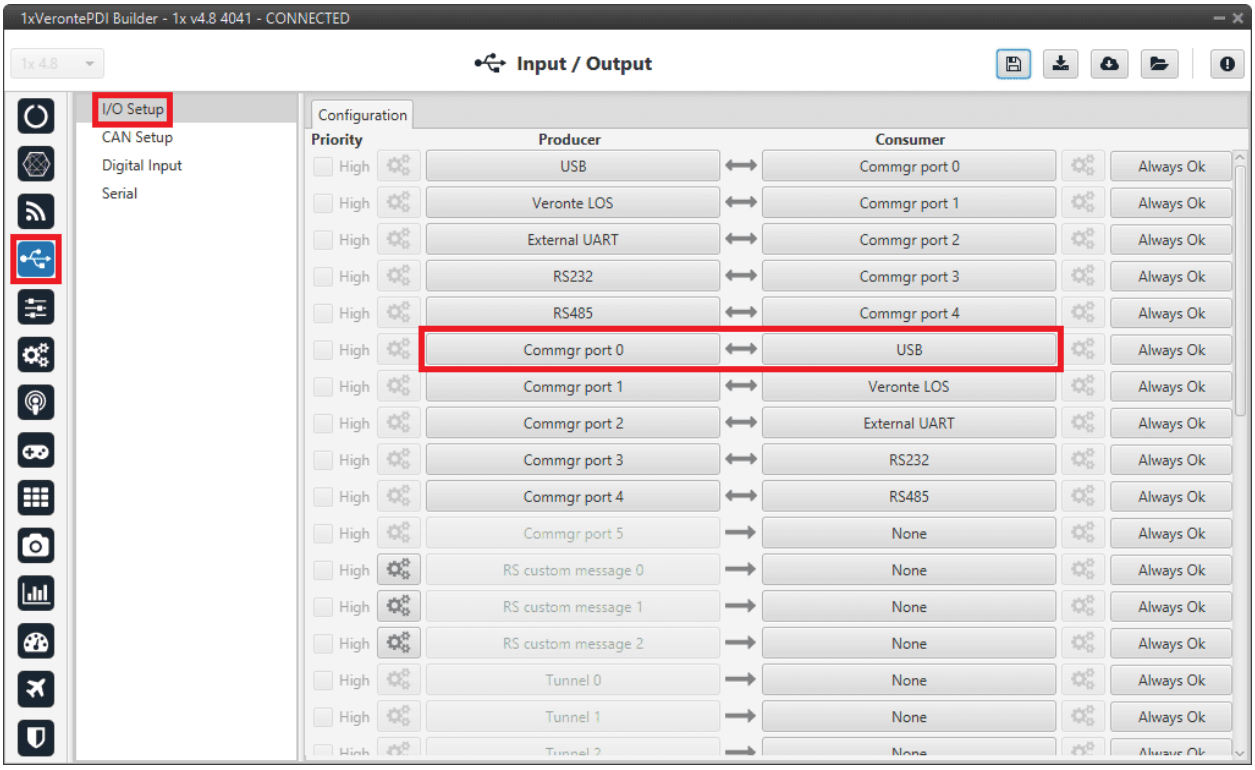


Fig. 210: **Ports panel - I/O Setup configuration**

This way, any messages that are received through a Commgr Port (i.e. through Veronte LOS) with **address 2**, will be re-routed through Port 0 (USB) and received by **Veronte Ops** software, including any messages generated by 1x ground unit itself.

Warning: An incorrect Port configuration can **disable USB communication**. If this happens, 1x will not be able to be detected through **Veronte Link** software. If this is the case, please visit Forcing maintenance mode - Troubleshooting section of the **1x Hardware Manual**.

2.7.2 4G

Checking the **Enable** box will enable the use of 4G communication through the **Veronte LTE** Consumer/Producer in the *I/O Setup*.

Warning: Be careful when configuring 4G communication on the Veronte Autopilot 1x as the **configurations** for Autopilots 1x with `hardware version 4.8` and for Autopilots 1x `hardware version 4.5` are **not compatible**.

ESIM

The embedded **ESIM** in Autopilot 1x allows the user to send and receive telemetry using a commercial data provider.

The connection between the air unit and the ground station is established through the Veronte Cloud server. To connect with Veronte Cloud the following parameters have to be set:

- **Host**: rt.utm.systems

- **Port**: 3114

Host and Port can be changed if the used server differs from Veronte Cloud, but the communication protocol does not change.

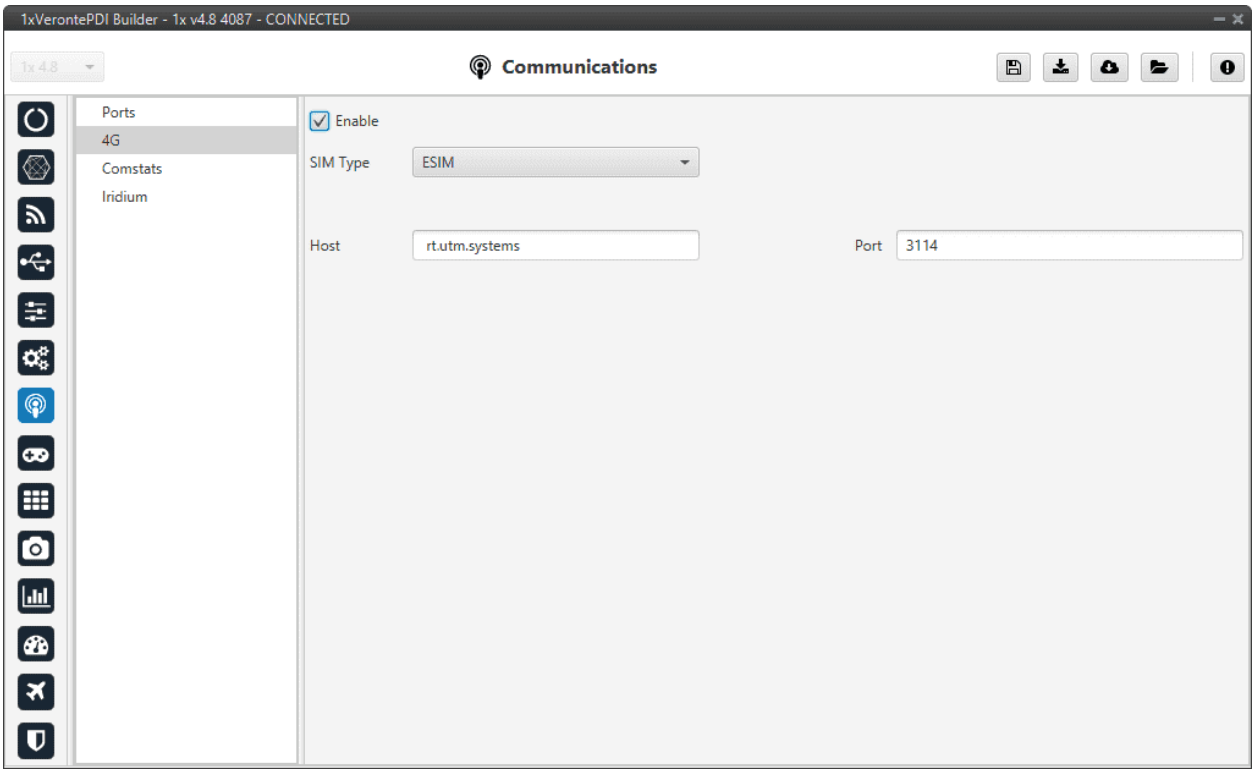


Fig. 211: **4G panel - ESIM**

Note: In order to use the embedded **SIM** card, the contract with the data supplier needs to be done through **Embention**. Please contact sales@embention.com for more information on availability, coverage, suppliers and prices in your country.

SIM

If needed it is also possible to install a custom SIM card on Autopilot 1x. PIN number and APN (Access Point Name) of the SIM card provider must be defined before enabling the 4G communication

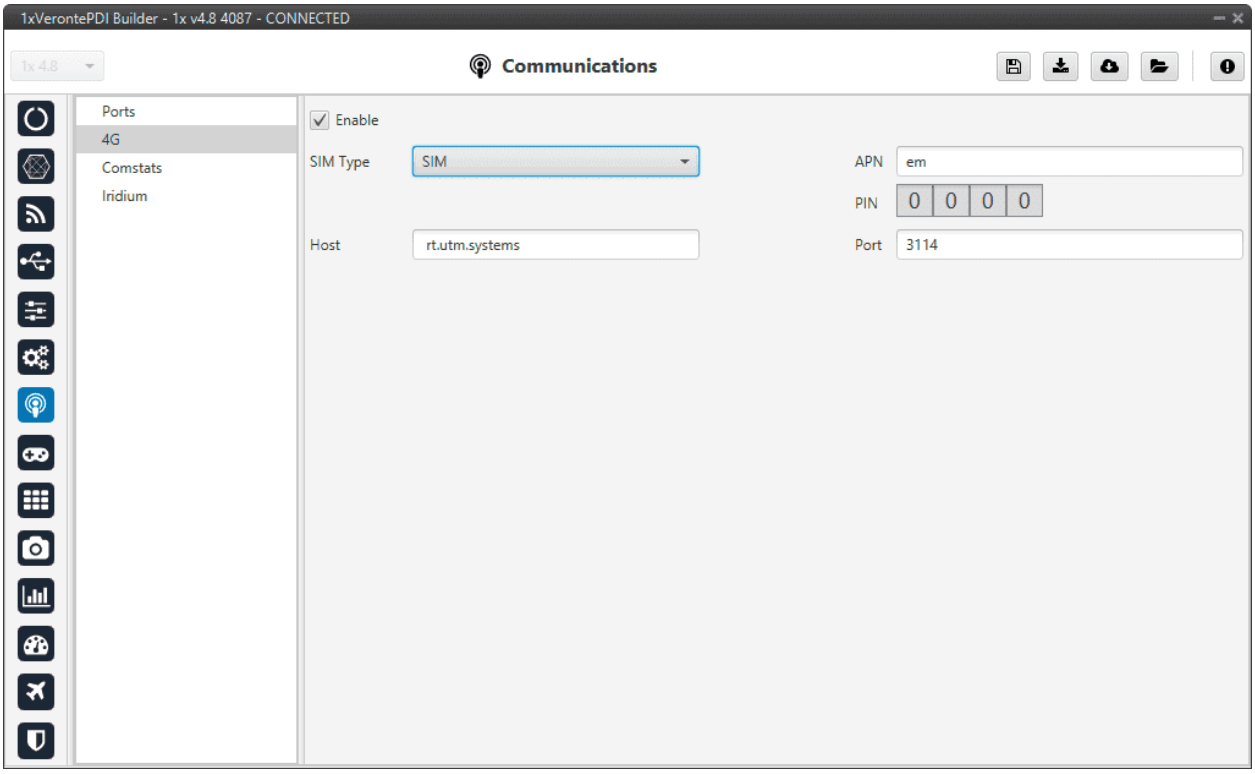


Fig. 212: **4G panel - SIM**

Warning:

- Introducing the **wrong PIN** number may **block** the SIM card.
- The installation of the SIM card must be done by **Embention** during the production of the unit. Please make sure to indicate the interest on using a Custom SIM card when ordering new 1x units.

2.7.3 Comstats

The Comstats feature allows Autopilot 1x to make an estimation of the overall **quality of the communication** channel.

1x will send periodically (**If enabled**) a message with its current communication statistics (packets sent and received per second). Then, any other 1x unit can receive this information and compare against its own statistics to estimate the average amount of packets lost in the communication.

The results of this estimation can be monitored in variables **RX Packet Error Rate** (ID 2000) and **TX Packet Error Rate** (ID 2001). These variables can be used to enable, for example, failsafe actions in case of degradation or loss of communications.

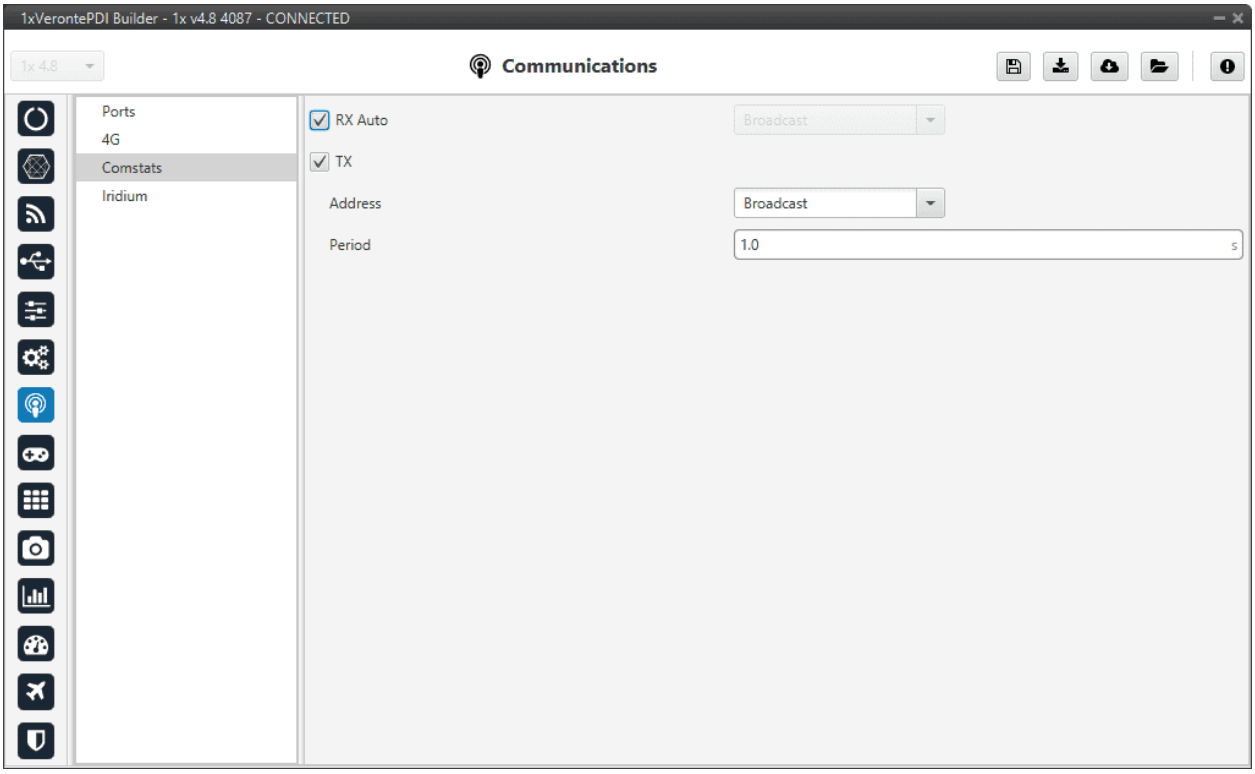


Fig. 213: **Comstats panel**

It is possible to configure the source or destination of the statistics, as well as the frequency at which the Comstats message is sent:

- **RX Auto:** Enabling this option will use the first remote AP found. If this option is disabled, the user must enter manually the address of the unit used for Comstats calculation. For more information on the available addresses, see List of addresses section of the **1x Software Manual**.
- **TX:** When enabled, the unit will periodically send its Comstats message (set the period). Enter the address to which the message should be sent. For more information on the available addresses, see List of addresses section of the **1x Software Manual**.

Note: Enabling **TX** will enable Autopilot 1x to send its Comstats message, but in order to compute Packet Error rate it's necessary to **receive** the **TX** message from a different unit.

Warning: **Packet error rate** is a good indicator of the status of the communication, but it is not representative of the radiolink status. For monitoring the status of the radiolink RSSI Variables (820-822) shall be used instead. Depending on the configuration it is posible to have bad Error rates with good RSSI (overloaded radiolink) or good Error rates with bad RSSI (degraded communication with low load on radiolink). For the best results, it is recommended to use a combination of both statistics for failsafe automations.

2.7.4 Iridium

Checking the **Enable** box will enable the use of Iridium communication through the **Iridium** Consumer/Producer in the *I/O Setup*.

Warning: Before using the module, the user will have to register both of them (sender/receiver) in the RockBlock website. For further information about the registration process, visit the RockBlock Management System.

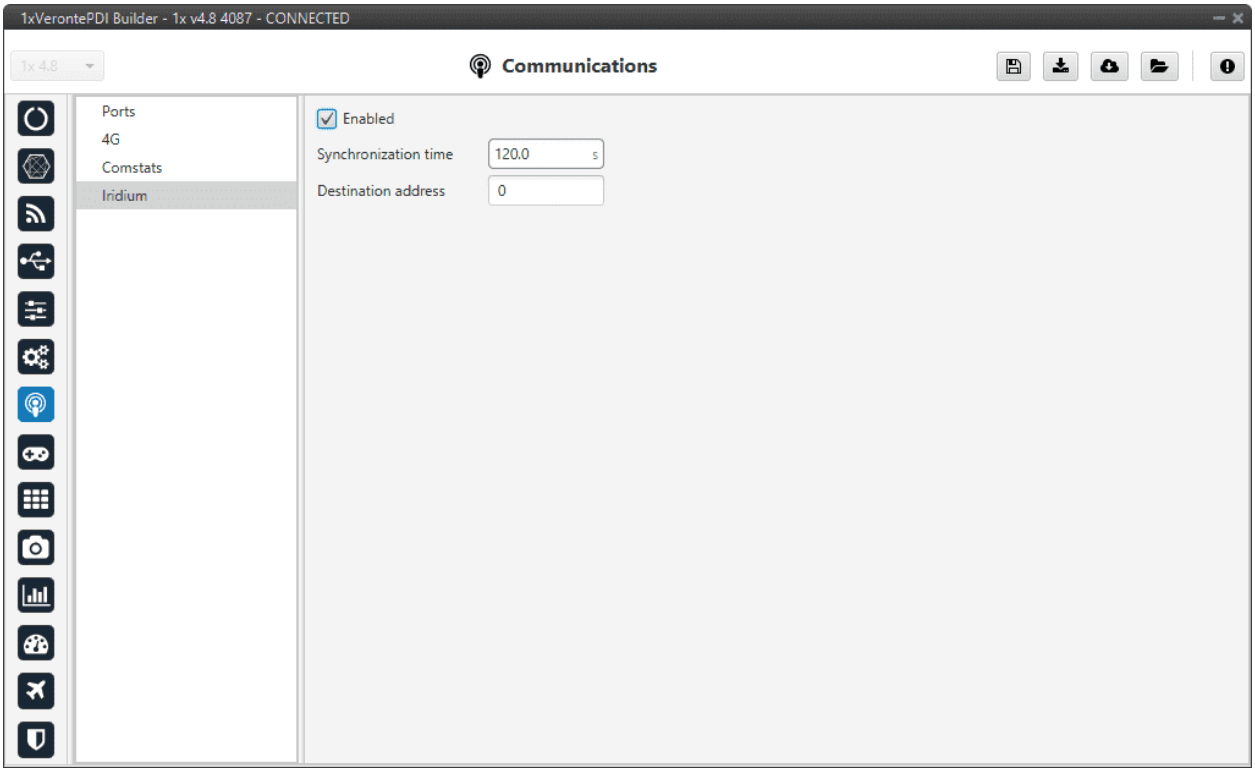


Fig. 214: **Iridium panel**

In this menu the following parameters have to be set:

- **Synchronization time**: This is the transmission period, i.e., the time between 2 consecutive messages. This is a parameter that the user should configure taking into consideration its mission.
- **Destination address**: SN (Serial Number) of the destination Iridium module.

Note: To configure the syncronization time, it would be advisable to think about how the user want to use the Iridium communication. The user will pay for credits, and each credit means one message. Each individual message has to be paid, so the syncronization time can be configured in order not to run out of credits.

2.8 Stick

In this section, the stick configuration on **1x PDI Builder** is explained.

This menu allows the user to set **up to four transmitters** and **one virtual stick**. The autopilot's capabilities allows it to receive information from four different transmitters at the same time plus transforming some values into a virtual stick.

The content presented in the next menus covers:

- Setting of the transmitter's parameters.
- Definition of exponential response-curves for the desired channels.
- Trimming of the channels' neutral position.
- Setting of the data receiving port on the autopilot.
- Definition of a virtual stick.

2.8.1 Transmitter (0-3)

The wired connected transmitters are configured through the following panels.

2.8.1.1 PPM

This panel provides the options to configure a Pulse Position Modulation (PPM) radio controller to control the platform fitted with the autopilot.

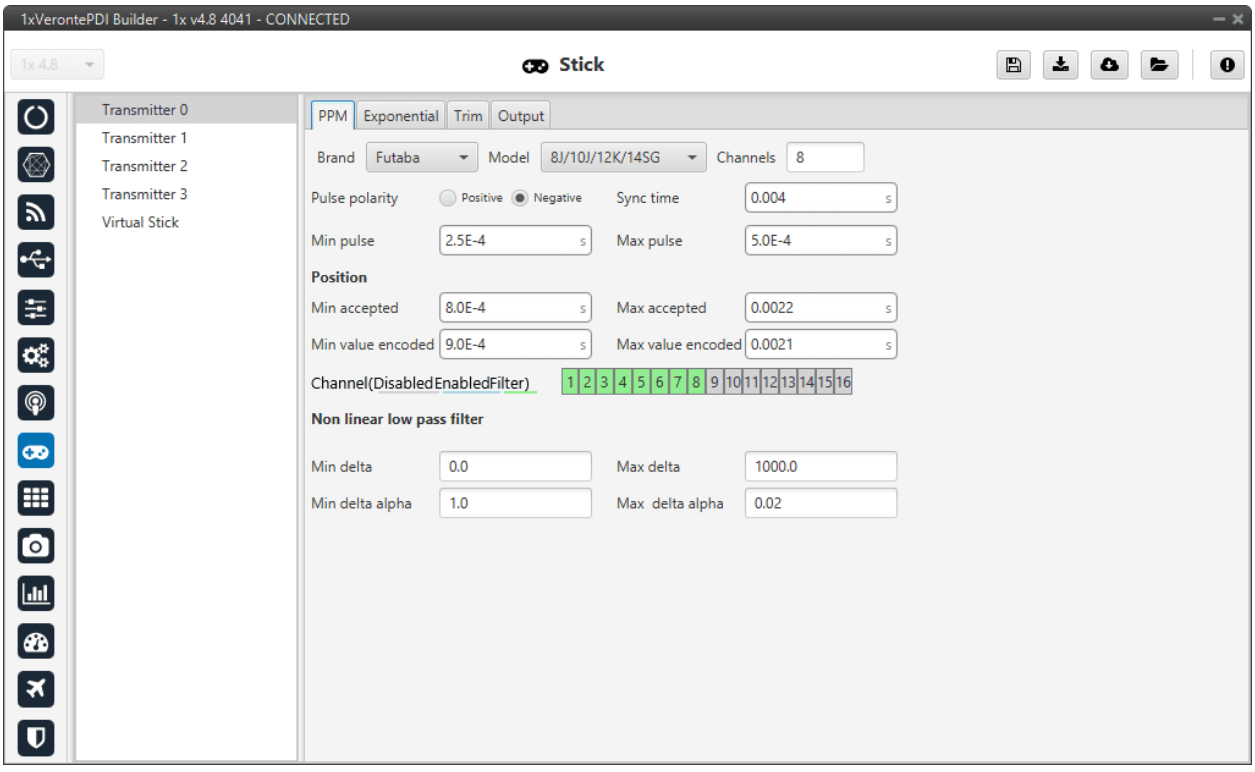


Fig. 215: **PPM panel**

- **Brand**, **Model** and **Channels**: **1x PDI Builder** has been configured to provide the user with the expected parameters to configure different transmitters models.

Brand	Models	Channels
Futaba	8J/10J/12K/14SG	8 (for 8J and 10J)
		12 (for 12K and 14SG)
	T18SZ	8
Jeti	DC 16/DC 24	16
FrSky	Taranis X9D	8
	Horus X12S	8
TBS	Crossfire	8
Embention	Stick Expander	16
Custom	-	-

  - Custom: If the user's transmitter is not among those mentioned above, choose this option and replace the parameter values with the appropriate ones.
- **Pulse polarity**: Indicates the pulse polarity:
  - **Positive**: Default signal is low and goes up to high.
  - **Negative**: Default signal is high and goes down to low.
- **Sync time**: Minimum time on the PPM output till the next frame. It tells the receiver to reset its channel counter.
- **Minimum/Maximum pulse**: Pulse length, it depends on the system and it is a constant value (usually 0.2-0.5 ms).
- **Position**
  - **Minimum/Maximum accepted**: Pulse length accepted for each channel. Standard for R/C servos uses a pulse of 1 ms for the maximum position at one end, 1.5 ms for the midpoint and 2 ms for the maximum position at the opposite end.
  - **Minimum/Maximum encoded**: If there is noise and the signal is varying around the minimum/maximum values accepted, Autopilot 1x will encode those values to the ones set here. For instance, a pulse length between 0.8-0.9 ms will be considered as one of 0.9 ms.
  - **Channels**: Sets the number of channels accepted. Besides, it is possible to Disable/Enable/Filter each channel individually.
- **Non linear low pass filter**
  - **Minimum/Maximum delta**: Default parameters are recommended.
  - **Minimum/Maximum delta alpha**: Default parameters are recommended.

The figure below shows the PPM signal that arrives to Veronte Autopilot 1x:

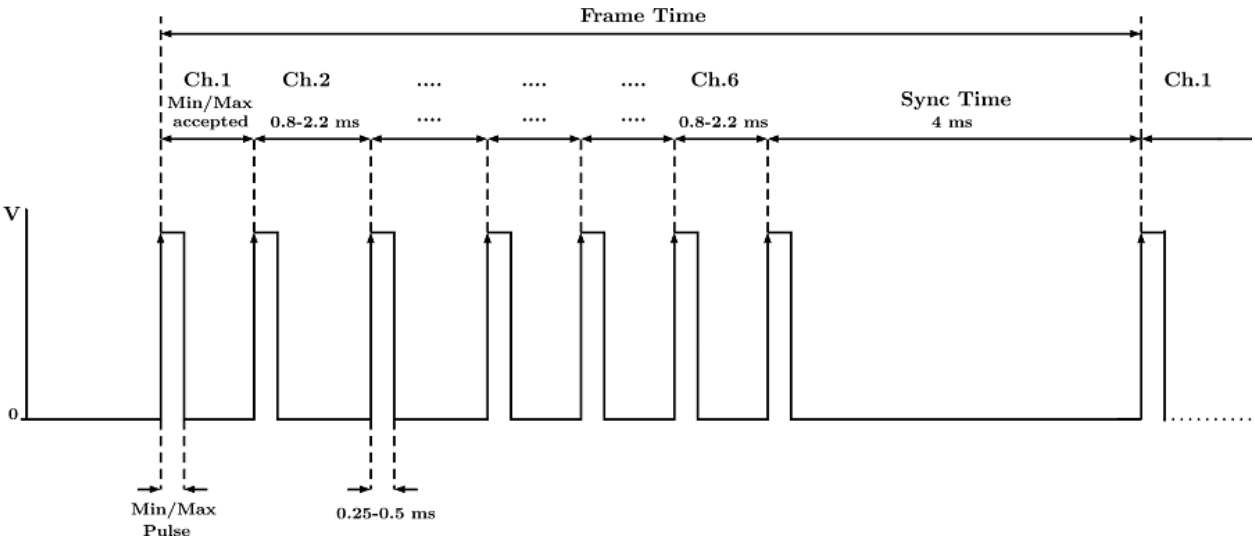


Fig. 216: **PPM signal**

2.8.1.2 Exponential

The second panel allows the user to define an exponential stick response for every channel.

The allowed inputs range from 0 to 1 and there is a graph showing the generated response curve, as can be seen in the figure below.

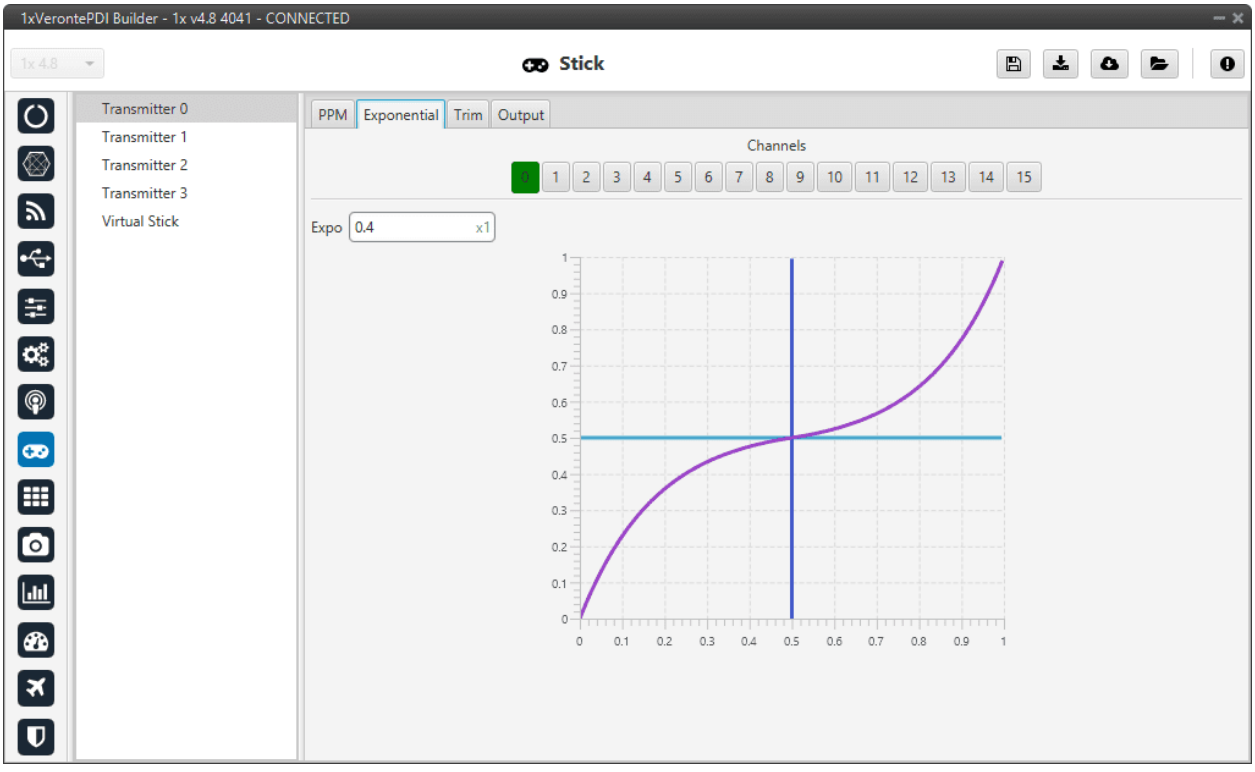


Fig. 217: **Exponential panel**

The **X axis** of the graph corresponds to the stick input and the **Y axis** is the result of applying the exponential function to that stick input.

2.8.1.3 Trim

By enabling the **Avanced** option, the user can set the expected trim values manually. The user should have a deep knowledge on its transmitter if this option is selected.

Finally, on the right hand side, the **Reset** button puts every parameter back to 0.

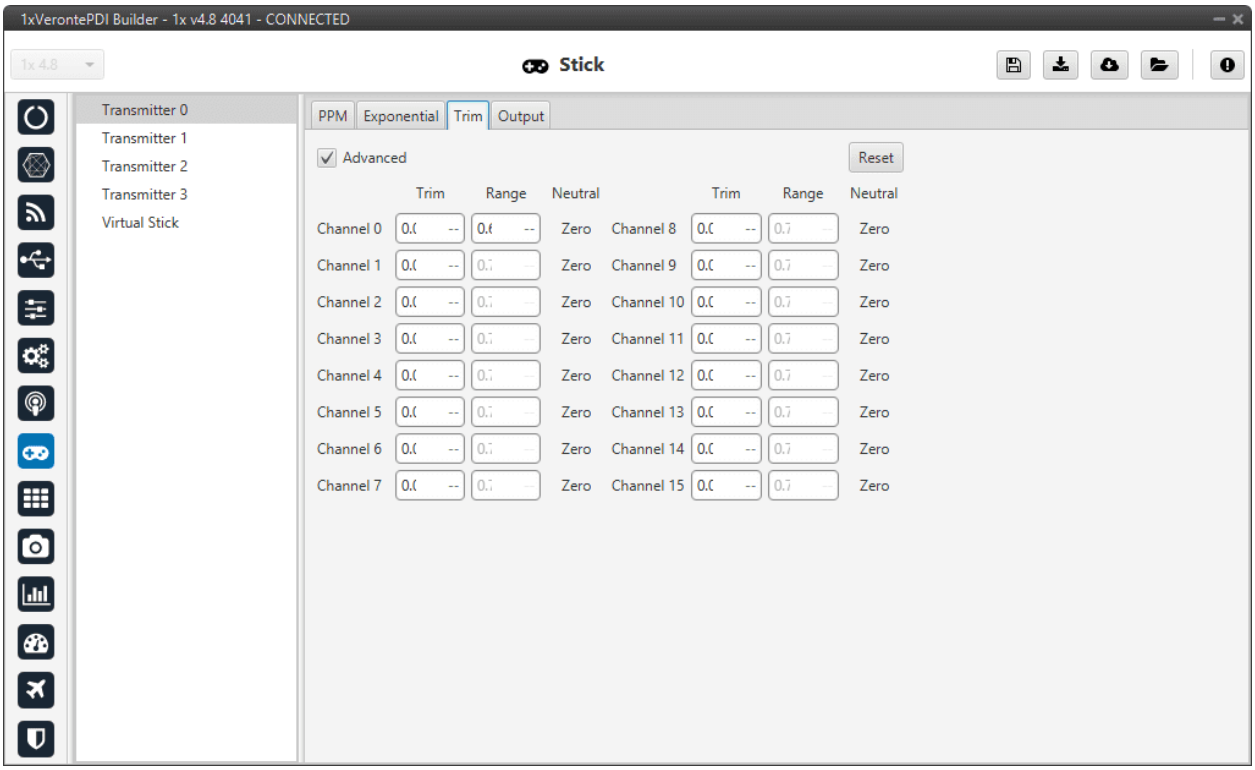


Fig. 218: **Trim panel**

2.8.1.4 Output

In this panel the user sets the receiving port and process the incoming commands. Once the stick has been configured, the commands that arrive at the ground autopilot have to be sent to the air unit.

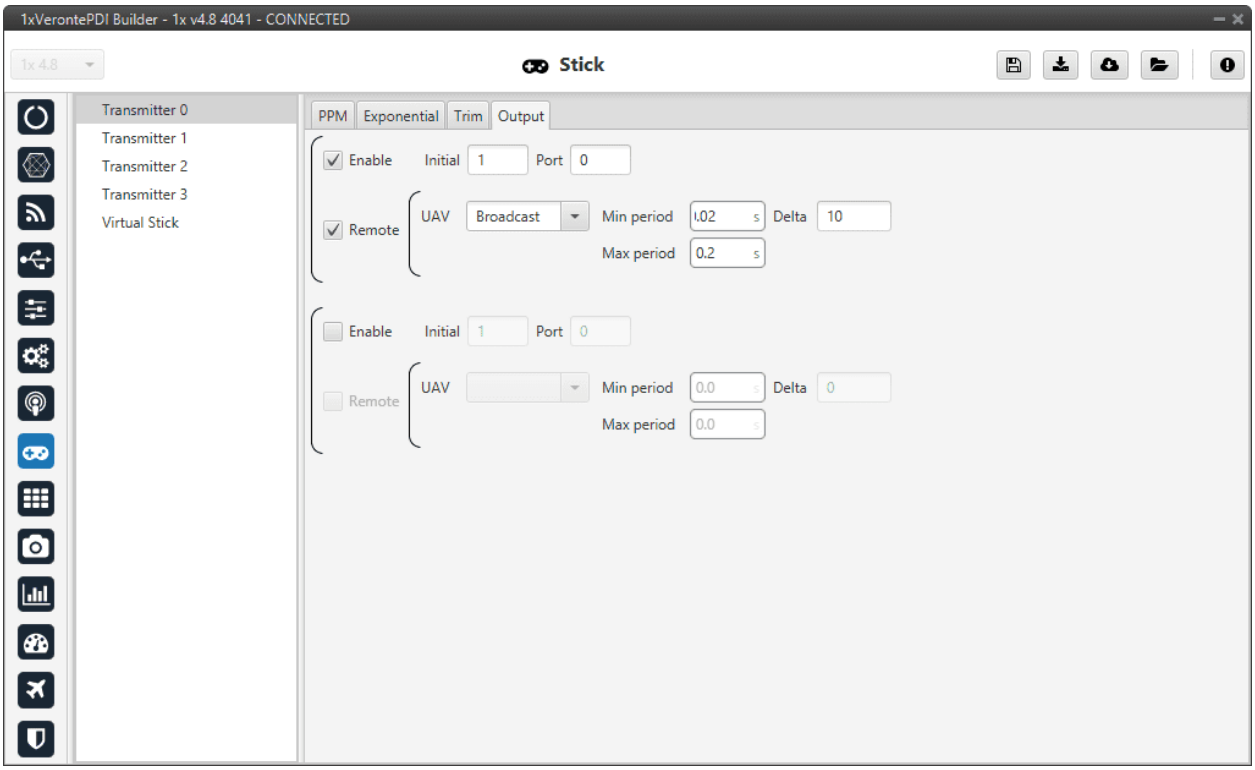


Fig. 219: **Output panel**

In this menu, the following parameters can be configured:

- **Enable**.
- **Initial Channel at destination**: The user indicates to which channel of the air autopilot will be sent the first channel received in the ground unit. The channels arrive at the platform in order and without spaces between them.

  For example, if at the GND channels 6,7,8,9 and 10 are enabled, the AIR will receive channels 1,2,3,4 and 5. Therefore channel 6 of the stick will be channel 1 in the AIR configuration.
- **Port**: If more than one transmitter is configured, each transmitter must be configured on a different port. This has to match the port set on the air unit.
- **Remote**: It has to be enabled if the user wants to allow the delivery of the commands to the platform.
  - **UAV**: The address of the UAV that receive the commands has to be indicated. The following options are the most common:
    * App 2: Veronte applications address.
    * Broadcast: The commands are sent to all units on the network. **This option is recommended**.
    * 1x v4.X XXXX: The address of a specific air unit.

  For more information on the available addresses, see List of addresses section of the **1x Software Manual**.

- **Min period**: As the period is the inverse of the frequency, this is the **maximum frequency**. Therefore, to give the pilot more control, this is the frequency that is set when the **stick is commanding**. A *Min period* of **0.02s** is recommended.
- **Max period**: As the period is the inverse of the frequency, this is the **minimum frequency**. Thus, to free up bandwidth, this is the frequency that is set when the **stick is idle**. A *Max period* of **0.2s** is recommended.
- **Delta**: This parameter determines whether the frequency is set to the minimum or maximum period set above.

  If Autopilot 1x detects a **change above** the **delta value**, the frequency goes to the **maximum frequency** (minimum period). While if the **changes are less than this value**, it switches to the **minimum frequency** (maximum period). **10 Hz** are recommended.

Note: An example of the *Stick integration* can be found in the **Integration examples** section of this manual.

2.8.2 Virtual Stick

This menu enables the processing of variables as stick control inputs, allowing reading stick signals distinct from PPM.

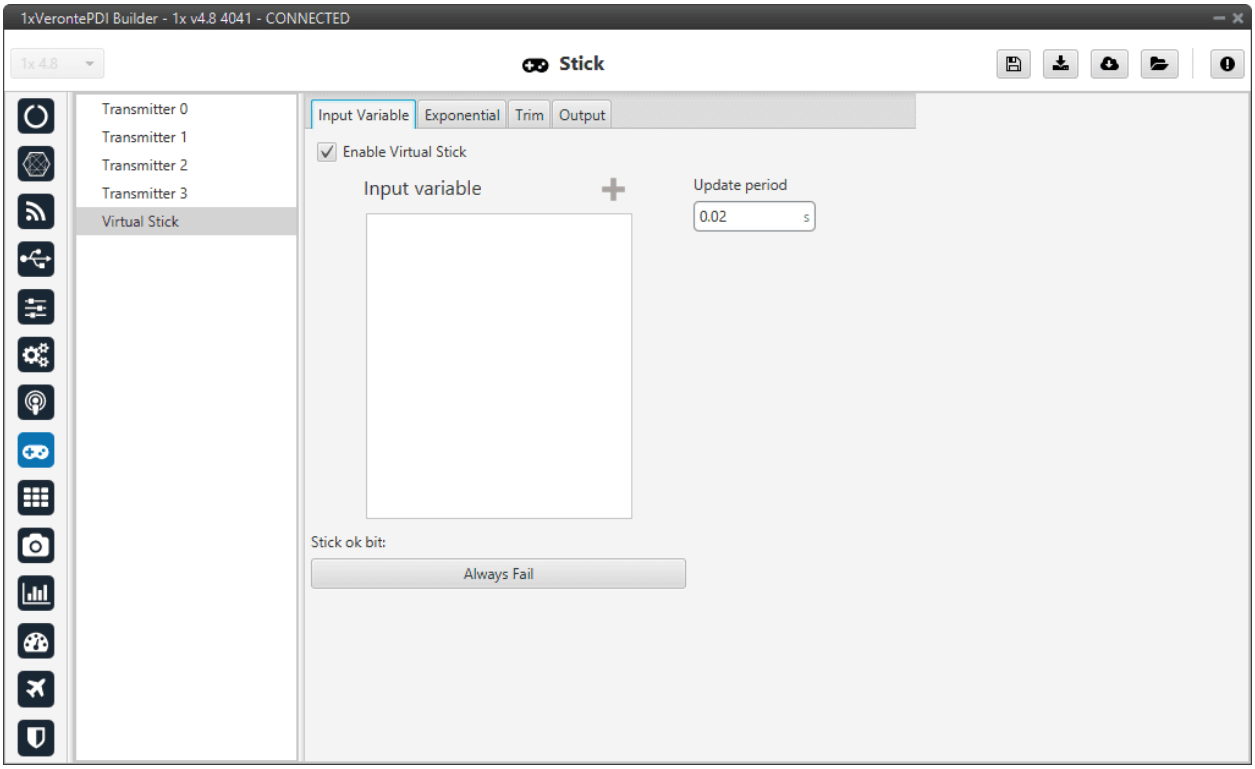


Fig. 220: **Virtual Stick - Input Variable panel**

In this panel the user can configure:

- **Enable Virtual Stick**
- **Input variable**: Place here the variables containing the stick information. A maximum of 16 variables can be added as *Input variables*.

Important: This menu assigns the introduced variables to the stick communication channels **in the entered order**, i.e.:

First variable added → **Channel 0**

Second variable added → **Channel 1**

Third variable added → **Channel 2**

- **Update period**: Configure the period required. A period of **0.02 s** is recommended.
- **Stick ok bit**: Select the bit which indicates if the virtual stick configuration is properly set.

The panels **Trim**, **Exponential** and **Output** are the same as the Transmitter ones, so refer to the *Transmitter* section for more information.

Note: An example of the *Virtual stick integration* can be found in the **Integration examples** section of this manual.

2.9 Block Programs

Block programs are the **core of Autopilot 1x**. In this menu, all **flight control algorithms** can be found, divided in different independent programs with different functions. All programs are executed at GNC (Guidance-Navigation-Control) time.

A **Program** is a custom algorithm executed by Autopilot 1x. While their main purpose is the control of the aircraft, Programs can be used to **develop a wide variety of applications**, from simple math operations to complex estimation filters.

Block programs provide the user with a **block programming interface** that 1x will then **execute at core frequency**. The fact that it is designed in this way gives it a high versatility, unlimited freedom and mimo control capabilities. This high versatility is thanks to this block programming interface, as they are **easily manageable** and **highly customisable blocks**, so that each customer can perfectly define their control algorithms regardless of the vehicle and the target they have.

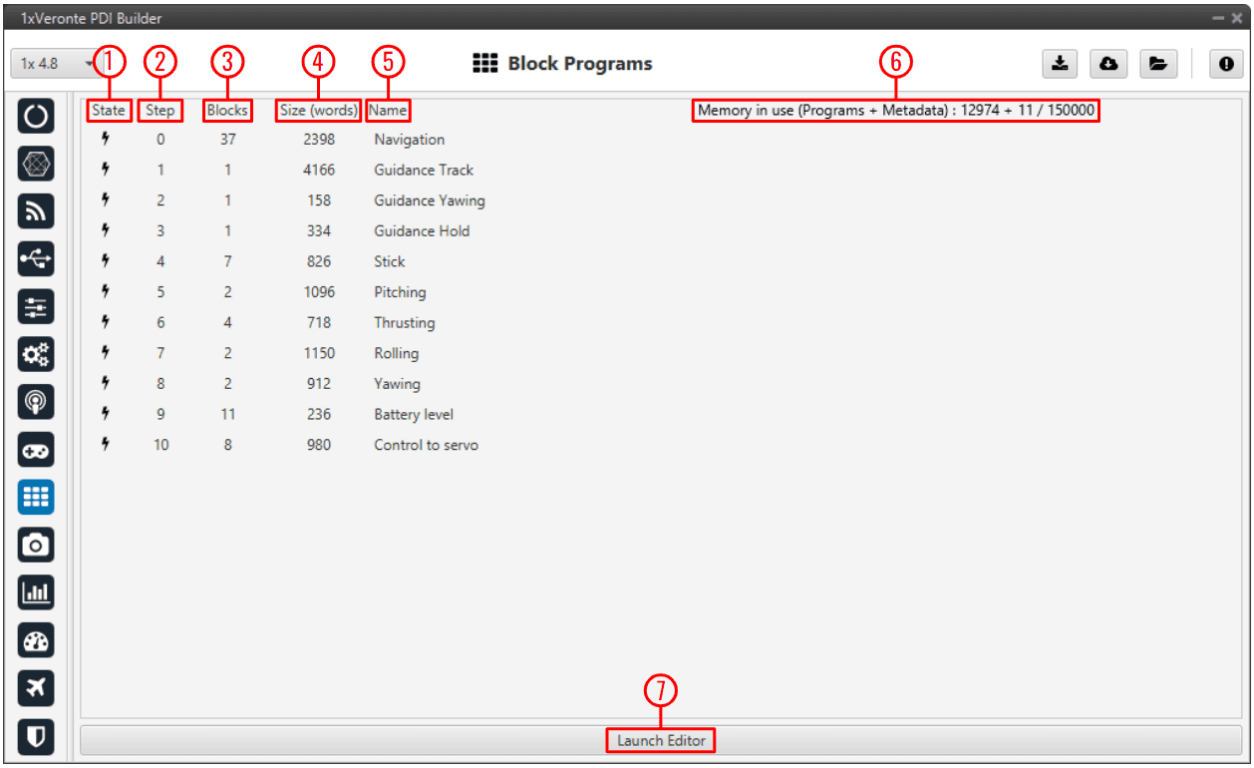


Fig. 221: **Block Programs menu**

1. **State**: There are two types of programs, those with a black lightning symbol and those with a grey lightning symbol.

   The black ones are programs that are executed periodically, **at core frequency**. While those with grey lightning are only "active" when they are executed using an automation (*Run Block Program*, see *Actions - Automations* section of this manual).

2. **Step**: This number determines the order of execution of the programs.
3. **Blocks**: This indicates the number of blocks in each program.
4. **Size (words)**: This is the memory taken up by each program.
5. **Name**: Program name, it is set by the user.
6. **Memory in use**: It is the operation performed to calculate memory in use.
7. **Launch Editor**: Click here to start configuring a program. A new window will appear:

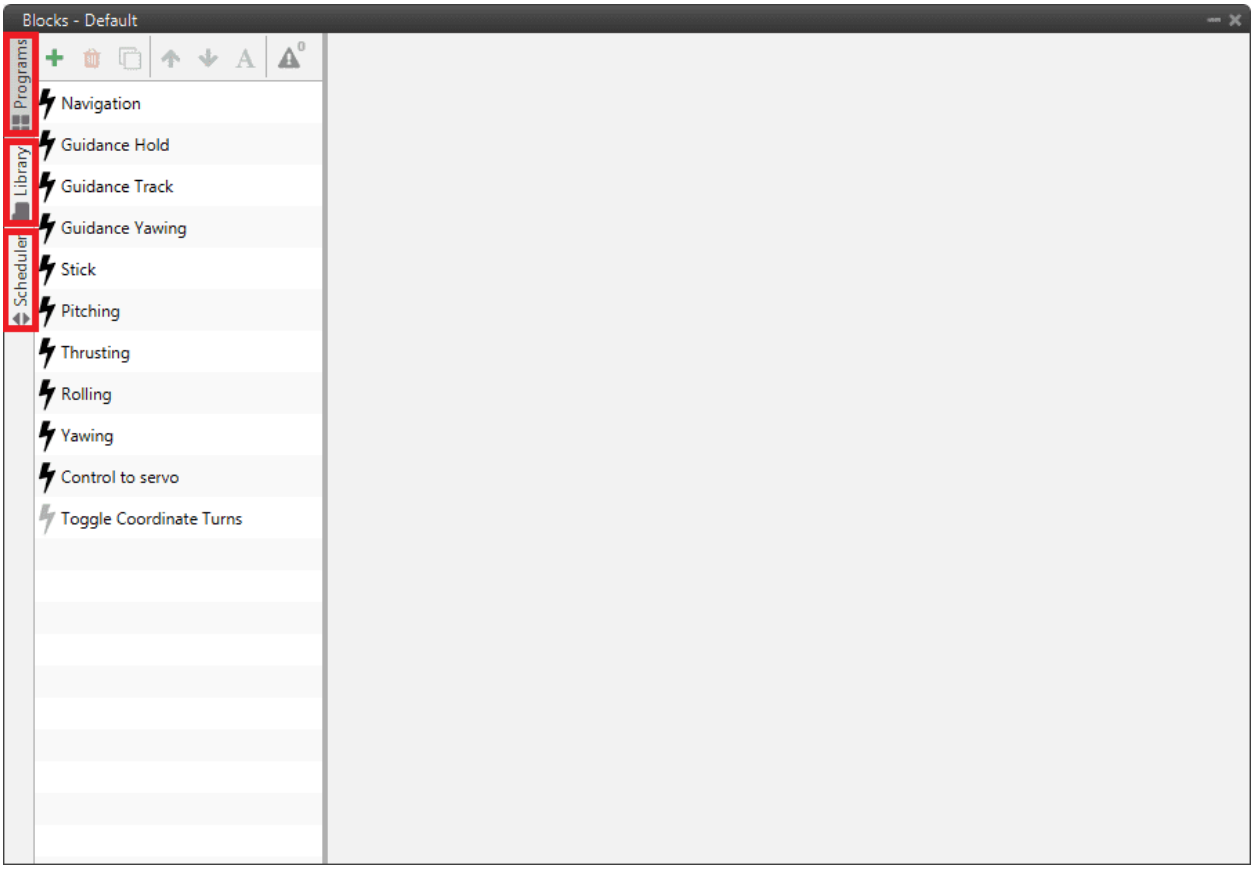


Fig. 222: **Block Programs tabs**

- **Scheduler**: In this tab users can configure the **frequency** (in Hz) at which **each program is executed**.

  By default the GNC frequency is defined for all programs, however, the user can modify it by pressing the '**+**' and '**-**' buttons next to the frequency.

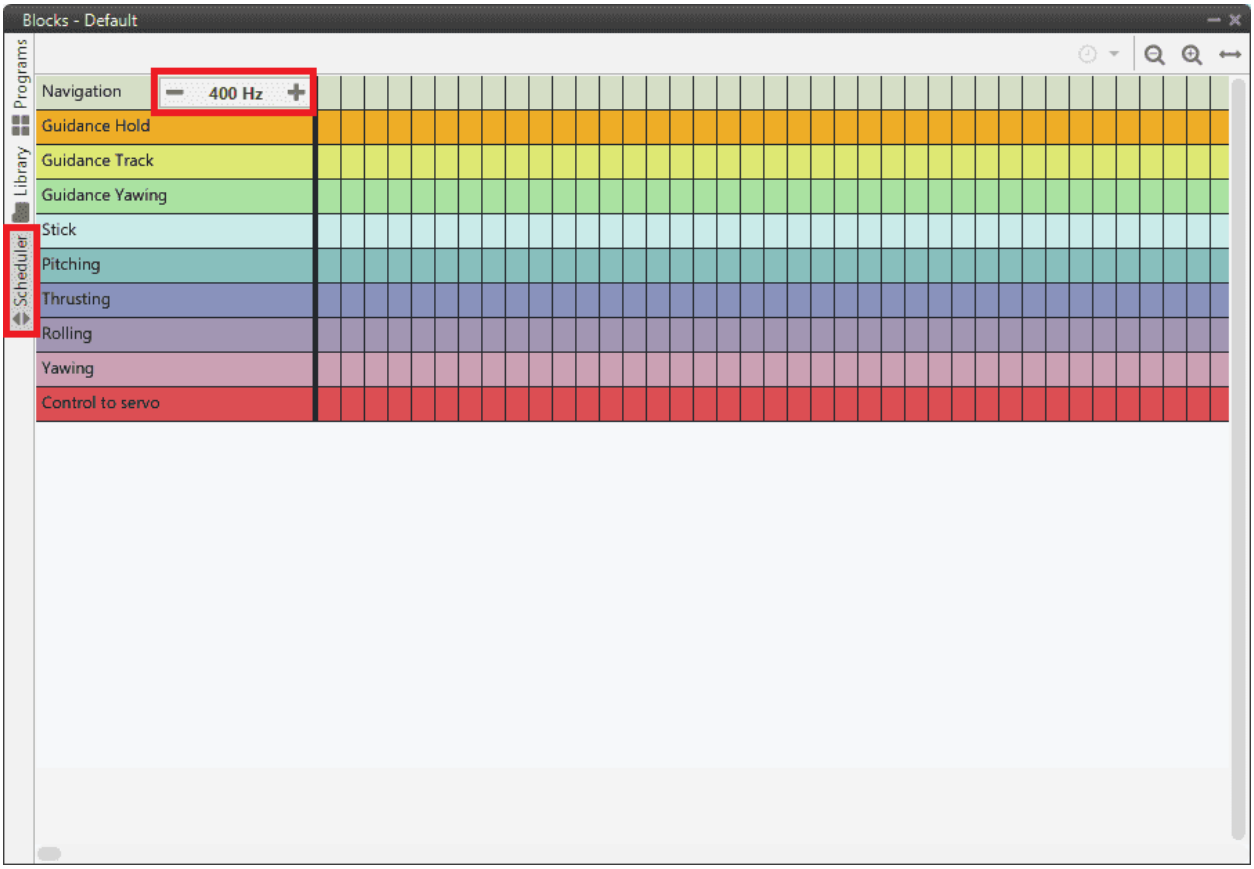


Fig. 223: **Block Programs menu - Scheduler**

If the user reduces the frequency of a program, it is possible to move it so that the the programs run in different slots:

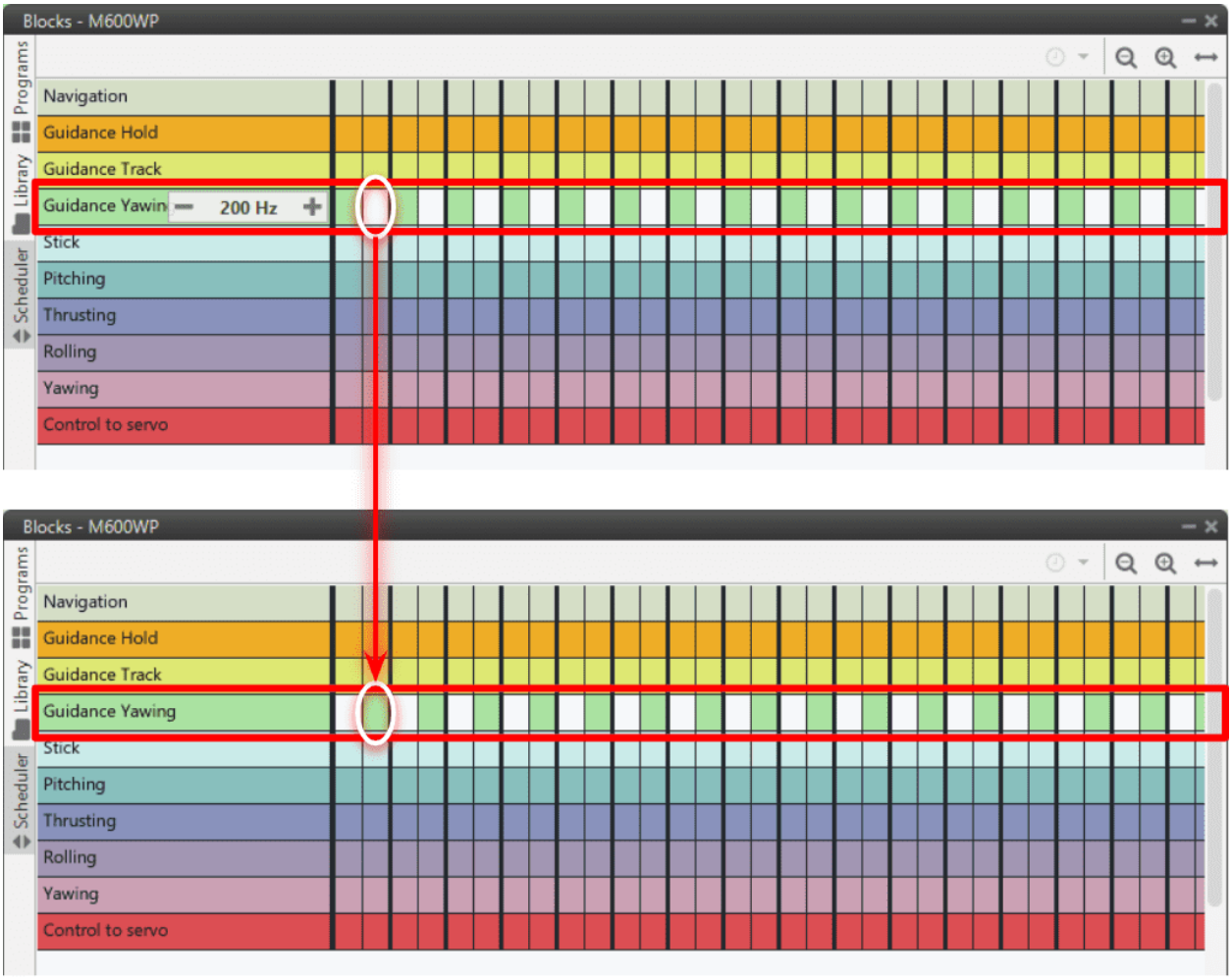


Fig. 224: **Block Programs menu - Scheduler slots**

Important: If the program is **not executed periodically** (grey lightning symbol), it will **not** appear in the **Scheduler** tab.

- **Library**: Here the user can create **custom blocks**. This tab is explained in more detail in *Library blocks* section.
- **Programs**: All Block Programs are created and configured in this tab:

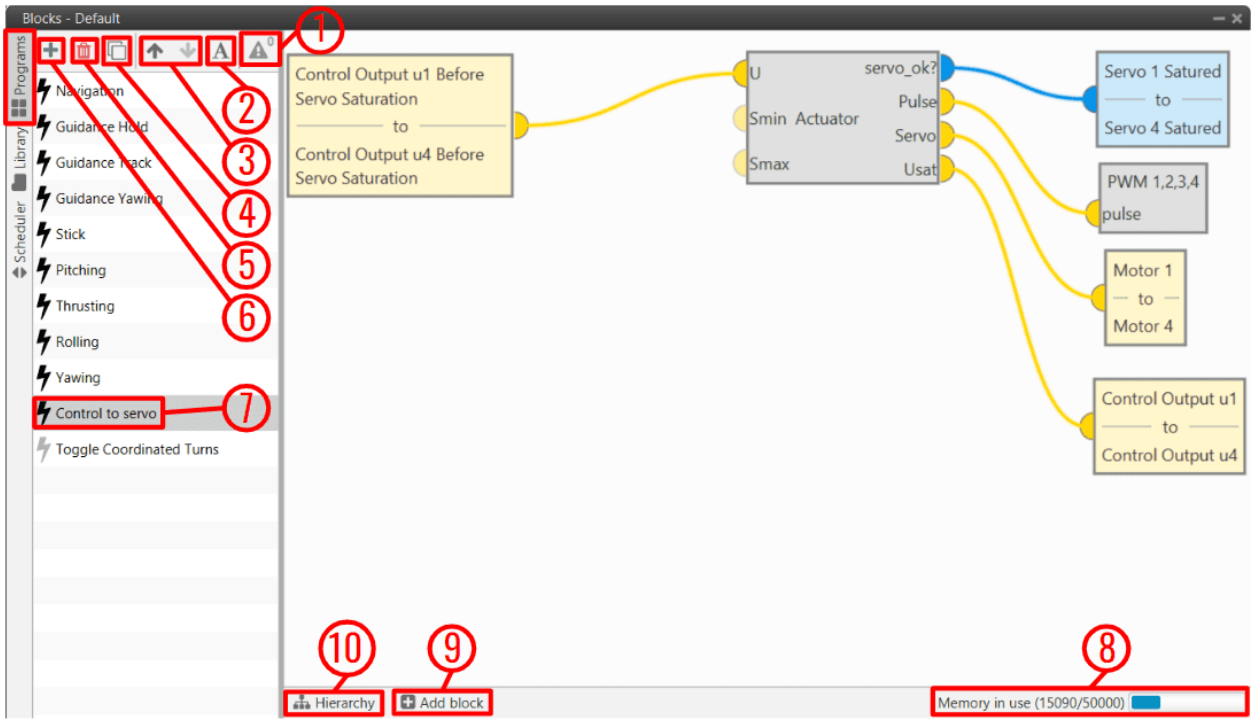


Fig. 225: **Block Programs menu - Programs options**

A. **Move to error** ⚠: When there are errors or warnings, the number of errors/warnings is displayed here and by clicking on them, **1x PDI Builder** takes the user to the program with that error.

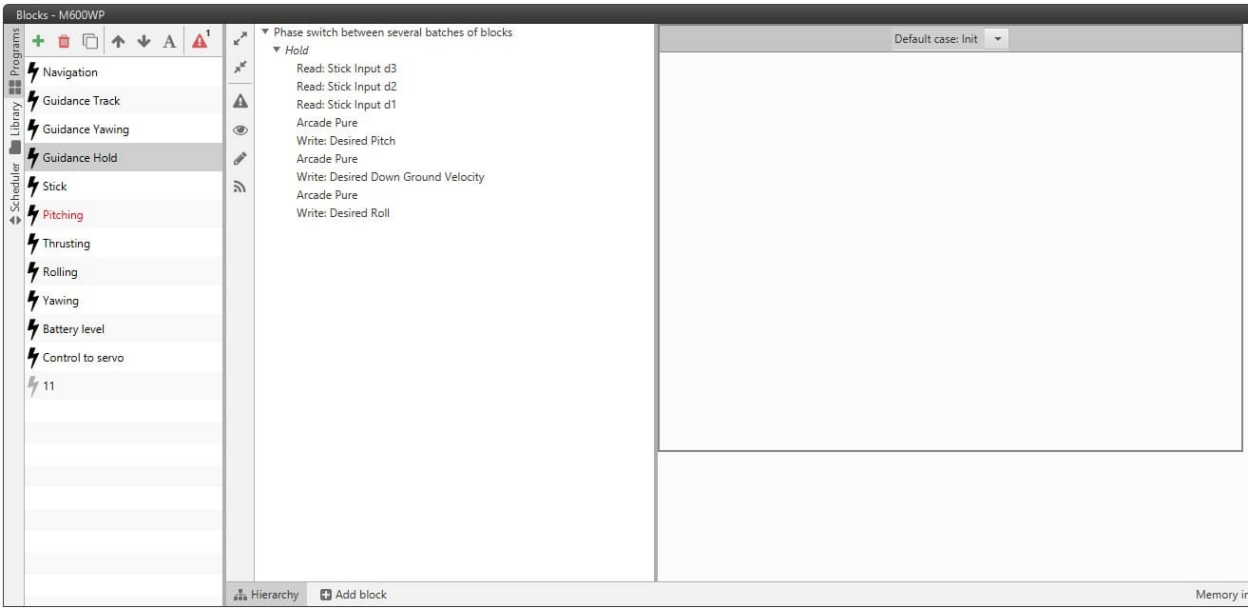


Fig. 226: **Block Programs menu - Move to error**

B. **Change name** A: Rename a selected program.

C. **Move up/down**  : Use them to determine the order of execution of a selected program. Programs are **executed from top to bottom**.

D. **Copy program**  : Copy a selected program.

E. **Remove program**  : Remove a selected program.

F. **Add program**  : Add a new empty block program.

G. **State** and **name** of program. Clicking on  icon will toggle the execution mode.

H. **Memory in use**: Estimation of the remaining memory available. If no more memory is available, no new blocks will be allowed to be created. The allocated memory for each block depends on the block type.

Attention:

- User must be aware that each block has its own size, so **the larger the size of a block, the more space it will take up**.
- Thus, the more programs are created, the more space is occupied.
- In addition, there is information stored as **metadata** about the organization and position of each block in the diagram that also represents part of this space.

Tip: To optimize memory, it is better to use more but smaller blocks than one large block.

I. **Add block**: By clicking here, a new column panel will appear where users can choose the block they want to add.

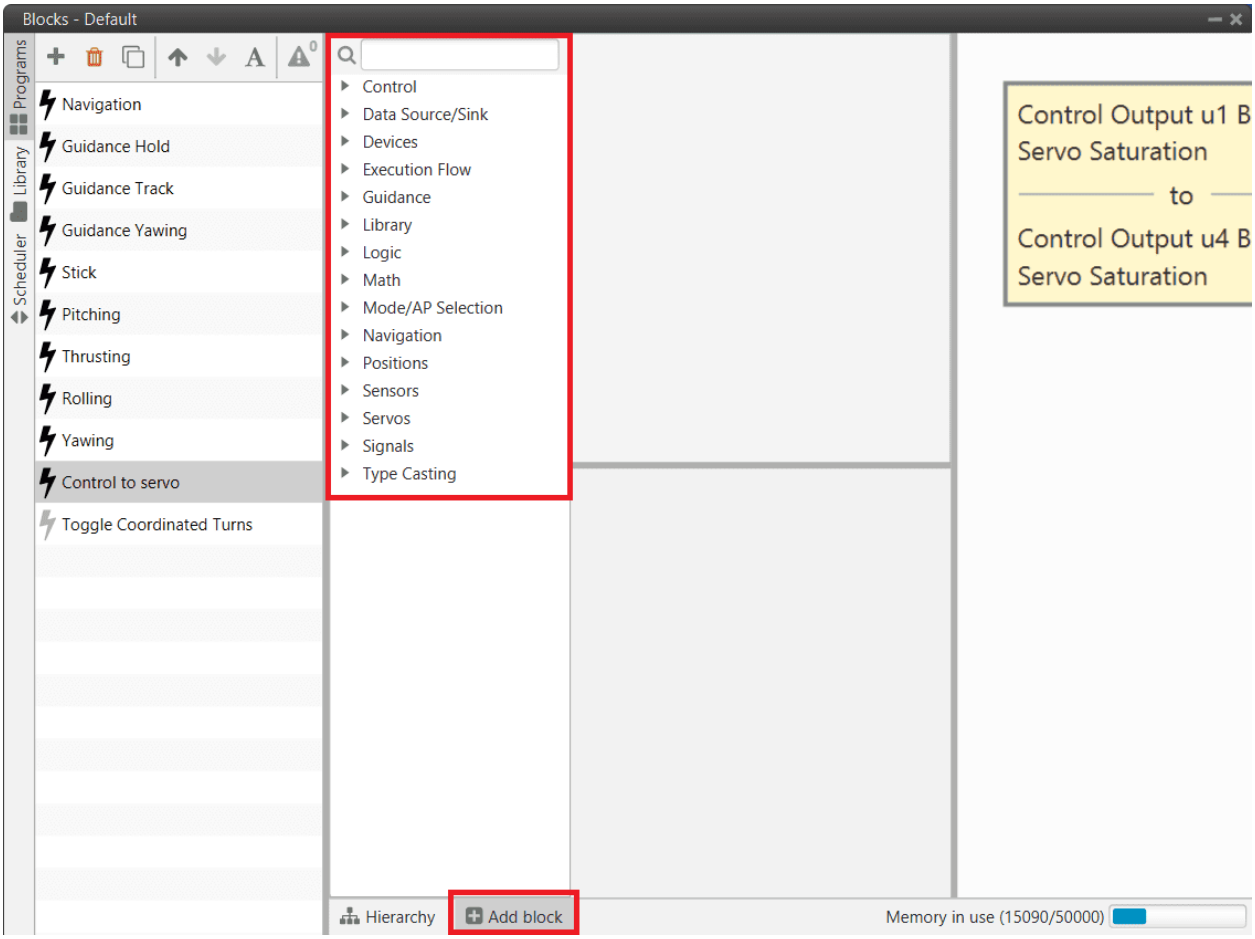


Fig. 227: **Block Programs menu - Add block panel**

J. **Hierarchy**: By clicking here, a new column menu will appear where users can see "information" about the existing blocks on the selected program.

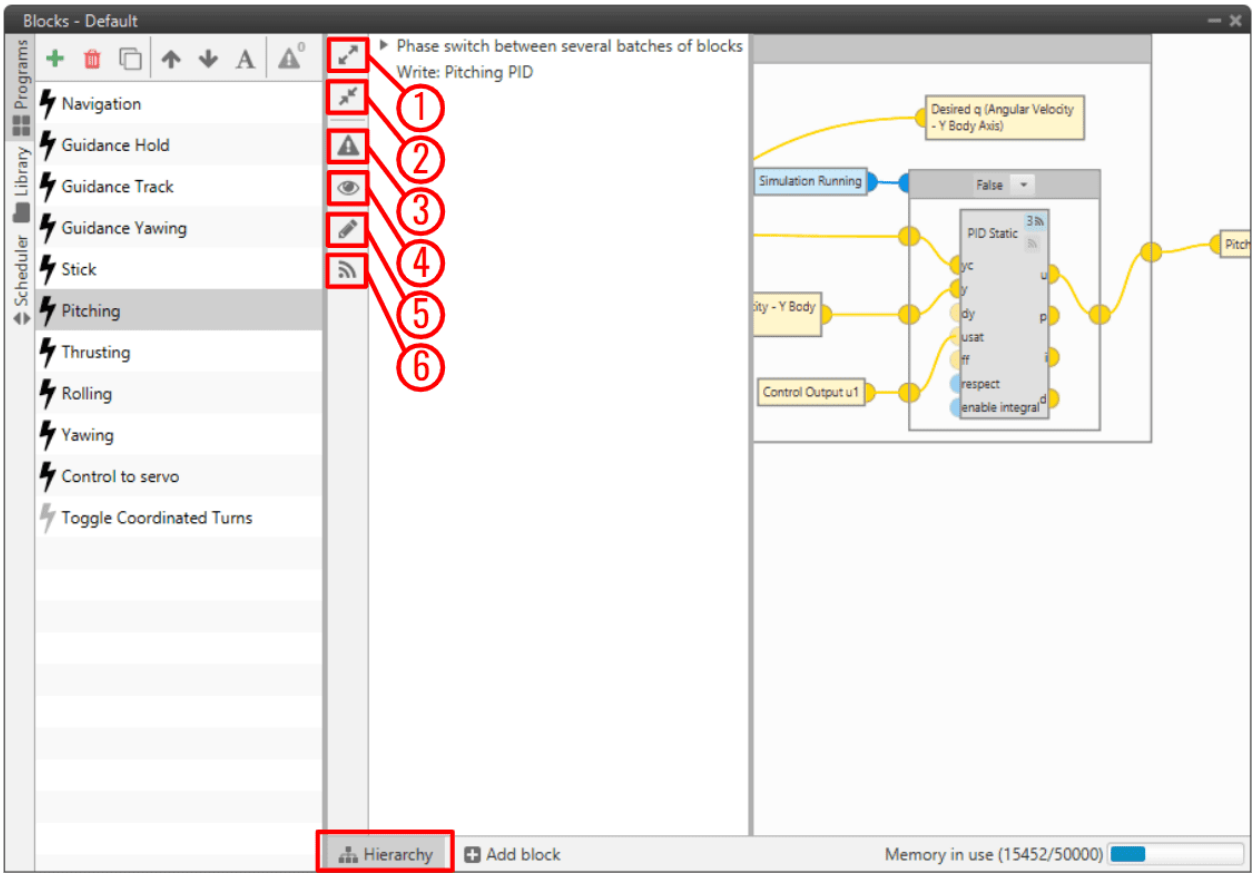


Fig. 228: **Block Programs menu - Hierarchy panel**

a. **Expand All**: By clicking on it, all menus that are collapsed will be expanded, e.g. in programs that have **Switch** blocks.

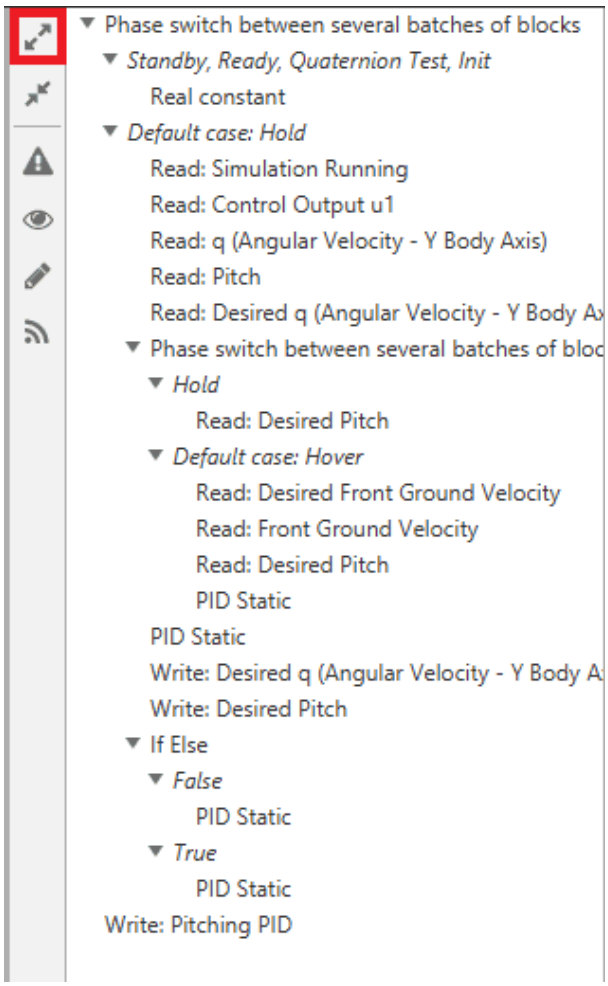


Fig. 229: **Expand All**

b. **Collapse All**: By clicking on it, all menus that are expanded will be collapsed, e.g. in programs that have **Switch** blocks.

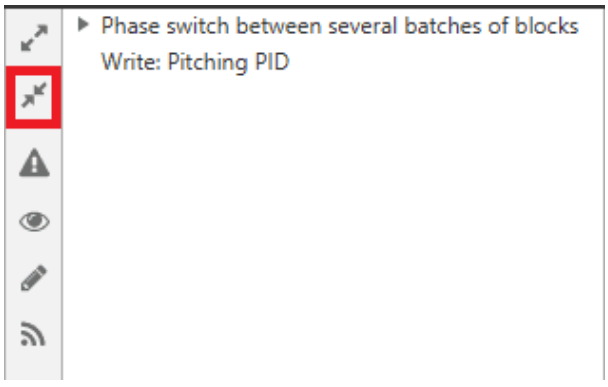


Fig. 230: **Collapse All**

c. **Show blocks with errors**:

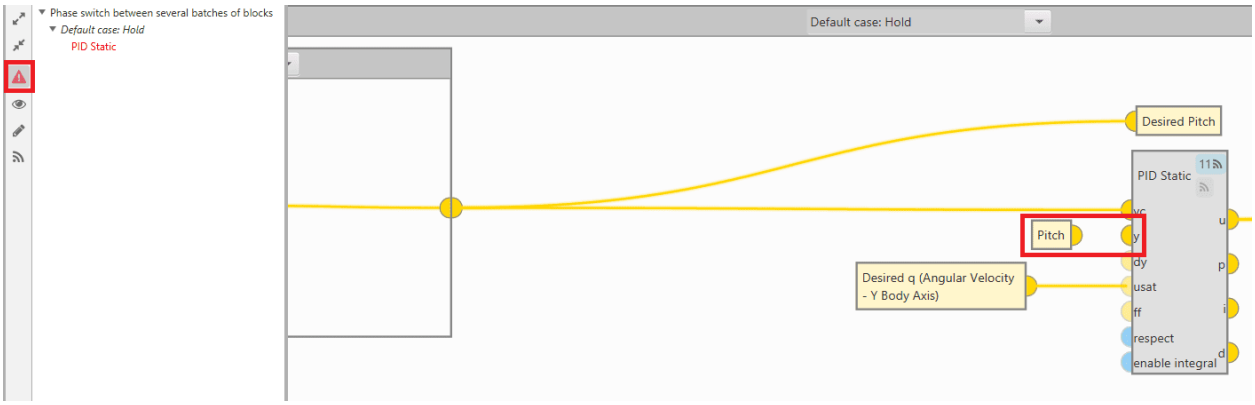


Fig. 231: **Show blocks with errors**

d. **Show blocks that read from variables**:

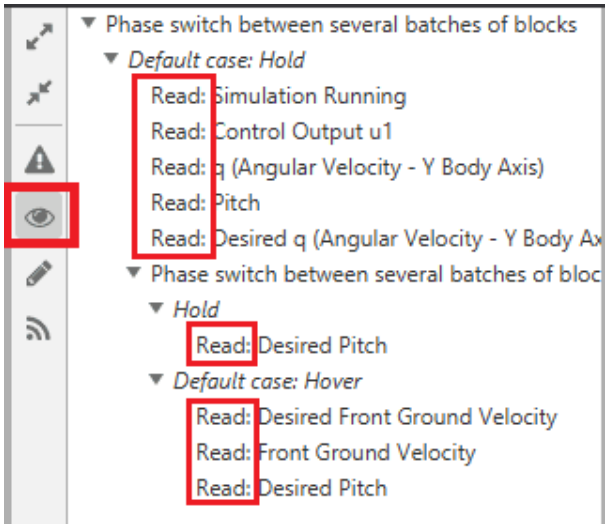


Fig. 232: **Show blocks that read from variables**

e. **Show blocks that write in variables**:

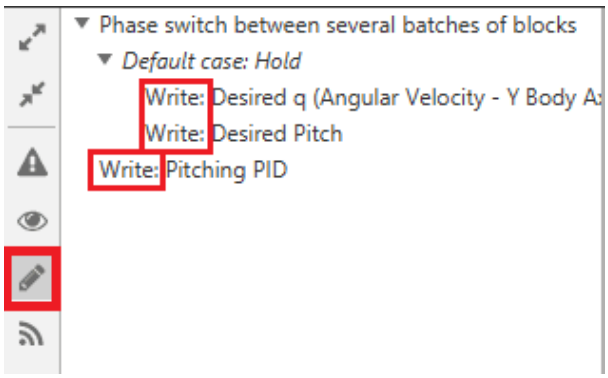


Fig. 233: **Show blocks that write in variables**

f. **Show blocks that can be commanded**:

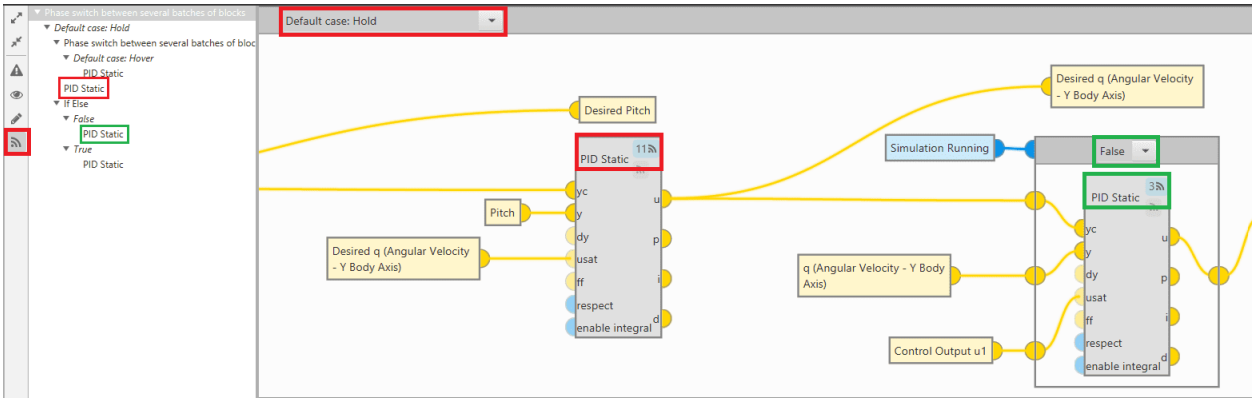


Fig. 234: **Show blocks that can be commanded**

Blocks

- To **add a block**:
  1. Click on '**Add Block**'.
  2. Search and **select a block**. When a block is selected, its description is displayed.
  3. **Click and drag** it to import into the program.
- By **right clicking** on a block, the user can:
  - **Edit**: This options opens its configuration menu. It is also possible to open the configuration by *double clicking* on the block.
  - **Remove block**: Remove the selected block.
  - **Copy**: Copy the selected block.
  - **Paste**: Paste the block that has been copied.
  - **Change constant value/variable/variables**: User can change the constant value entered/the selected variable/s, e.g. to be read or written.

    Note: Only available for *Data Source/Sink blocks*.

- To **re-locate blocks**, just *click and drag* them.

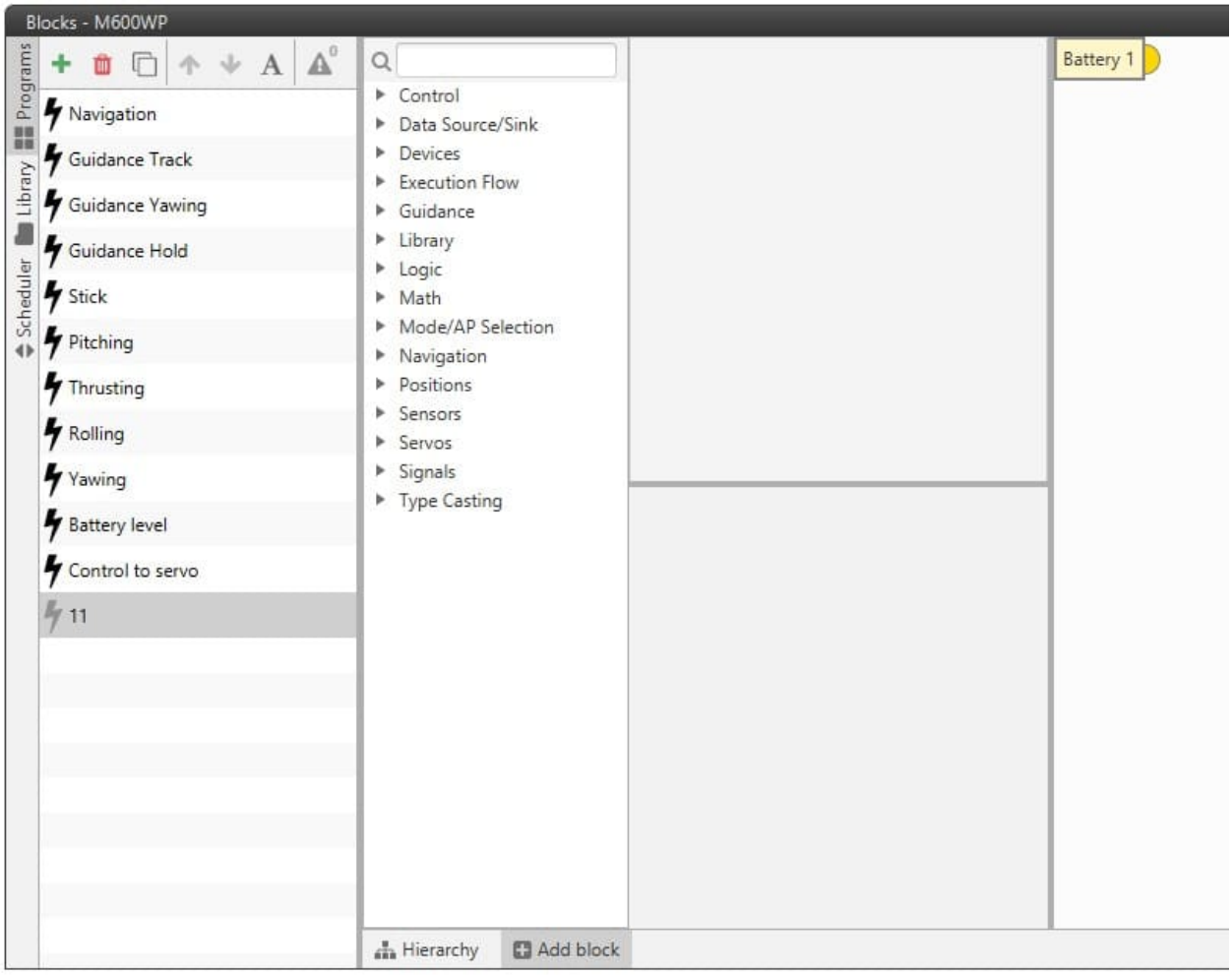


Fig. 235: **Block Programs menu - Adding blocks**

Block inputs and outputs use a color code in order to indicate variable types:

- : 32-bit Real variables.
- : BIT Boolean variables.
- : 64-bit Feature variables.
- : 16-bit Integer variables.
- : Position measurement data.
- : Guidance data.
- : Sensor data for EKF.

Note: Connectors can also be **Arrays** of variables.

An input and an output can be **linked** directly with the *mouse* and **unlinked** by *right clicking* on the input/output:

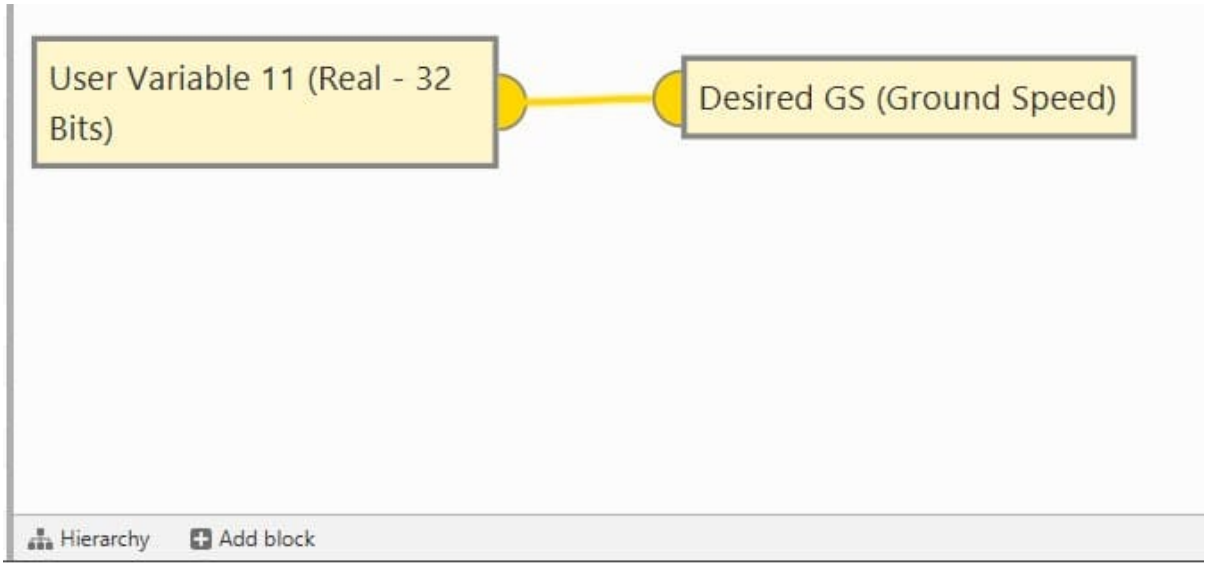


Fig. 236: **Linking blocks**

Note:

- An input and output with **different variable types** cannot be linked without a **Type Cast block**. For more information on this block, see *Type Casting blocks* section of this manual.

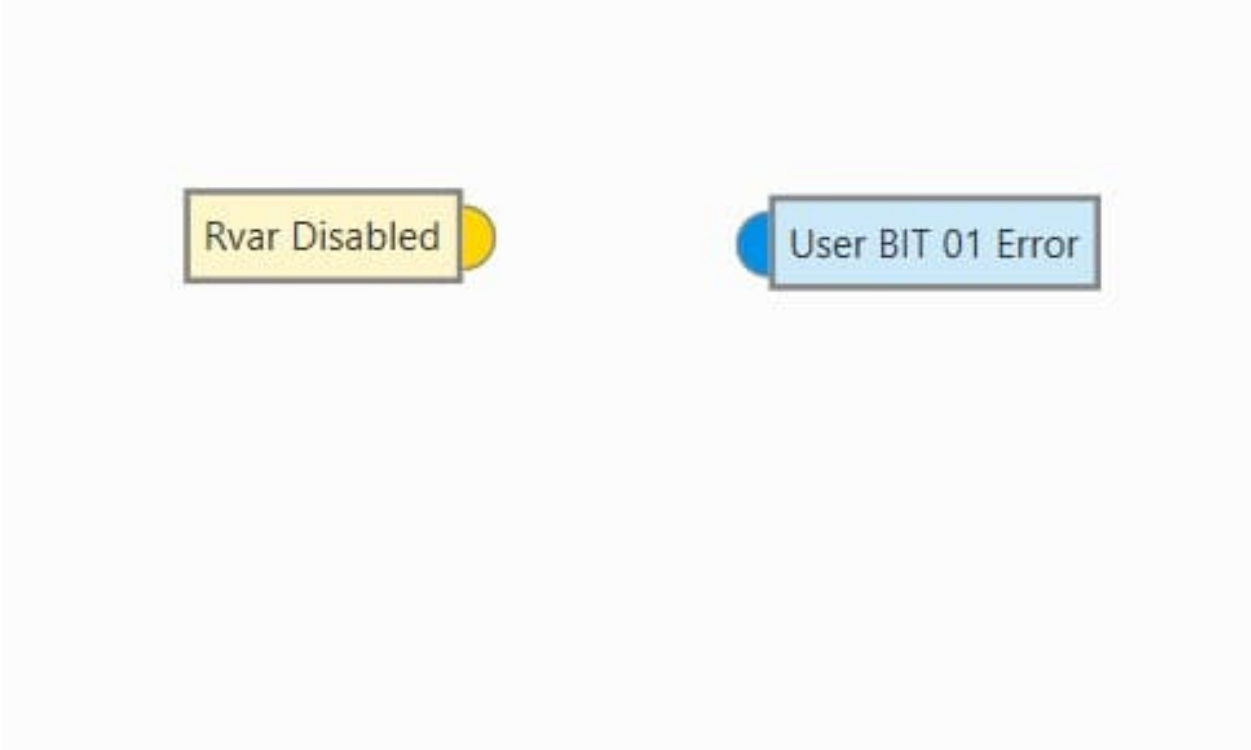


Fig. 237: **Type Casting block**

- All **inputs** of each block **must be connected**, otherwise **1x PDI Builder** will report an **error**. **Outputs do not need to be linked**.

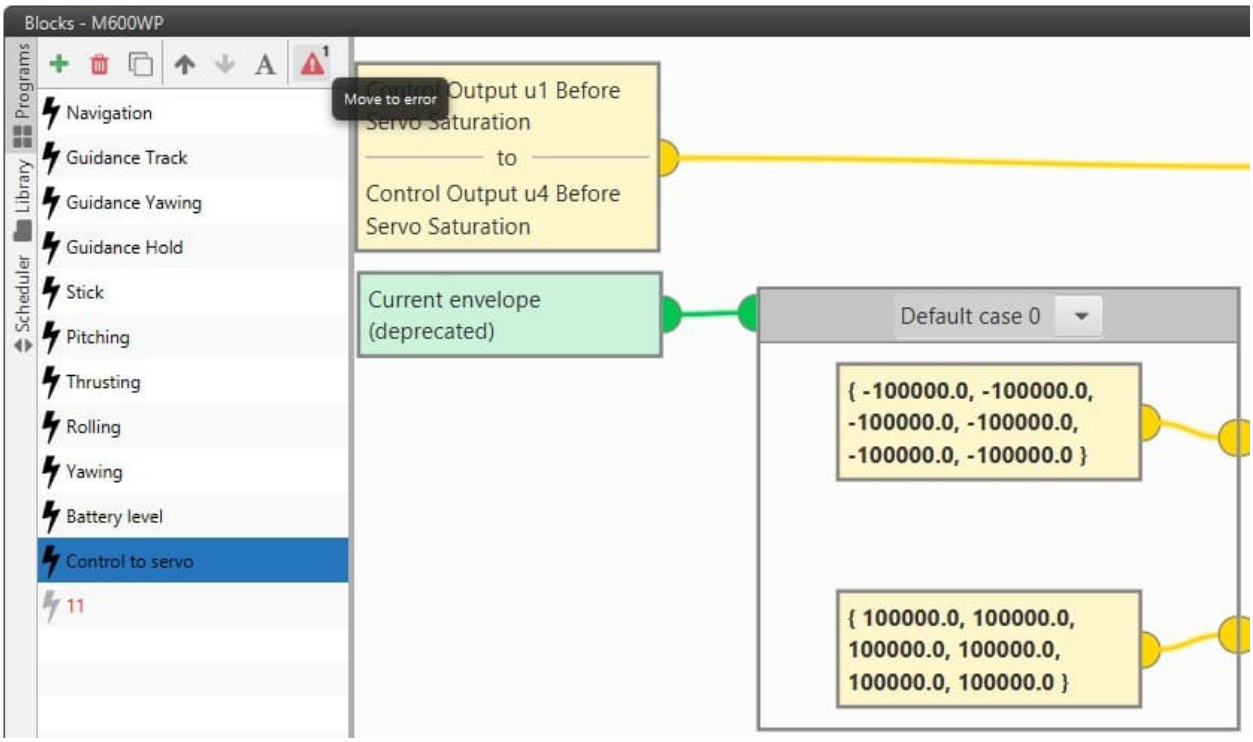


Fig. 238: **Errors in blocks**

- An exeception of the previous rule are translucid inputs  , which are optional. These inputs will have a default value if not linked.

The different types of blocks available in **Block programs** are:

- *Control*: Control-related blocks (**PID Static**, Tsched PID, ECU control etc.).
- *Data Source/Sink*: Input/Output blocks. Programs can have access to any variable available within Autopilot 1x system. Results can then be stored in **User Variables** for display, use as a different program input, feedback, etc.
- *Devices*: These blocks allow to configure **devices connected to the autopilot**.
- *Execution Flow*: Programming-like blocks for operation flow control. These blocks allow to **alter parts of a program depending on a condition** (If-Else, Integer Case, Phase Case, etc.).
- *Guidance*: In these blocks the **guidance of the flight phases** is configured.
- *Library*: Blocks **created manually** combining already defined blocks. They are created in the '**Library**' tab which can be accessed on the left hand side of this menu.
- *Logic*: **Logical gates** to operate with boolean variables (AND, NOT and OR).
- *Math*: **Mathematical blocks**, which include a variety of mathematical operators: **basic** (sum, multiply, square root, etc.), **trigonometric** (sine, cosine, tangent, etc.), **vectors** (norm, dot product, rotations, etc.).
- *Mode/AP Selection*: Blocks that allow to interact with **flight modes** and redundancy (Autopilot 4x).

- *Navigation*: These blocks allow the **autopilot navigation** to be configured.
- *Positions*: Blocks for **operating with position-type variables** (create position, read position, relative position, etc.).
- *Sensors*: In these blocks, some of the **sensors are configured**.
- *Servos*: Blocks related to **servos configuration**.
- *Signals*: Blocks for **signal processing** (IIR filter, rate limiter, etc.).
- *Type Casting*: Blocks for variable **conversion** (Real to BIT, Integer to real, etc.).

2.9.1 Control blocks

Control blocks are those related to the creation of control loops.

2.9.1.1 PID

PID Static block allows the user to build a PID (Proportional, Integral and Derivative) controller with fixed gains.

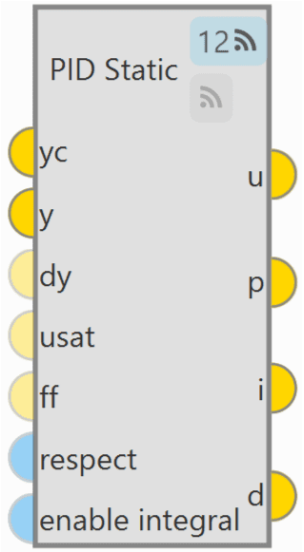


Fig. 239: **PID block**

As can be seen in the figure above, PID Static block has 2  buttons that enable or disable the block to be commanded from the 1x PDI Tuning software.

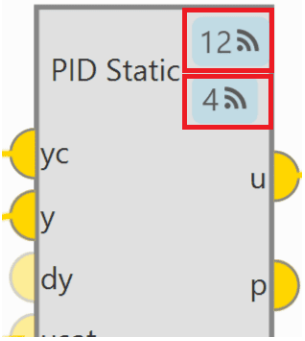


Fig. 240: **PID block - Command buttons**

The first button is for activating 'Command PID' and the second one is for the 'Autotune' command. For more information on these commands, please refer to the Tuning section of the **1x PDI Tuning** user manual.

Each command button has a **different ID** that allows the user to identify it during the command.

Note: To avoid disabling a block by mistake, the following warning message appears when disabling it:

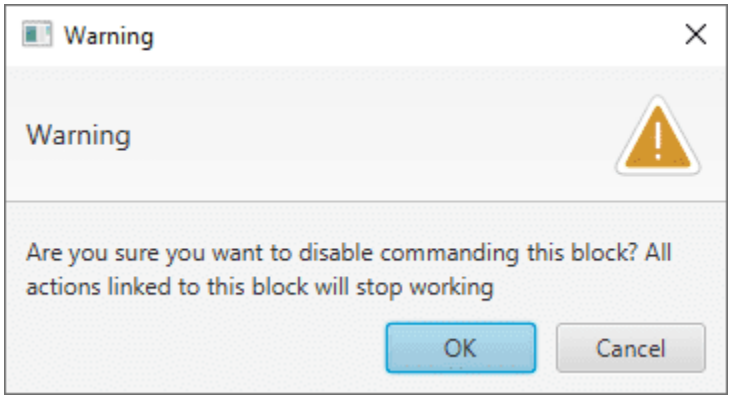


Fig. 241: **Warning message when disabling the command**

- The PID mathematical implementation in Veronte Autopilot 1x is the following:

$$C = Kp + \frac{1}{T_i} \cdot IF(z) + \frac{Td}{\tau + DF(z)}$$

Where:

$$DF(z) = T_s \cdot \frac{z}{z-1} \quad ; \quad IF(z) = \frac{T_s}{2} \cdot \frac{z+1}{z-1} \quad ; \quad LPF(s) = \frac{1}{\tau s + 1}$$

- **Inputs**
  - **yc**: Target value, desired set-point of the controlled variable.
  - **y**: Closed loop, value of the controlled variable.
  - (Optional) **dy**: Derivative of the controlled variable (computed numerically from 'y' if not connected).

- (Optional) **usat**: Previously applied control action after saturation (used for anti-windup and respect). If not connected a value of zero is assumed.
- (Optional) **ff**: Feed-forward control, this value is added to the 'u' output before applying the output limits. If not connected a value of zero is assumed.
- (Optional) **respect**: When TRUE the output 'u' is equal to the input 'usat' and the integral component is estimated with the information in 'y' and 'yc'. When FALSE the PID works as usual. If not connected a value of FALSE is assumed.
- (Optional) **enable integral**: When TRUE the the PID works as usual. When FALSE the integral is exponentially discharged. If not connected a value of TRUE is assumed.

- **Outputs**
  - **u**: Control output after applying PID limits.
  - **p**: Proportional part of the output before the PID limits are applied.
  - **i**: Integral part of the output before the PID limits are applied.
  - **d**: Derivative part of the output before the PID limits are applied.
- **Configuration menu**:

  Double click on the block to open its configuration menu.

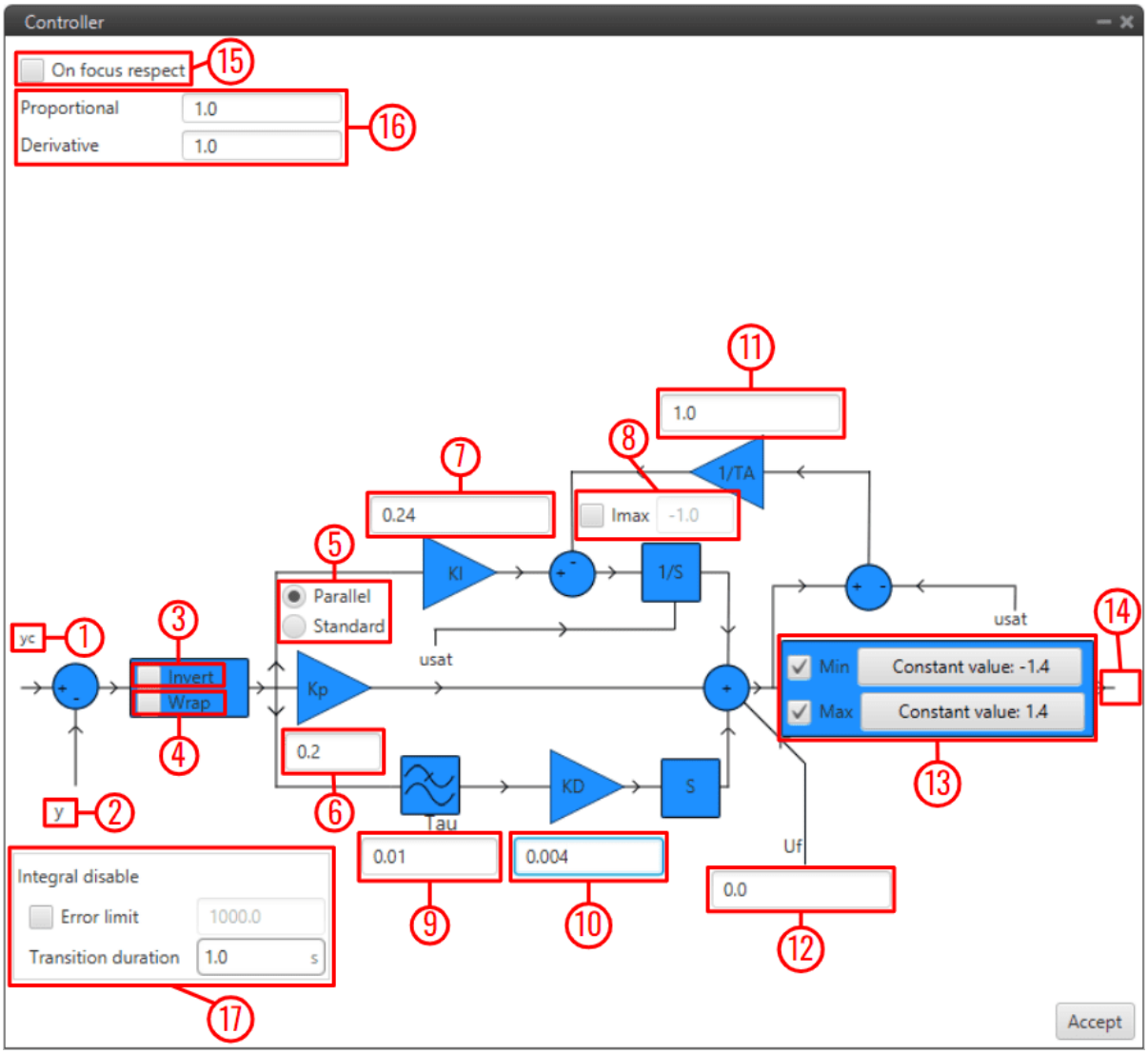


Fig. 242: **PID block configuration**

1. **yc**: Input variable.
2. **y**: Feedback variable.
3. **Invert**: Apply a -1 gain .
4. **Wrap**: Perform a [-pi, pi] wrap.
5. **Parallel/Standard**: In *parallel* mode, PID gains are independent. In *standard* mode, I & D gains are scaled by P gain.
6. **KP**: Proportional gain.
7. **1/TI**: Integral gain.
8. **Imax**: Maximum value for integral term. Value must be possitive and the limit applied is symmetrical ([-Imax, Imax]).
9. **tau**: Time constant for the derivative term first order LPF.
10. **TD**: Derivative gain.

11. **TA**: Anti-windup gain. Recommended value around x10 KI. Unloads integral term if output is saturated.
12. **Uf**: Output offset. **Feedforward** value is also applied at this point.
13. **Min/Max**: Output limits. Users can manually enter a value or select a variable.
14. **u**: PID output.
15. **On focus respect**: If respect is enabled, when the PID is first executed, an initial *I* value will be applied so that **'u' = 'usat'** for the first iteration.
16. **Proportional/Derivative Beta**: **yc** scaling for proportional and derivative terms. Unless necessary, value should always be **1**.
17. **Integral disable**: Disables integral term if **(yc - y) > Error limit**.

Tip: Remember to always use '**wrap**' for direction controllers, such as 'Heading' or 'Yaw' PIDs. This will allow the UAV to always turn in the right direction.

2.9.1.2 T-Sched PID

TSched PDI block is a PID (Proportional, Integral and Derivative) controller with table scheduled parameters. It allows to scale most PID parameters using an external variable, usually the speed (Ground speed or IAS).

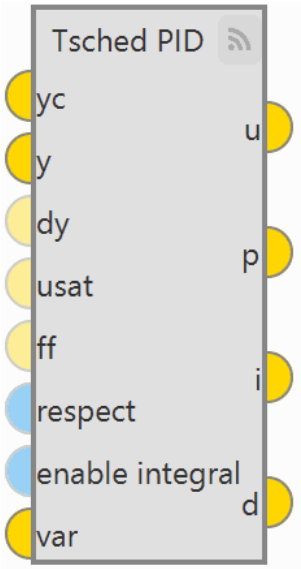


Fig. 243: **TSched PID block**

It works in a very similar way to the PID Static block, except that in that block the gains are fixed and in the TSched block they are adjusted for different values of the input variable.

For this reason, the inputs and outputs are the same, but in this block an additional input is added:

- **Input**

  var: Scaling variable for gain scheduling used to interpolate in the table to obtain the PID parameters.

- **Configuration menu**

  Double click on the block to open its configuration menu.

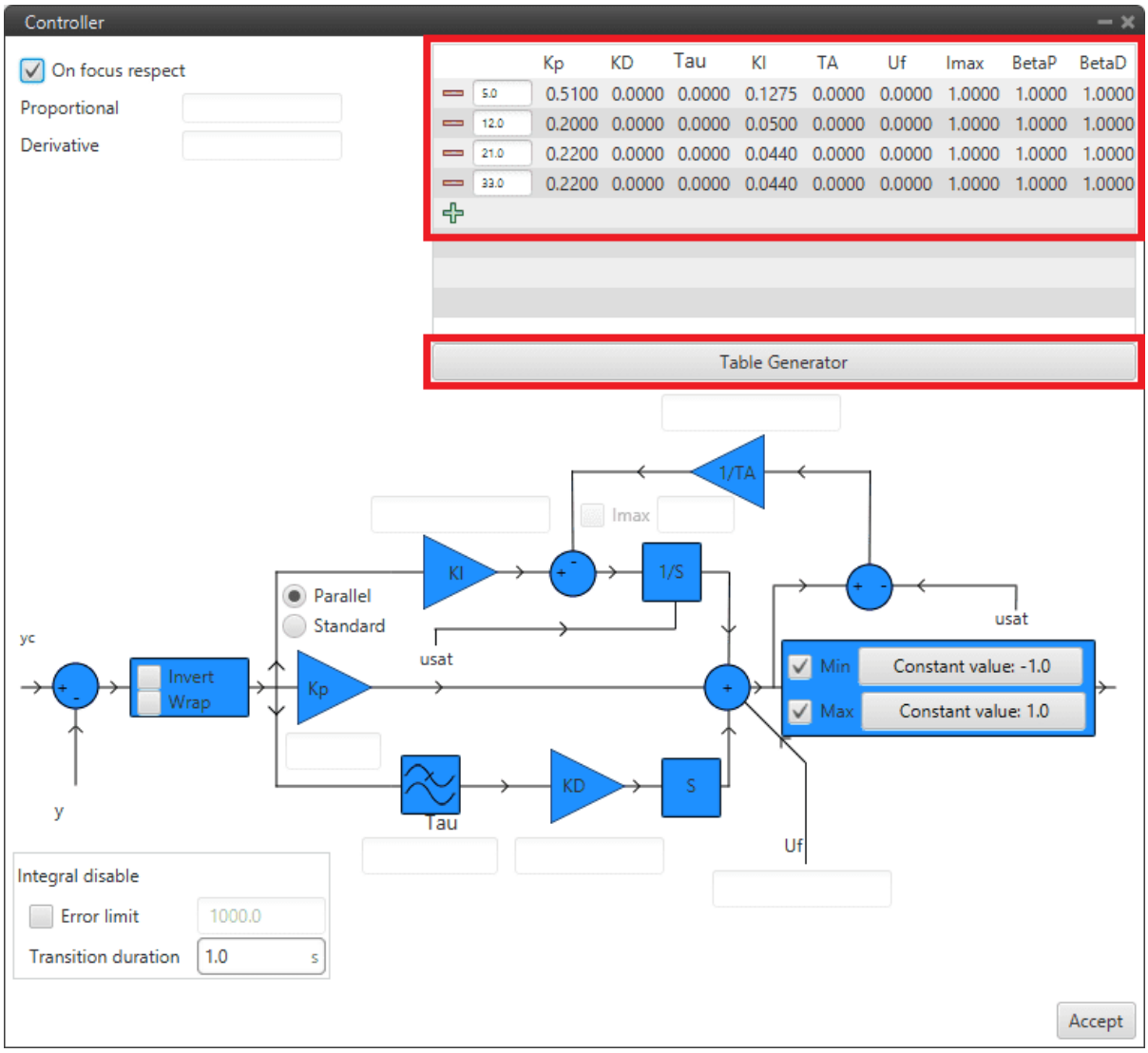


Fig. 244: **TSched PID block configuration**

In this block, the PID gains for the different values of the input variable must be entered in the table above instead of in the diagram. To add more values, simply click on + and to remove them, click on −.

If the variable is outside the limits, the values of the closest point will be applied. Values between points are **linearly interpolated**.

In addition, clicking on "Table Generator" will bring up another configuration menu:

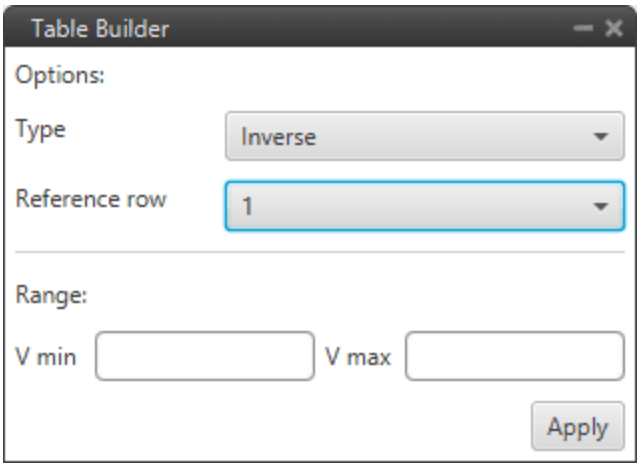


Fig. 245: **TSched PID block configuration - Table Generator**

This is another option to enter the PID gains for different values of the input variable, instead of manually entering all the gains, the **Table Generator** will generate them according to the parameters to be configured:

- **Type**: Depending on the type of function selected, the gains are calculated differently.
  - **Inverse**: It will calculate the different gains with an inverse function.
  - **Proportional**: It will calculate the different gains with a proportional function.
  - **Quadratic**: It will calculate the different gains with a quadratic function.

Type	Result (K_p)
Inverse	$K_{p_i} \frac{V}{V_i}$
Proportional	$K_{p_i} \frac{V_i}{V}$
Quadratic	$K_{p_i} \frac{V_i^2}{V^2}$

  - **Reference cow**: Users must select the reference values from which the other values will be calculated.
  - **Range**: The minimum and maximum values of the input variable between which the gains have to be calculated have to be defined.

Then just click on "**Apply**" and the maximum number of points that the table allows will be generated.

Example

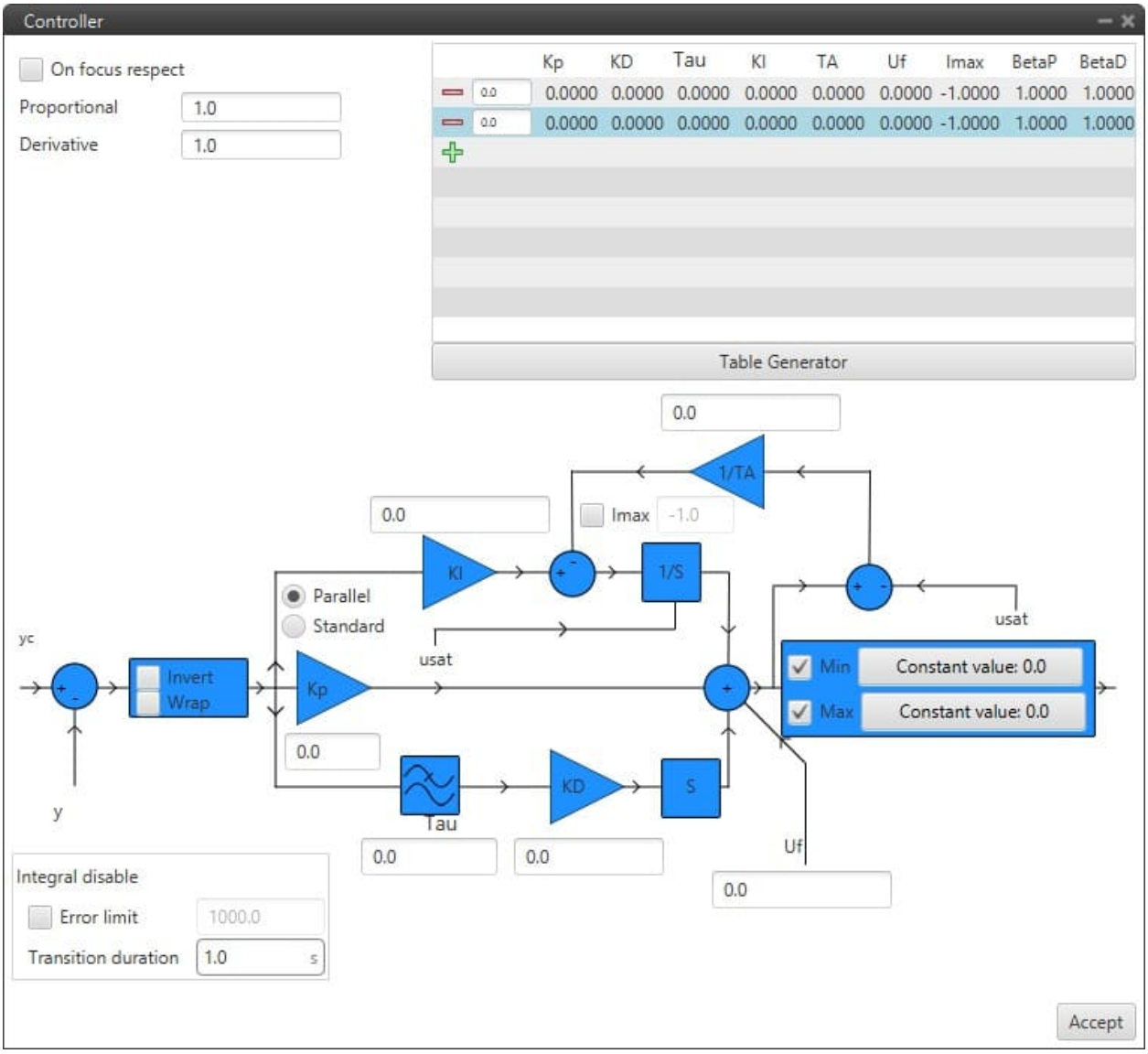


Fig. 246: **Example of Table Generator**

2.9.1.3 ECU Control

The **ECU Control** block allows to control the winding speed of the microjets and to ensure safe motor operation based on PID control and shaft speed for the microjets.

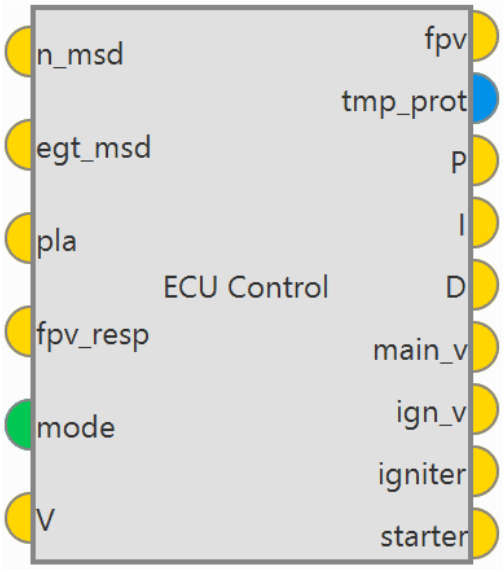


Fig. 247: **ECU Control block**

- The ECU Control mathematical algorithm is **based on a PID scheduler**, this PID control has been used because the dynamics of the motor changes with RPM so it is very different at low and high speed. For more information on PID Scheduler control, see *T-Sched PID* section of this manual.
  - First, the PLA magnitude is converted in commanded speed based in a look-up table. This information must be provided by engine producer.
  - To protect from a common problem of engines the commanded speed is limited. For this, first, the maximum speed is limited to the configurable parameter N_{max} to ensure mechanical integrity.

    Then, engine acceleration is limited to protect from compressor surge, so the maximum speed rate is limited to a configurable parameter in block, $\dot{N}_{max}$.

    Finally, engine deceleration is limited to protect from blow out, so the minimum speed rate is limited to a configurable parameter in block, $\dot{N}_{min}$.
  - If **EGT is less than the maximum value** (configurable), the error magnitude in speed is minimized with a PID scheduler: $e = y - yc = N - N(PLA)$.

    And the control output of this block is the FPV commanded to engine.
  - If **EGT measurement is higher than maximum temperature** (configurable EGT_{max}), a protection protocol is initiated and the fuel injected is zero.

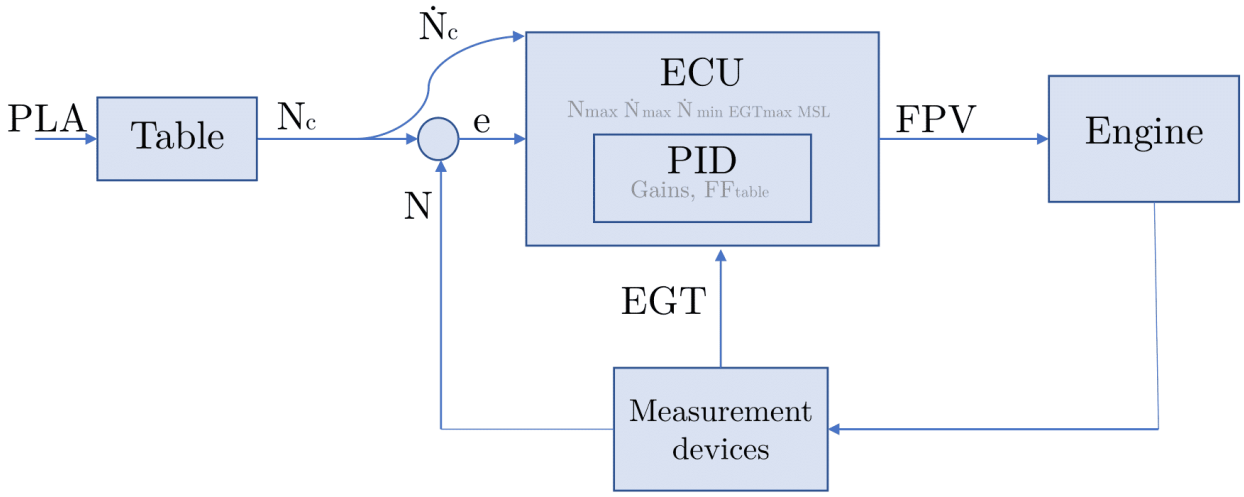


Fig. 248: **ECU Control aglorithm**

- **Inputs**
  - **n_msd**: Measured Speed from sensor.
  - **egt_msd**: Exit Gas Temperature from sensor.
  - **pla**: Power Level Angle demanded from pilot (value from sensor).
  - **fpv_resp**: Fuel Pump Voltage to do respect.
  - **mode**: Mode of execution:

  0 ⇒ Off: all variables set to zero.

  1 ⇒ Checking: starter engine is commanded to max to check if engine is okay.

  2 ⇒ Starting: pilot has total control with PLA once engine is runnning.
  - **V**: Voltage to Engine.
- **Outputs**
  - **fpv**: Fuel Pump Voltage to supply.
  - **tmp_prot**: Boolean to active temperature protection.
  - **P**: Proportional part of controller.
  - **I**: Integral part of controller.
  - **D**: Derivative part of controller.
  - **main_v**: Voltage to main valve.
  - **ign_v**: Voltage to Igniter valve.
  - **igniter**: Voltage to igniter.
  - **starter**: Voltage to starter engine.

- **Configuration menu**:

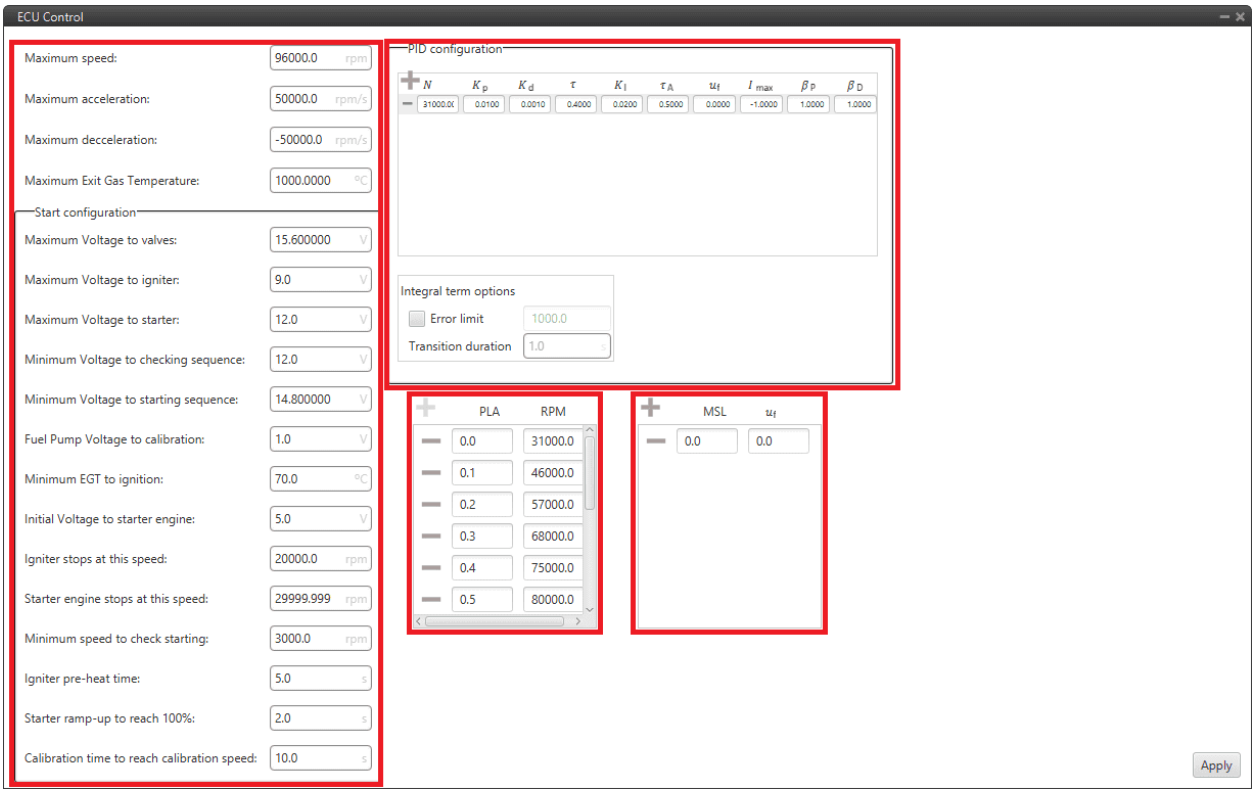


Fig. 249: **ECU Control block configuration**

  - All parameters to be configured in the left column of the panel configuration constitute the engine characterization and must be filled in with the user's engine specifications.
  - **PID configuration**: Users must configured the PID scheduler here. For more information on its configuration, check *T-Sched PID* section of this manual.
  - **PLA / RPM**: Users can enter the percentage of throtlle with respect to RPMs.
  - **MSL /** u_f: Because atmospheric pressure decreases with altitude (MSL), a feed forward (u_f) is configured as a function of altitude in order to improve control at high MSL.

2.9.1.4 Fuzzy Logic Controller

The **Fuzzy Logic Controller** (FLC) block implements the Fuzzy Logic algorithm allowing users to perform robust control of any system.

Fig. 250: **Fuzzy Logic Controller block**

A FLC has to be embedded in a **closed-loop control system**.

Plant output is designed by $u(t)$, its inputs are denoted by $y(t)$, and reference input to the FLC is denoted by $y_c(t)$. So, FLC will have $y(t)$ and $y_c(t)$ as controlled and commanded inputs, respectively, and $u(t)$ as control output.

- The FLC mathematical implementation in Veronte Autopilot 1x is the following:
  - The controller try to minimise the value of **error**, denoted by:

$$e_k = y_k - y_{ck}$$

  - And it gets the value of **change in error** (derived from the error) $ce(t)$ to do the *fuzzy set*:

$$ce_k = e_k - e_{k-1}$$

  - Once is defined error and change in error, its values have to be pass to fuzzy values scaling it with its gains k_e and k_{ce} :

$$en_k = k_e \cdot e_k$$
$$cen_k = k_{ce} \cdot ce_k$$

  - When these values are updated, the Membership Functions have to be defined as desired. Although their shapes could be any function (trapezoidal waveform, Gaussian waveform, etc.), they are usually **triangular waveform**.

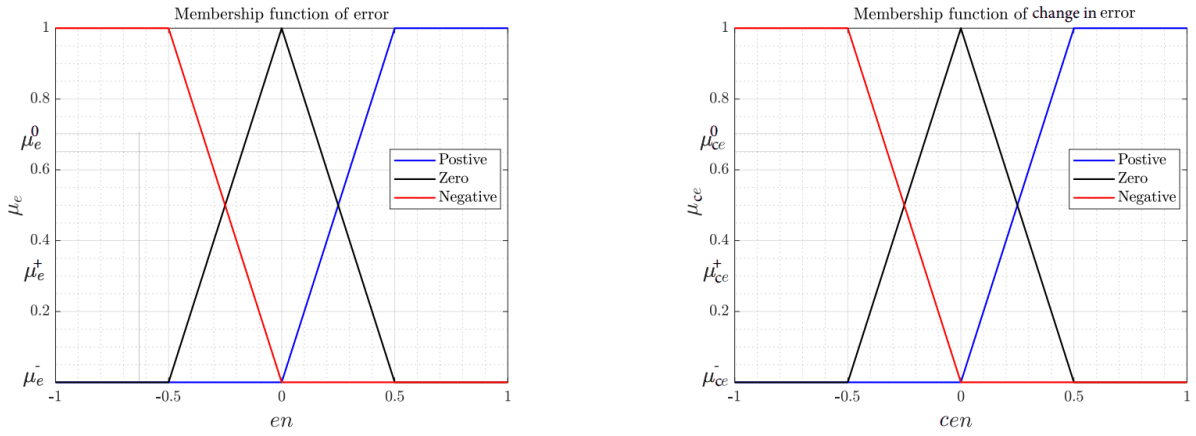


Fig. 251: **Example of membership functions of error and change in error**

- The exact values of the 2 last equations are used to get the weights $[\mu_e^+, \mu_e^0, \mu_e^-]$ and $[\mu_{ce}^+, \mu_{ce}^0, \mu_{ce}^-]$. These values are obtained from error and change in error membership functions.
- Then, those outputs must be arrange into a table, called the lookup table.

Δu		***cen***		
		+	**0**	**-**
en	**+**	+	+	0
	0	+	0	-
	-	0	-	-

Fig. 252: **Example of Fuzzy Logic look up table**

- On the other hand, it is necessary to apply the fuzzy logic by this table, so it is checked the sign of en_k and cen_k , and the result is the membership function of the output to apply Δu.

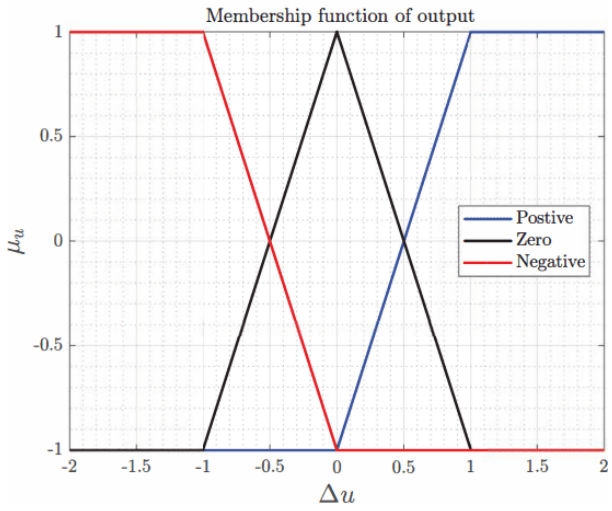


Fig. 253: **Example of membership function of output**

- Once the six values of weights $[\mu_e^+, \mu_e^0, \mu_e^-]$ and $[\mu_{ce}^+, \mu_{ce}^0, \mu_{ce}^-]$ are obtained, they must be combined in the nine possible combinations, selecting the minimum between both and getting its respective value of Δu in the output membership function.
- The final value of output is obtained with the Center of Gravity method:

$$\Delta u = \frac{\sum_{i=1}^{9} \Delta u_i \cdot \mu_i}{\sum_{i=1}^{9} \mu_i}$$

- Finally, the real output value must be integrated in time and converted from fuzzy variables to real variables with its gain value k_u :

$$u_k = u_{k-1} + k_u \cdot \Delta u \cdot \Delta t$$

- **Inputs**
  - **yc**: Desired set-point of the controlled variable.

- **y**: Value of the controlled variable.
- **u_resp**: Value of the output to do respect.
- **g**: Vector of controller gains.

- **Output**
  - **u**: Control output after controller.
- **Configuration menu**:

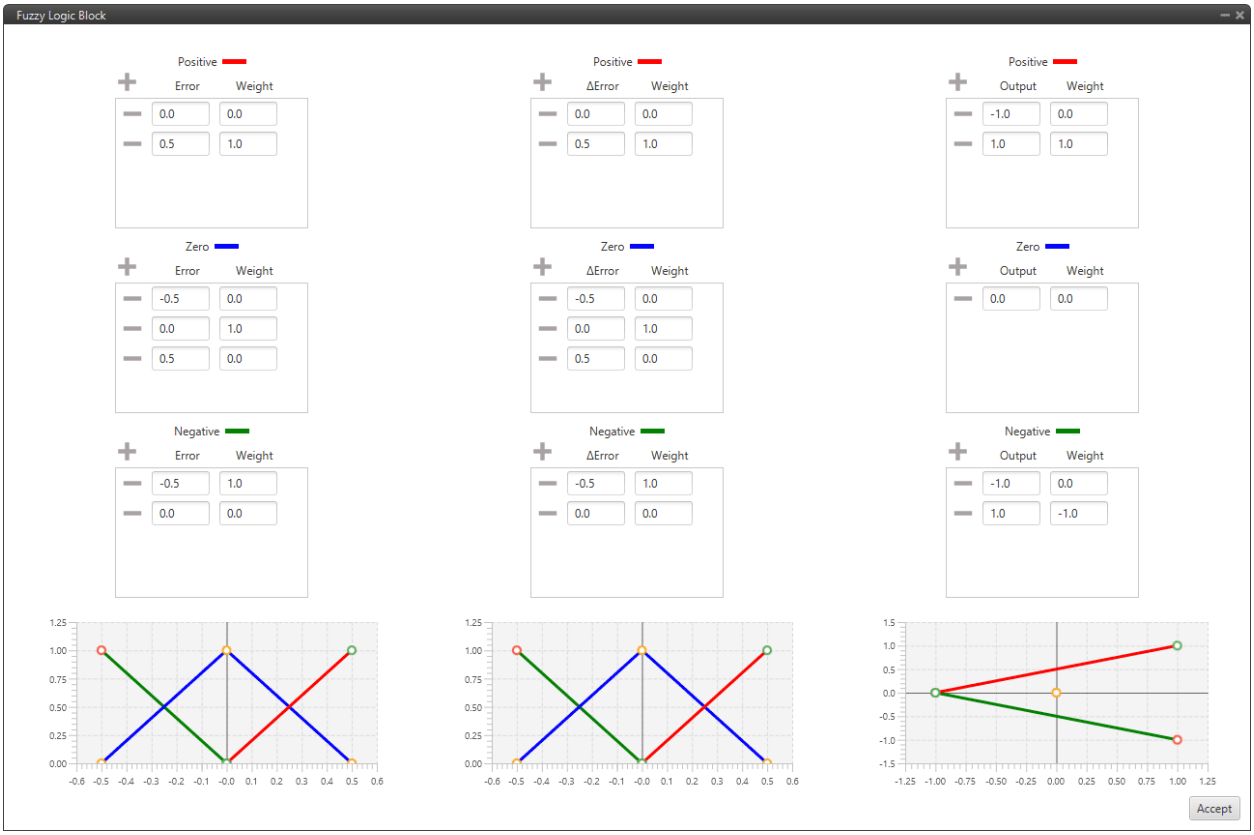


Fig. 254: **Fuzzy Logic Controller block configuration**

These figures represent, respectively from left to right, the error membership function, the change in error membership function and the output membership function that have been explained above.

The user can configure these functions as desired by adding or deleting points.

Note: The default configuration is already designed for control.

2.9.1.5 Driver Control Filter

First, it is presented how the **Adaptive-Predictive control** algorithm has been implemented in blocks.

The part of the algorithm that tries to estimate the system transfer function (System Identification), the part that acts as a filter for the system output (Driver Control Filter) and the control part (Predictive control) are separated.

Therefore, **Driver Control Filter** block works together with the *System Identification* and the *Predictive Control* blocks for the **Adaptative-Predictive control algorithm**.

An example of use is shown below:

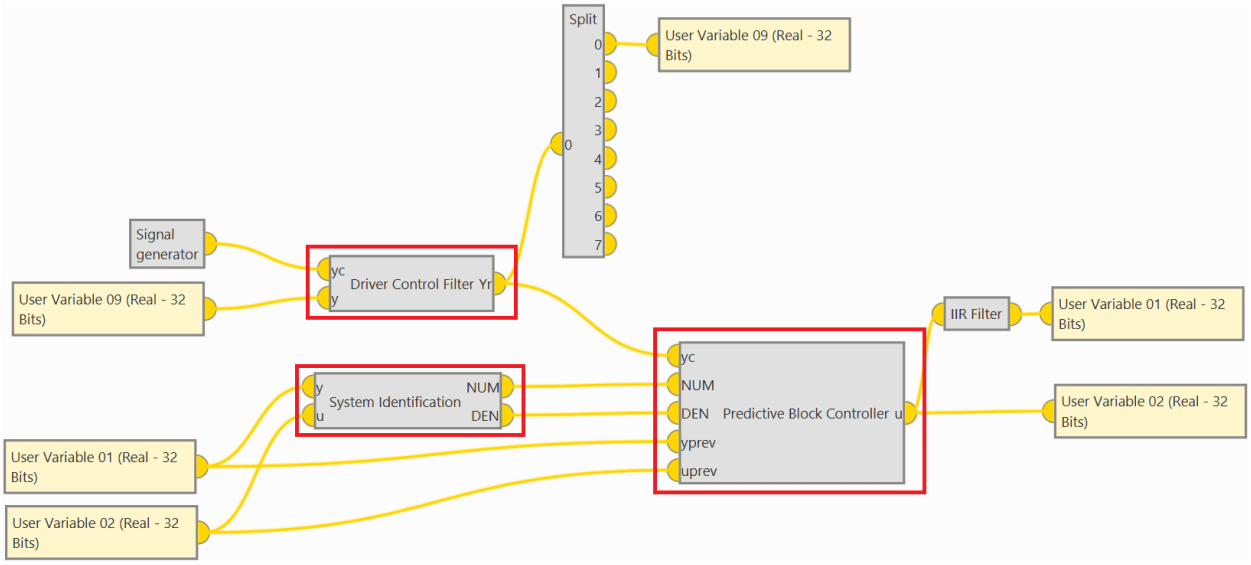


Fig. 255: **Adaptive-Predictive control blocks example**

This example corresponds to the integration of a simulation. Therefore, an *IIR Filter block* and a *Signal generator block* have been added to simulate a real physical system, i.e. **these blocks would not be needed in a real scenario**:

- **IIR Filter** simulates how the output responds to the input.
- **Signal generator** simply simulates a desired input function.

Driver Control Filter block gives a vector with variables set points (SP) using a second order filter.

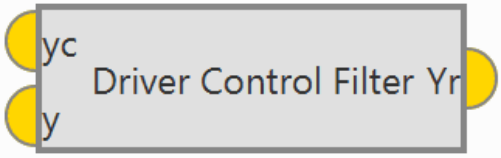


Fig. 256: **Driver Control Filter block**

It acts as a 2-order filter with optimal coefficients for the Adaptive-Predictive control algorithm. These coefficients are calculated from the configurable parameters of the block.

- **Inputs**

  - **yc**: Desired system output (Set Point).

- **y**: Measured system output.

- **Output**
  - **Yr**: Projected desired trajectory vector.

- **Configuration menu**:

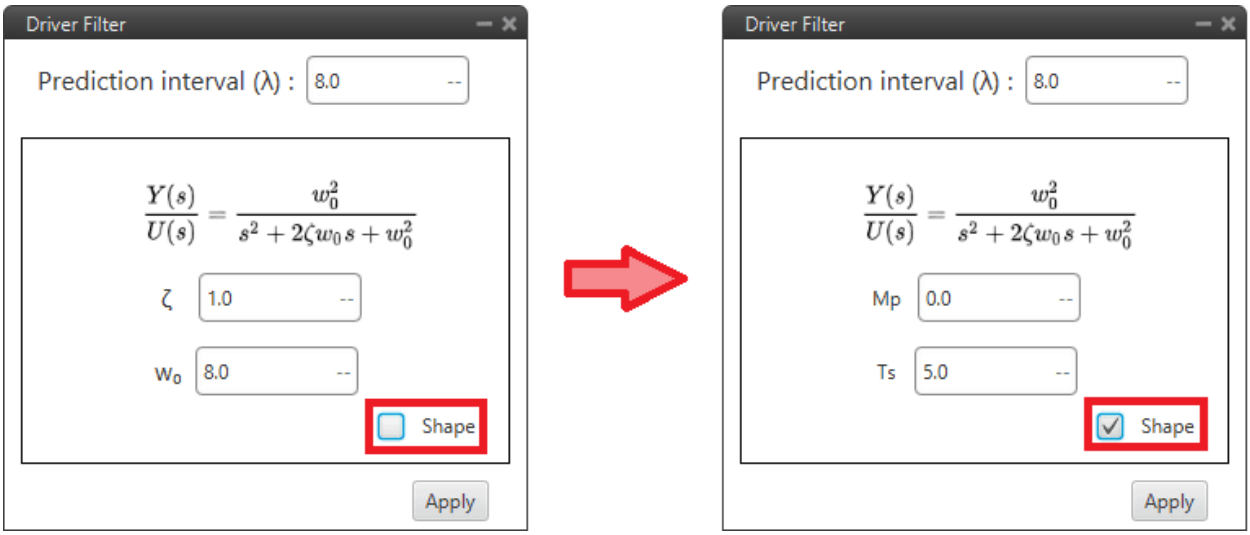


Fig. 257: **Driver Control Filter block configuration**

The following parameters must be set:

– **Prediction interval** (λ): Number of future instants (Prediction Horizon), how many samples are taken from the vector.

– **Shape**: Depending on whether it is enabled or disabled, the paramters to be entered are different. Users can enable or disable it depending on the data available to them:

  * ζ: Damping ratio.
  * w_o: Natural frequency.
  * **Mp**: Maximum overshoot.
  * **Ts**: Settling time (2% criteria).

Note: The relationships between the different parameters are:

$$Mp = \exp\left(-\zeta \cdot \frac{100 \cdot \pi}{\sqrt{1-\zeta^2}}\right)$$

$$Ts = \frac{4}{\zeta \cdot w_0}$$

2.9.1.6 System Identification

System Identification block gives the coefficients of the transfer function at Z-domain,

$$T(z) = \frac{B(z)}{A(z)}$$

- Where, A and B are polynomials in z.

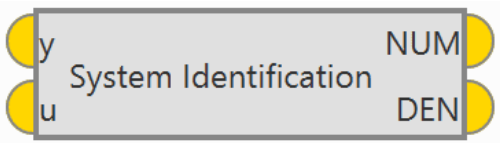


Fig. 258: **System Identification block**

This blocks works together with the *Driver Control Filter* and the *Predictive Control* blocks for the *Adaptative-Predictive control algorithm*, as explained above.

- **Inputs**
  - **y**: System output. This is the measurement.
  - **u**: System input. This is the control action.
- **Outputs**
  - **NUM**: System coefficients numerator.
  - **DEN**: System coefficients denominator.
- **Configuration menu**:

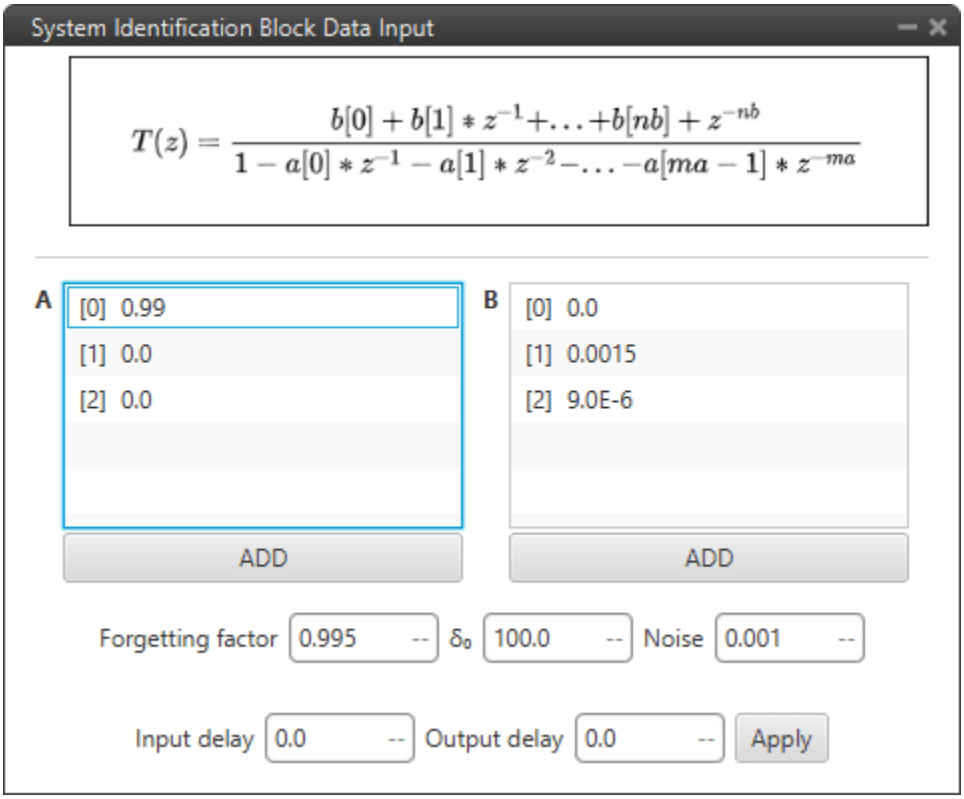


Fig. 259: **System Identification block configuration**

- **Forgetting factor** (λ): Determines how many inputs and outputs are taken into account for the estimation. **Recommended** values **between 0.98 and 0.995**.
- δ_0: Initial value of covariance matrix.
- **Noise** (γ): From this noise threshold, the RLS (Recursive Line Square) is not calculated.
- **Input/Ouput delay**: Given the input/output size, a delay is applied to the sample vector size.

2.9.1.7 Predictive Control Block

Predictive Block Controller gives the optimal control output given a dynamic model as a result of the system identification block.

Fig. 260: **Predictive Control block**

This blocks works together with the *Driver Control Filter* and the *System Identification* blocks for the *Adaptative-Predictive control algorithm*, as explained above.

- **Inputs**
  - **yc**: Desired system output (SP) or trajectory.
  - **NUM**: Numerator coefficients of system model plant.
  - **DEN**: Denominator coeffcients of system model plant.
  - **yprev**: Previous system output.
  - **uprev**: Previous system input.
- **Output**
  - **u**: Control output.
- **Configuration menu**:

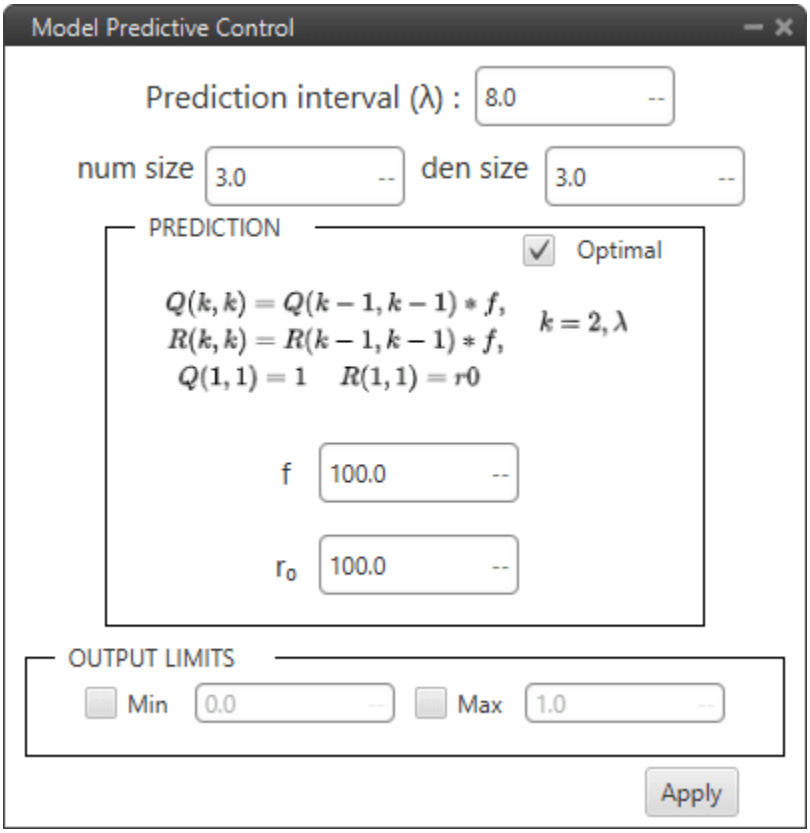


Fig. 261: **Predictive Control block configuration**

The following parameters must be set:

- **Prediction interval** (λ): Number of future instants (Prediction Horizon), how many samples are taken from the vector.
- Optimal: Depending on whether it is activated or deactivated, the following parameters are auto-calculated or not.

  - ∗ **f**: If this factor is greater than 1, the past measurements have greater weight.
  - ∗ $\mathbf{r_0}$: If this factor is grater than 1, there is a more aggresive follow-up of the reference, on the contrary it is smoother.
- **Output Limits**: Maximum and minimum limits for the controller output (for the control signal u).

2.9.1.8 Quaternion Control

Quaternion Control block for fixed multirotor aircraft.

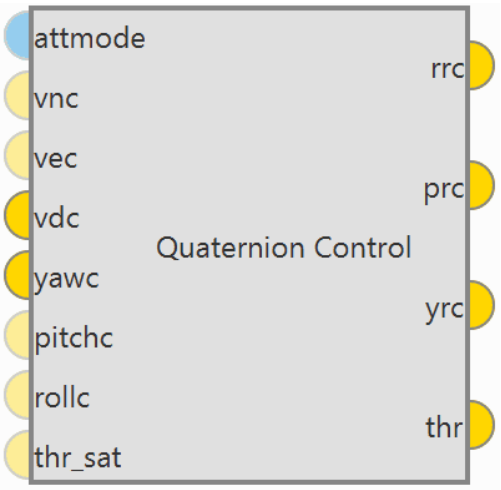


Fig. 262: **Quaternion Control block**

- The Quaternion Control mathematical implementation in Veronte Autopilot 1x is the following:
  - The quaternion control algorithm calculates the desired direction and magnitude of the thrust vector of a multicopter in order to achieve the desired NED velocities, these generate a three-component acceleration vector in NED coordinates, which is compared with the actual direction of the thrust vector to obtain a quaternion representing the rotation from the actual to the desired thrust.

    The error is applied to a control law determined by a time constant that provides the body angle rates to achieve the desired acceleration.
  - In addition, the algorithm is **divided in two quaternion calculations** depending on the yaw being controlled or not:
    - ∗ Reduced: Only the crucial angles of the thrust desired vector are considered. The yaw is not controlled directly.
    - ∗ Full: Both the pointing direction of the vector and the yaw is controlled.
- **Inputs**

  (Optional) **attmode**: Flag for velocity (hover) or angle (hold) control. If true only the angles will be controlled, if false or not connected the velocity of the aircraft will be controlled.

  (Optional) **vnc**: Desired north velocity (only used if velocity mode is active). Assumed zero if not connected.

  (Optional) **vec**: Desired east velocity (only used if velocity mode is active). Assumed zero if not connected.

  vdc: Desired down velocity.

- **yawc**: Desired yaw.
- (Optional) **pitchc**: Desired pitch (only used if angle mode is active). Assumed zero if not connected.
- (Optional) **rollc**: Desired roll (only used if angle mode is active). Assumed zero if not connected.
- (Optional) **thr_sat**: Saturated thrust from previous step. Used for respect and antiwindup.

- **Outputs**
  - **rrc**: Desired roll rate.
  - **prc**: Desired pitch rate.
  - **yrc**: Desired yaw rate.
  - **thr**: Desired thrust.
- **Configuration menu**:

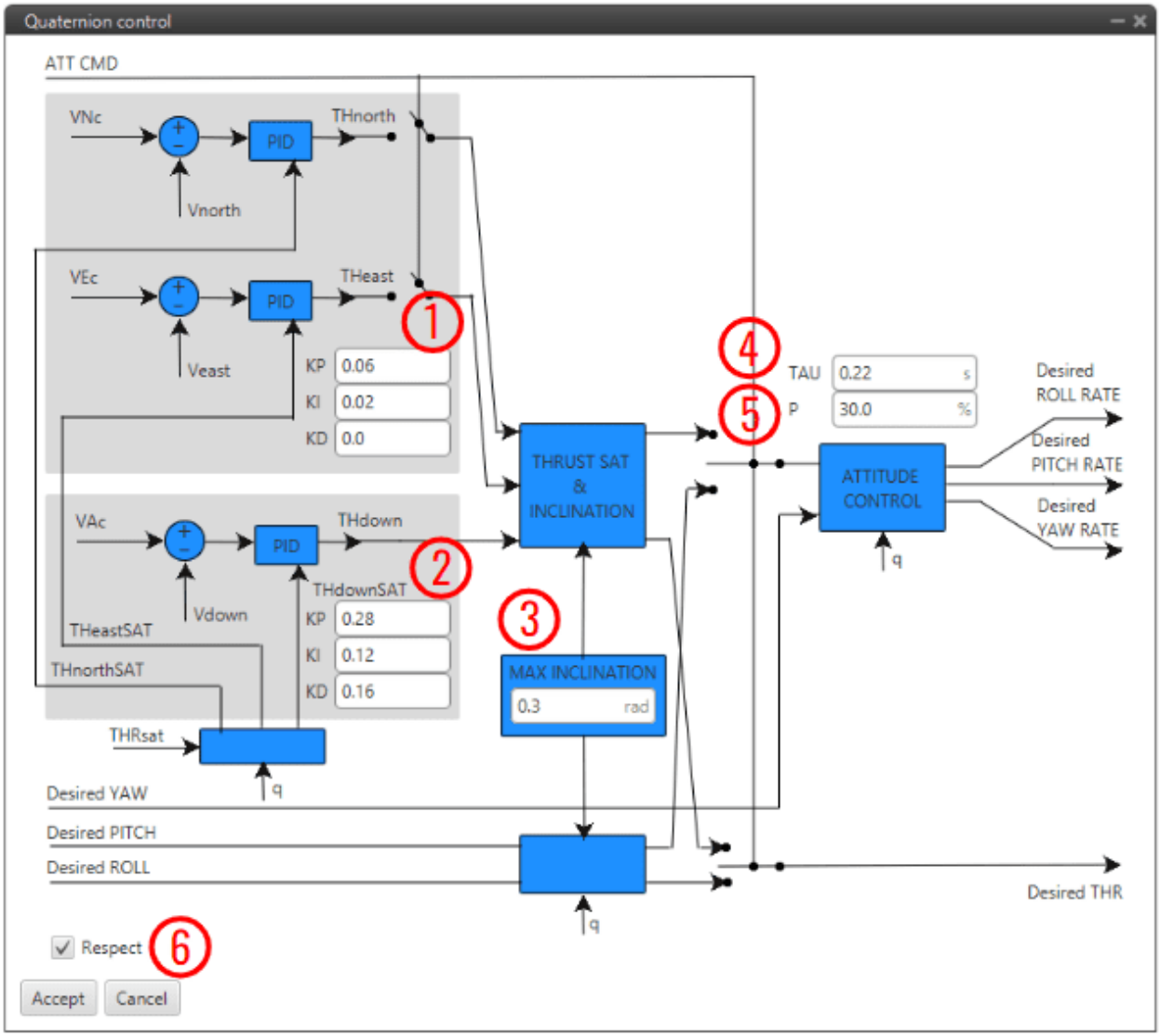


Fig. 263: **Quaternion Control block configuration**

As this block is mainly constituted by 3 PID controllers (fore more infromation, see *PID block*), only some relevant parameters are detailed below:

1. PID to transform desired velocities into desired accelerations in NED.
2. PID to transform the vertical velocity into desired acceleration.
3. **MAX INCLINATION**: Maximum inclination allowed to the aircraft.
4. **TAU**: Time constant of the system. It is also the gain used for the feedback controller.

   Recommended values between **0.1 and 0.25 seconds**.
5. **P**: Reduction factor indicating how the yaw is to be controlled compared to pitch and roll.

   The idea is that the "yaw control power" is lower in a multicopter as it is controlled by angular momentum difference between motors while pitch and roll is controlled by thrust difference.

   By default, it is configured to 30%.
6. **Respect**: If enabled, when the block is executed for the firs time, an initial *I* value will be applied so that **'thr' = 'thr_sat'** for the first iteration.

Furthermore, the thrust is assumed to be between 0 and 1. In case the user has other values for the thrust, it is recommended to use an *Interpole block* at the input and output to adjust the range between 0 and 1.

2.9.1.9 Total Energy Control

Total Energy Control block has been designed for the **decoupling of the control of the speed and FPA in fixed-wing aircrafts**. This block uses the internal navigation estimation.

It will provide two errors that must be minimized in order to obtain the desired speed and flight path:

- **Energy Distribution Error**: Distribution of sistem energy between kinetical and geopotential energy. This error shall be used to control the aircraft's **pitch** or **FPA**.
- **Energy Rate Error**: Rate of change of the Total System Energy. This error accounts for the necessary increase or decrease in **thrust**.

The outputs **Desired energy rate** and **Desired distribution energy rate** are useful as feed forward in control.

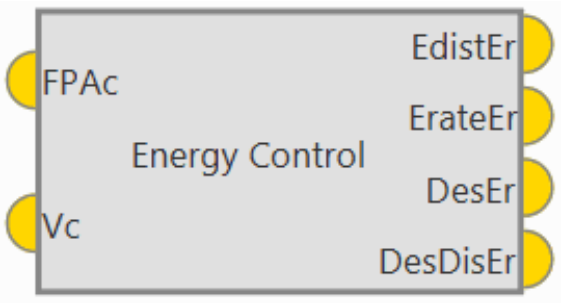


Fig. 264: **Energy Control block**

- **Inputs**

  FPAc: Desired FPA (Flight Path Angle) set-point.

  Vc: Desired velocity set-point. Depending on the block configuration the velocity can be IAS or Ground Speed.

- **Outputs**
  - **EdistEr**: Energy distribution error for pitch control.
  - **ErateEr**: Energy rate error for thrust control.
  - **DesEr**: Desired energy rate.
  - **DesDisEr**: Desired distribution energy rate.
- **Configuration menu**:

  Some parameters of the Energy algorithm can be modified by double clicking on the block:

Fig. 265: **Energy Control block configuration**

1. **Proportional gain for acceleration**: This is an indication of how aggresive the algorithm is when trying to gain speed. The higher the value, the faster the algorithm will try to 'dive' in order to gain speed.

   A typical **recommended** value is around **0.1-0.3**. Higher values are only recommended for fast maneuvering platforms.

2. **V_STALL**: Stall velocity parameter can be enabled or disabled. Users can manually enter this value or select a variable.

3. **Desired speed**: The user must choose between **IAS** and **GS** (Ground Speed) for reference. The use of GS is not recommended unless Airspeed measurement is not available.
4. **K_STALL**: Stall correction coefficient. If 1, energy control is balanced for altitude and speed. If 0 only speed control is taken into account.
5. **IAS/V_STALL**: Speed/Stall ratio. Ratio between current speed and minimum speed.
6. **Stall correction interpolation function**: Defines how the relationship between the stall correction coefficient and the Speed/Stall ratio works. **The default configuration** (as shown in the figure above) is **recommended**.

Note: The Stall correction coefficient is a **Safety** tool that can be used to sacrifice altitude control in order to improve speed control when speed gets close to the stall velocity (**V_STALL**) defined above.

2.9.2 Data Source/Sink blocks

- **Source blocks** allow to **import into the program any variable** available in the system. Additionally the **Const Real/Vector** allows to create a constant variable or vector.
- **Sink blocks** allow **overwriting** the following system **variables**:
  - User Variables
  - Desired variables (variables whose name starts with '**Desired**')

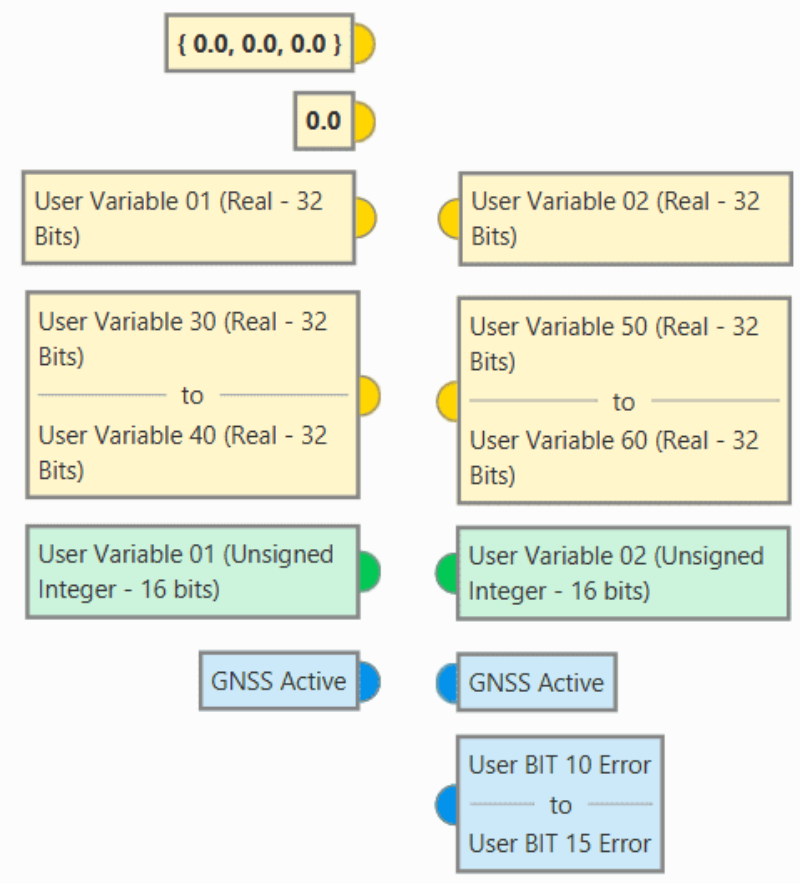


Fig. 266: **Source/Sink blocks**

Warning:

- **Desired variables** are naturally written by **Guidances** algorithms. If an active **Guidance is writting** a certain Desired variable, writting it with a **Sink block** should be **avoided**.
- Using **Sink blocks to overwrite System Variables** usually results in the change not taking effect, but in **some cases** could end up causing Autopilot 1x to **malfunction**.
- Avoid using **Sink blocks** to write any variable that does **not belong** to one of the **groups listed** above.

2.9.3 Devices blocks

Devices connected to Autopilot 1x and a clock can be configured with these blocks.

2.9.3.1 Clock

Clock block computes the time elapsed since the last reset or since the last step execution (depending on the block configuration).

Fig. 267: **Clock block**

- **Input**
  - (Optional) **Reset**: The clock is reset when the input value is TRUE. Assumes FALSE if unconnected.
- **Output**
  - **Time**: Computed time in seconds.
- **Configuration menu**:

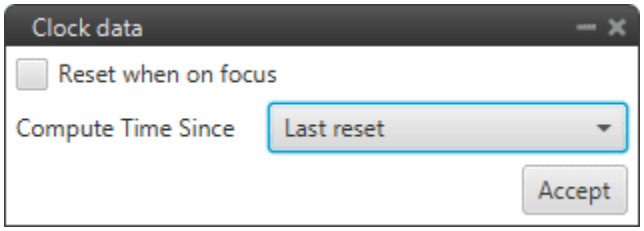


Fig. 268: **Clock block configuration**

  - **Reset when on focus**: If enabled, the clock will be reset the first time it is executed.
  - **Compute Time Since**: The available options are:
    - **Last reset**: The block acts as a 'normal clock', counting the time since it was started/restarted.
    - **Last step**: Time elapsed since the program, in which the block is added, was executed.

2.9.3.2 Gimbal

Gimbal block is a gimbal device controller that uses current navigation estimation.

It allows users to configure a Gimbal Camera by defining the movements the system has (from predefined combinations of Pan, Tilt and Roll), its logic and a distance vector.

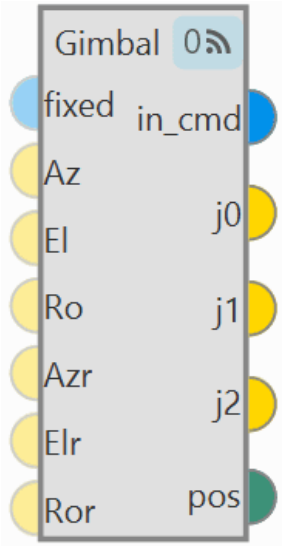


Fig. 269: **Gimbal block**

- **Inputs**

  (Optional) **fixed**: Mode of operation:

  - If **TRUE** the gimbal is in **absolute orientation mode** and uses the **inputs** '**Az**', '**El**' and '**Ro**'.

  - If **FALSE** the gimbal is in **arcade mode** and uses the **inputs** '**Azr**', '**Elr**' and '**Ror**'.

  - If **not connected** assumes **FALSE**.

  (Optional) **Az**: **Desired azimuth**. This input is only used when '**fixed**' input is **TRUE**. Assumes **zero if not connected**.

  (Optional) **El**: **Desired elevation**. This input is only used when '**fixed**' input is **TRUE**. Assumes **zero if not connected**.

  (Optional) **Ro**: **Desired roll**. This input is only used when '**fixed**' input is **TRUE**. Assumes **zero if not connected**.

  (Optional) **Azr**: **Desired azimuth rate**. This input is only used when '**fixed**' input is **FALSE**. Assumes **zero if not connected**.

  (Optional) **Elr**: **Desired elevation rate**. This input is only used when '**fixed**' input is **FALSE**. Assumes **zero if not connected**.

  (Optional) **Ror**: **Desired roll rate**. This input is only used when '**fixed**' input is **FALSE**. Assumes **zero if not connected**.

- **Outputs**

  in_cmd: Has a value of **TRUE** when the gimbal block is being **externally commanded**, FALSE otherwise.

  j0: Gimbal joint 0 angle in radians. This is the **desired Pan angle**.

  j1: Gimbal joint 1 angle in radians. This is the **desired Tilt angle**.

  j2: Gimbal joint 2 angle in radians. This is the **desired Roll angle**.

  pos: **Position** in the surface of the Earth where the **gimbal is pointing to**.

- **Configuration menu**:

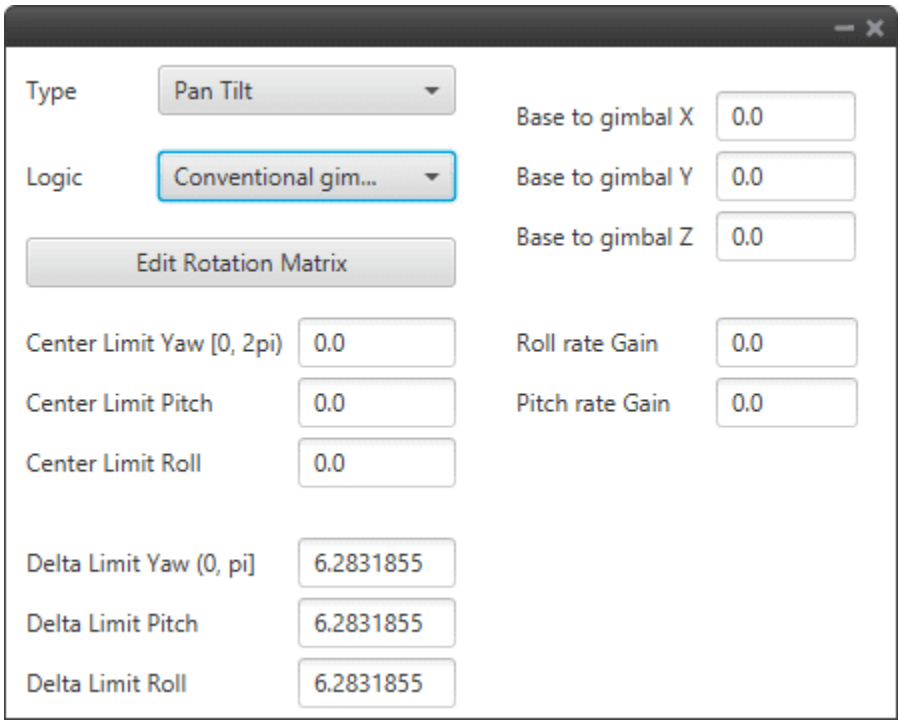


Fig. 270: **Gimbal block configuration**

The following parameters must be configure:

– **Type**: Defines the **angles that the Veronte Autopilot 1x will control** from the payload system from a combination of Pan (Z-axis, same as Yaw), Tilt (Y-axis, same as Pitch) and Roll.

The three options available are:

* **Pan & Tilt**
* **Pan, Roll & Tilt**
* **Roll & Tilt**

– **Logic**: Defines the kind of payload system configured:

* **Conventional gimbal**: This option writes over the **variables** *Joint 1-3 of Gimbal 1-3* which are **used** later to **configure camera control and stabilization from Autopilot 1x**.
* **Self-stabilized gimbal**: The payload system **only needs movement inputs** and the variables mentioned will have no output.

– **Base to gimbal X/Y/Z**: Defines the vector linking Veronte Autopilot 1x controlling the payload system and the payload system itself, on Veronte body axes.

– **Edit Rotation Matrix**: Matrix to rotate the system to match the aircraft coordinate system.

Fig. 271: **Gimbal block configuration - Rotation Matrix**

- **Center Limit Yaw [0, 2pi]/Pitch/Roll**: Center of the range of movement.
- **Delta Limit Yaw [0, pi]/Pitch/Roll**: That is how much the gimbal can move in positive and negative from the center defined above.

Note: This part of limit is because there are some gimbals that cannot make a full turn on some axis.

- **Roll/Pitch rate Gain**: Gains to compensate for **roll rate** or **pitch rate inputs**.

An example of use is given below:

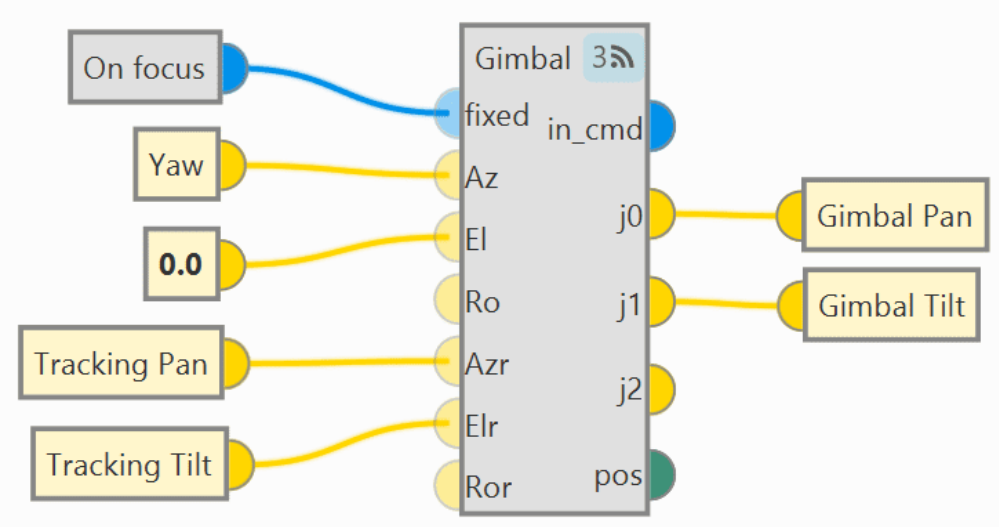


Fig. 272: **Gimbal block - Example of use**

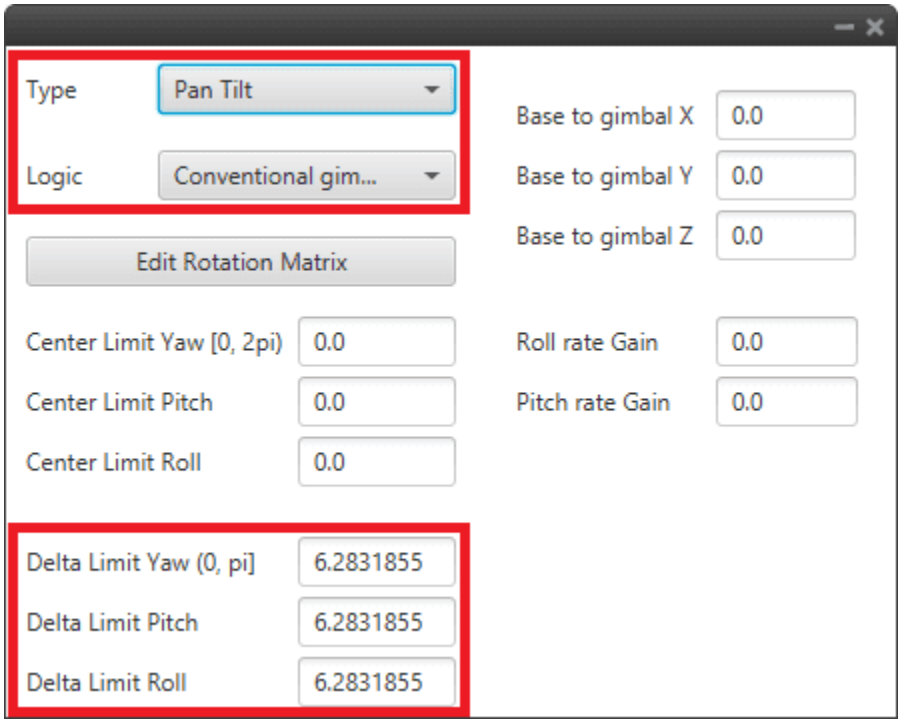


Fig. 273: **Gimbal block - Configuration example**

2.9.3.3 Stick

Stick block is a stick reader, with it the user can configure the stick parameters for manual and arcade modes.

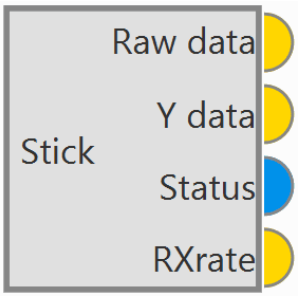


Fig. 274: **Stick block**

Warning: This block is mandatory for the use of the transmitter. For more information on the stick configuration, see the *Stick - Integration examples* section of this manual.

- **Outputs**
  - **Raw data**: Raw stick channels.
  - **Y data**: Stick channels after transformation (matrix and offset).
  - **Status**: TRUE if the stick is read without timeouts, FALSE otherwise.

- **RXrate**: Stick update frequency rate (Hz).

Configuration menu:

- **Sources**: In this tab the user can set multiple transmitter inputs with the respective **priority**, **from top to bottom**.

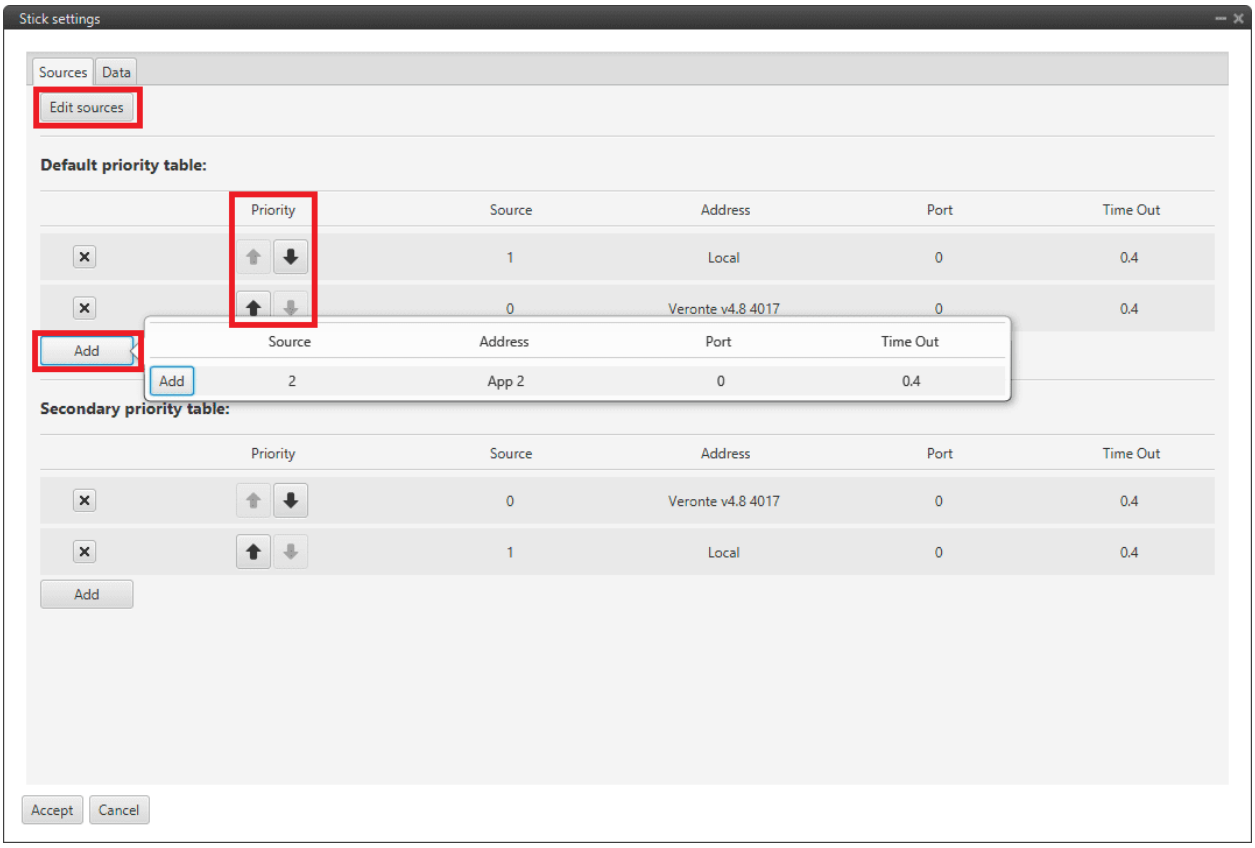


Fig. 275: **Stick block configuration - Sources**

  - **Priority table**: By default, one priority table is set. The user can configure a second one.
    - **Priority**: Use arrows to determine the priority of the selected source. Priority is set **from top to bottom**.
    - **Add**: An already defined source can be added to the priority table.
  - **Edit sources**: New sources can be defined in this menu.

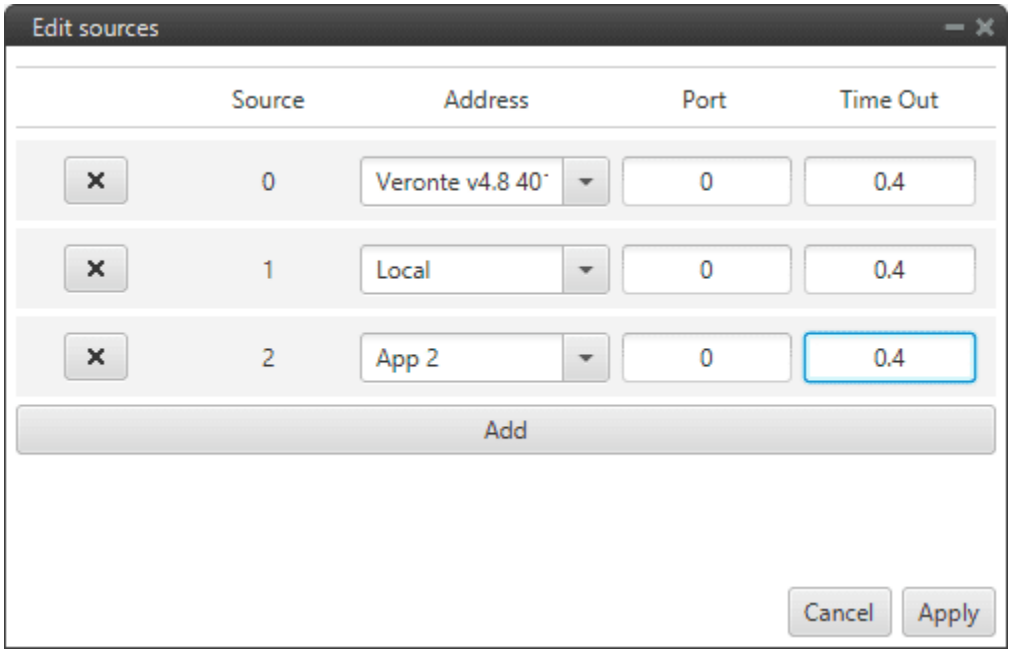


Fig. 276: **Edit sources menu**

* **Source**: It is the order in which sources are created in this menu. This **does not set the priority**.
* **Address**: This defines the source from which the stick information is taken from. The following options are the most common:
  - App 2: Means that the information is coming from the **stick widget** of **Veronte Ops**.
  - Local: Represents the **actual selected autopilot** (i.e. the transmitter is connected to it).
  - Any: The information comes from all linked autopilots.
  - 1x v4.X XXXX: Means the information is coming from a **particular autopilot**, which needs to be visible in **Veronte Link**.

  For more information on the available addresses, see List of addresses section of the **1x Software Manual**.
* **Port**: From each source it is posible to have more than one stick information, e.g. two transmitters can be connected to the same autopilot. The port is an identifier to distinguish them.
* **Time Out**: This defines the time to consider the source inactive. Therefore the incoming stick information will be always the one from the source with higher priority and active. Once it is considered inactive the following active source will send its stick information. The lower this value, the more frequent the stick will be lost. We **recommend** a value of **0.4 s**.

- **Data**: In this tab the user can configure $Mix\ Matrix$, $Raw\ channels$ and $Offset$.

  The movement that the pilot makes on the stick produces variations on a vector called $(Raw\ channels)$ of length n, where n goes from 1 to the total number of employed transmitter channels. The values reached by the components of $(Raw\ channels)$ are limited between 0 and 1. These stick movements need to be processed to produce the input signals that will go into the control algorithm, in the case of arcade mode; or directly into the servos for manual mode.

  The process begins by mapping each one of the sticks inputs to PWM signals into a vector called $Output$ of length m, where m goes from 1 to the total number of actuators.

  The full definition of $Output$ is $Output = (Mix\ Matrix) \cdot (Raw\ channels) + Offset$, where:

- $(Mix\ Matrix)$ is a matrix that transforms raw stick inputs $(Raw\ channels)$ to PWM signals $Output$.
- $Offset$ is an offset vector, which corrects the $Output$ vector.

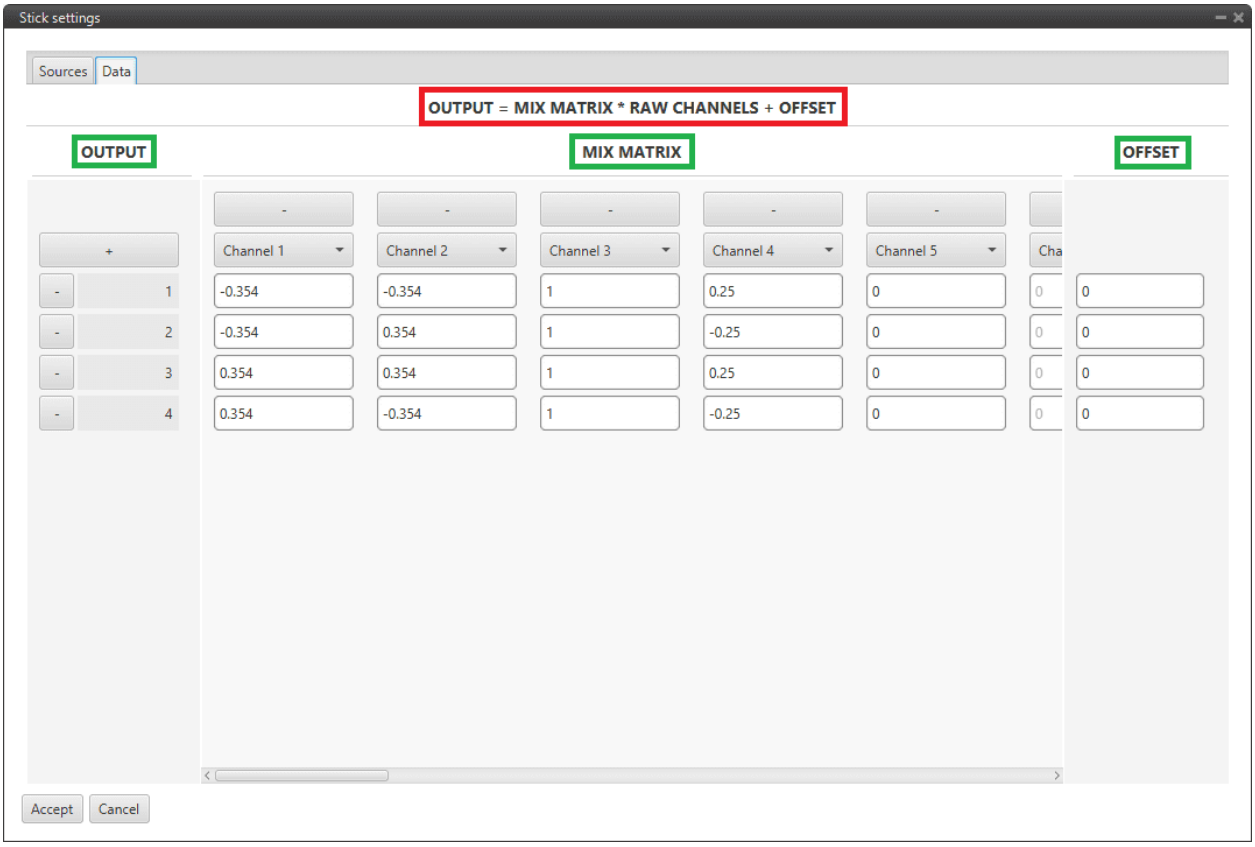
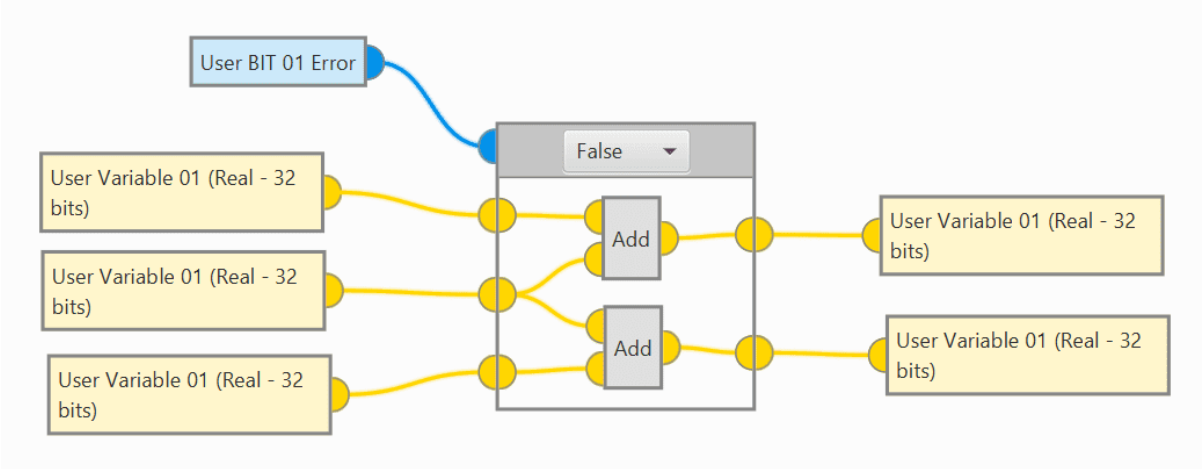


Fig. 277: **Stick block configuration - Data**

2.9.4 Execution Flow blocks

Execution Flow blocks allow to switch sections of a program during its execution among a set of pre-configured options.



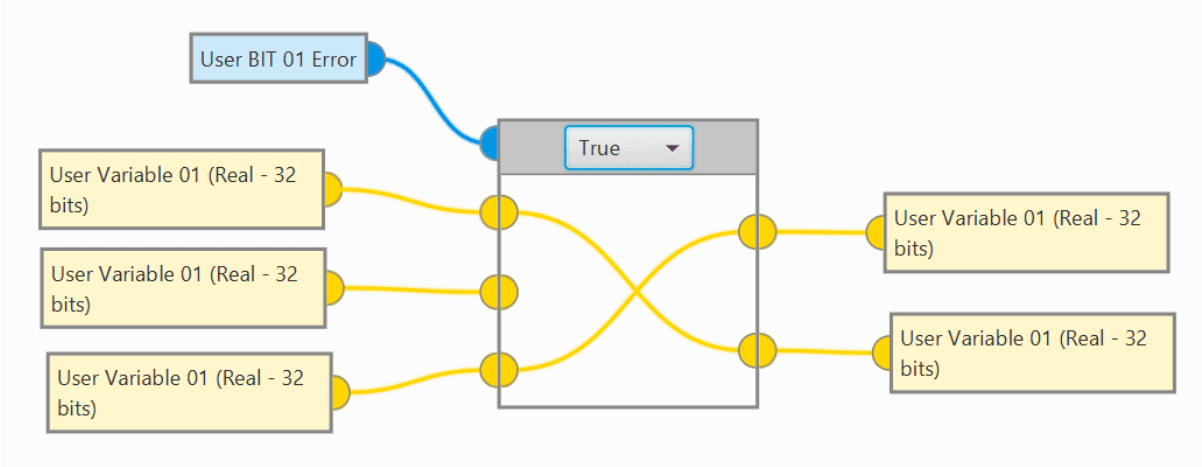


Fig. 278: **Execution flow blocks**

Execution flow blocks are divided into 2 different types of blocks: *On focus block* and *Switch blocks*.

2.9.4.1 On focus block

The On Focus block outputs a boolean value, which is **only True the first time** the block is executed.

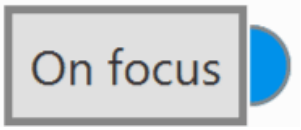


Fig. 279: **On focus block**

- If used inside a **Switch Block**, the value will be **True** each time the case is selected.
- On Focus can be used to **trigger actions** or **initialize variables** whenever a case is switched.

The following example would initialize **User Variable 01** to **7** whenever **Landing** phase is selected:

Fig. 280: **On focus block example**

2.9.4.2 Switch blocks

- **If-Else Switch block**: Choose between two cases based on the state of a boolean variable.

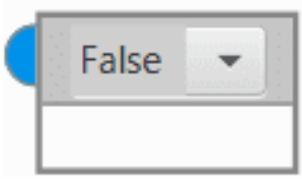


Fig. 281: **If/Else switch block**

It is possible to swap the blocks of each case (False/True) by simply **right-clicking** inside an if/else switch block and selecting '**Invert**'.

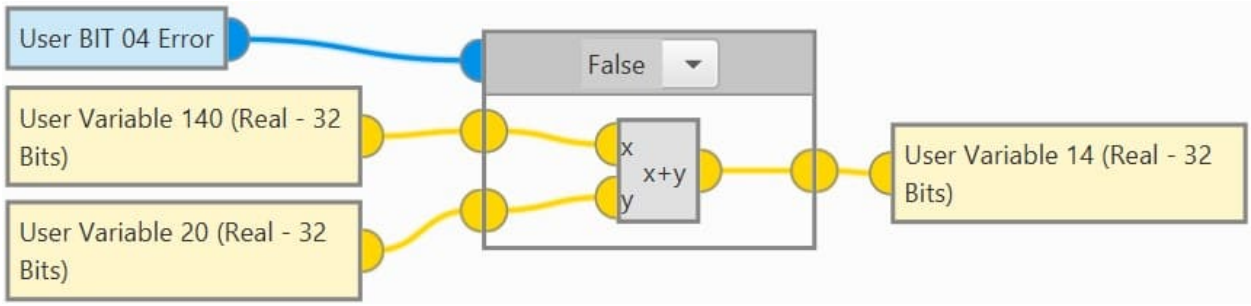


Fig. 282: **Invert created blocks**

- **Integer Switch block**: Choose a case based on the value of an integer variable.

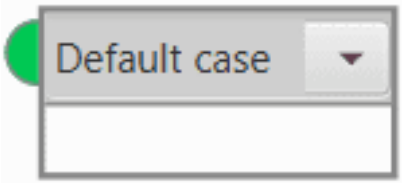


Fig. 283: **Integer switch block**

Right click on the Integer switch block to access the configuration options:

  - **Add Case**: Create a new empty case.
  - **Copy Case**: Create a copy of the current case.
  - **Delete Case**: Delete the current case.
  - **Add Entry**: Add a new **entry** to the current case. An **entry** is a condition under which the case will be selected. The same entry can on only be in one case at a time. Adding an **entry that already exists** will move said entry to the current case.
  - **Delete Entry**: Remove an entry from the current case.
  - **Set as Default case**: The Default case will be executed whenever the switch condition does not match any of the existing entries.

- **Phase Switch block**: Same as Integer Switch, but using Flight Phases as the switch condition.

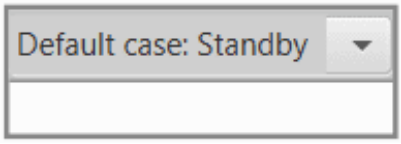


Fig. 284: **Phase switch block**

> **Warning:** **Phase Switch and Integer Switch Blocks** will report a '**PDI ERROR**' if they don't have at least 1 case with entries.

Use of switch blocks

- **Add blocks inside**: Drag and drop the desired block inside the Switch block:

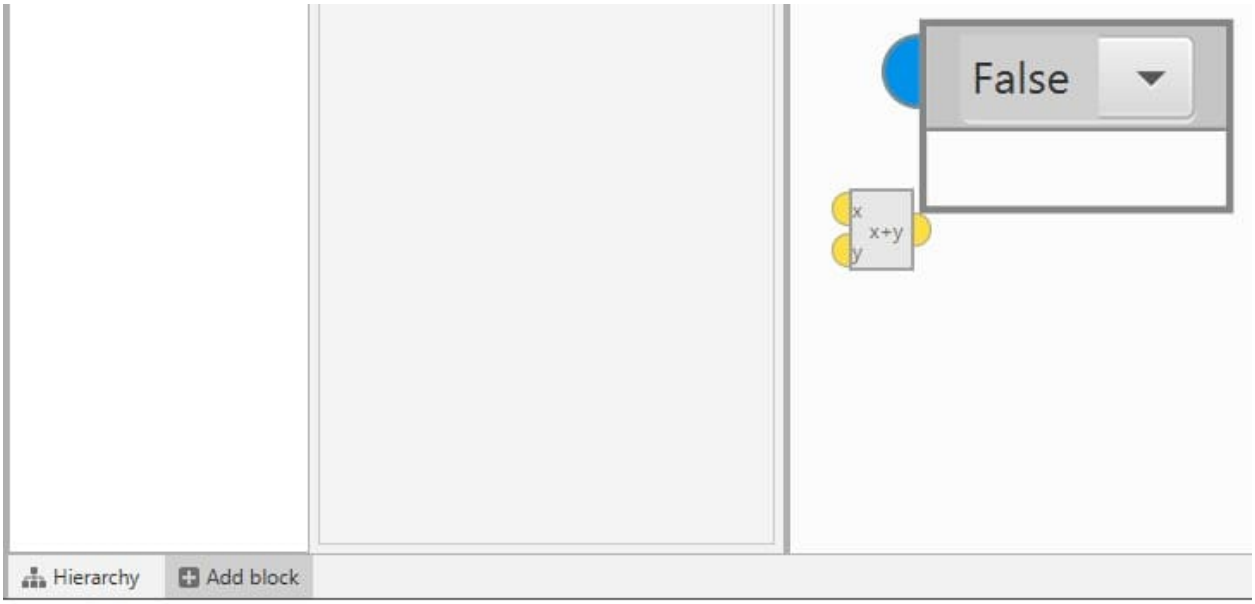


Fig. 285: **Create a block inside a switch block**

- **Input/Output**: **Right click** inside a Switch block and select **add Input/Output**. To **remove** them, **right click** and select **Remove Input/Output**:

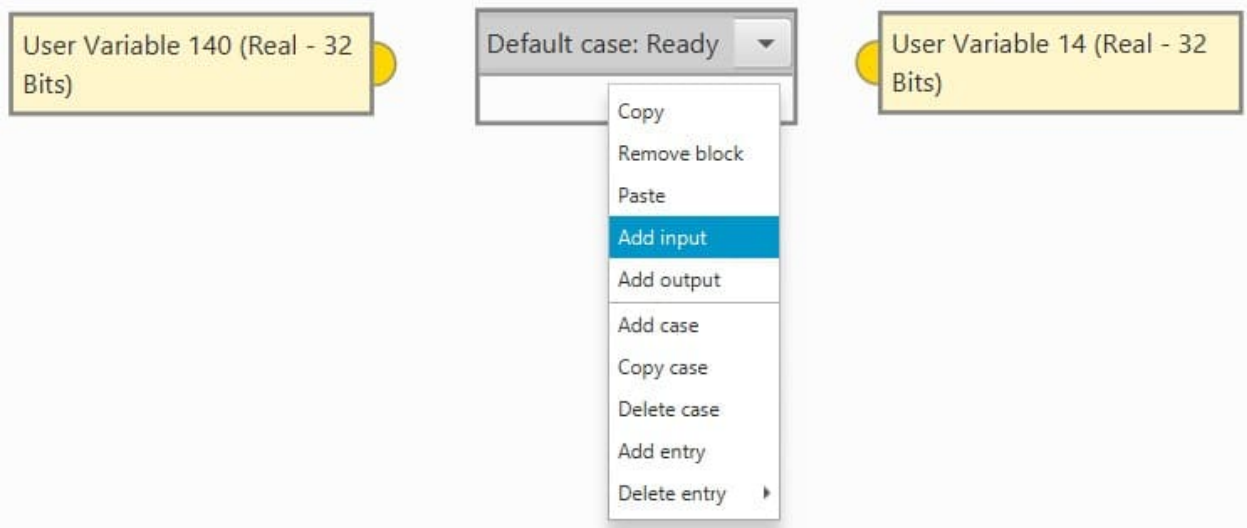


Fig. 286: **Add/Remove an input/output of a switch block**

Note: The size of a switch block depends on the blocks it contains. A switch block will always have the size of the **biggest** of its existing cases.

2.9.5 Guidance blocks

When defining a guidance system, we refer to a set of commands sent to the platform controller in order to make it carry out a certain task. This task could be follow a line, climb, land, hold one of its states at a certain value and so on.

In **1x PDI Builder**, it is possible to combine a series of guidances to create custom flight phases that will make the aircraft perform in a given way.

Each **Guidance block** contains a **set of parameters to be configured**. All of them are presented as follows.

Name	Description
Climb	Makes the aircraft climb from the start of the phase to another altitude.
Cruise	Makes the aircraft follow a determined route created by the user.
Landing	Creates the route that the airplane will follow to land.
Rendezvous	Used to create a meeting point where the Air unit will approach a second unit (either Air or Ground) within a determined offset.
Taxi	Creates a linear path along the runway that is followed by the aircraft.
VTOL	Vertical take-off and landing.
Yaw current/heading/north	Indicates the behavior of the platform in the yaw axis.

2.9.5.1 Guidance blocks common configuration

All the guidance blocks presented below, have the **same inputs and outputs**, and some **common configuration parameters**.

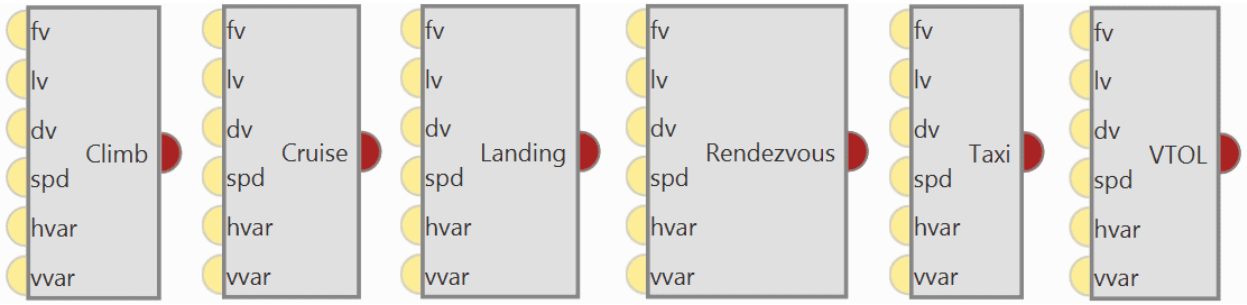


Fig. 287: **Common guidance blocks**

- **Inputs**

  (Optional) **fv**: **First** component of desired 'hover here' arcade velocity in the horizontal plane. The actual direction of this speed depends on the selected arcade axis.

  (Optional) **lv**: **Second** component of desired 'hover here' arcade velocity in the horizontal plane. The actual direction of this speed depends on the selected arcade axis.

  (Optional) **dv**: Down (vertical) 'hover here' desired arcade velocity.

  (Optional) **spd**: **Arcade cruise speed increment**.

  (Optional) **hvar**: Scale variable for the T-Sched PID of the horizontal guidance.

  (Optional) **vvar**: Scale variable for the T-Sched PID of the vertical guidance.

- **Output**

  Pin 0: Guidance data for the **Guidance Computation block**.

  > **Warning:** To produce a guidance computation, these blocks must be connected to the *Guidace Computation* block via this output.

- **Configuration menu**:

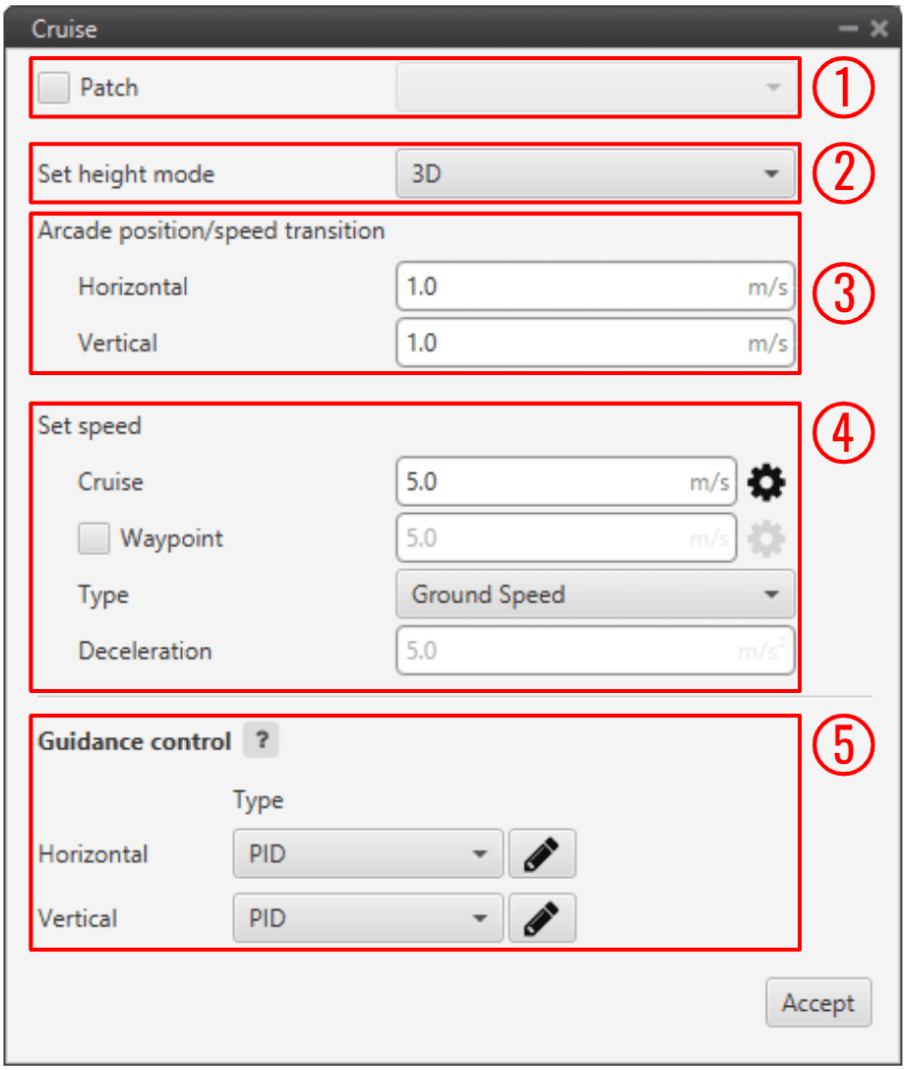


Fig. 288: **Common guidance blocks configuration**

All the parameters that define the guidance are detailed:

1. **Patch**: This option allows the user to select the first path to be flown by the aircraft. The user should first enable this option and then select the desired path to bo the first-of-the-route.
2. **Set height mode**: Height mode indicates how the aircraft will perform the defined path. There are three possible height modes:
   - **2D mode**: If this mode it is selected, the platform will follow the predifined route without taking into account the altitude of the waypoints, it will keep the altitude that it has at the moment it enters in the guidance.
   - **2.5D mode**: The vehicle will follow a 3D trajectory that connects both waypoints. However, it will give **priority to horizontal guidance**. **Autopilot 1x will try to adjust its position and altitude** to the path (both horizontally and vertically), but if for any reason it cannot reach the altitude of the final waypoint, it will considered that it has been reached if its position matches the position of the waypoint.
   - **3D mode**: The vehicle will follow a 3D trajectory that connects both waypoints. In this case, **horizontal and vertical guidance have the same priority** level. This means that Autopilot 1x will not consider that a waypoint has been reached until its position and altitude match the waypoint's ones.

As this type of guidance may result in a vertical flight, it is **reserved for multicopters or hybrid platforms**.

3. **Arcade position/speed transition**: In Arcade mode the trajectory generated (*position*) is not followed and instead the aircraft moves according to the commanded *speed*.

   The **Horizontal** and **Vertical speed parameters** serve as the upper thresholds for when the aircraft guidance should be based on position, even in Arcade mode. This parameters are mainly **useful** for platforms like **multicopters**.

4. **Set speed**: This option sets the speed that the vehicle will have during the manoeuvre.

   - **Cruise**: Lets the user set the velocity modulus of the guidance. This velocity can be slightly modified by the autopilot control algorithms.
   - **Waypoint**: If enable, it indicates the speed at which the platform will reach the waypoints of the path, i.e. it will travel along the path with the speed indicated in the option **Cruise** and then it will decelerate or accelerate to the speed indicated here.
   - **Type**: Defined speed. Can be **IAS** (Indicated Airspeed) or **Ground speed**. Normally, **IAS** is used for **airplanes** and **Ground speed** for **multicopters**.
   - **Deceleration**: This can only be configured when **Waypoint** option is enabled. Maximum allowed acceleration/deceleration to meet the desired velocity.

5. **Guidance control**: These PIDs are defined to guarantee stability of guidance loop, they are used to calculate the Desired Speed Vector based on the current position error.

   Then, the resulting vector, along with the Guidance parameters, will be used to generate the Desired variables (ID 100 - 258 Rvars) that can be used as inputs for the control loops.

   For both **Horizontal** and **Vertical** guidance, the user can choose the **type of control** between *PID* or *T-Sched PID*.

   Besides, by clicking on the  icon of both guidance, a pop-up window will appear where the control parameters should be entered, for more information on the latter check *Control blocks*.

   In the horizontal-position PID (see image below), North-East current position of the aircraft is compared to the desired position. The output of the PID controller is going to be a ground speed in the North-East plane, which translates into a desired lateral and front ground speeds in body axes. The same logic applies for the vertical-position PID.

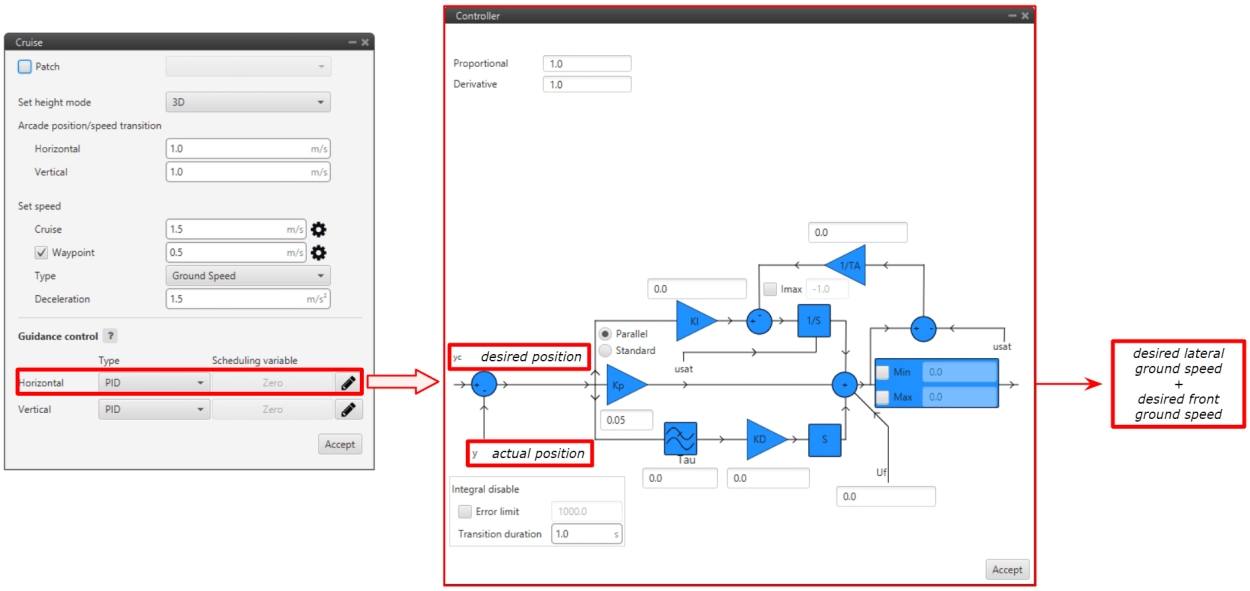


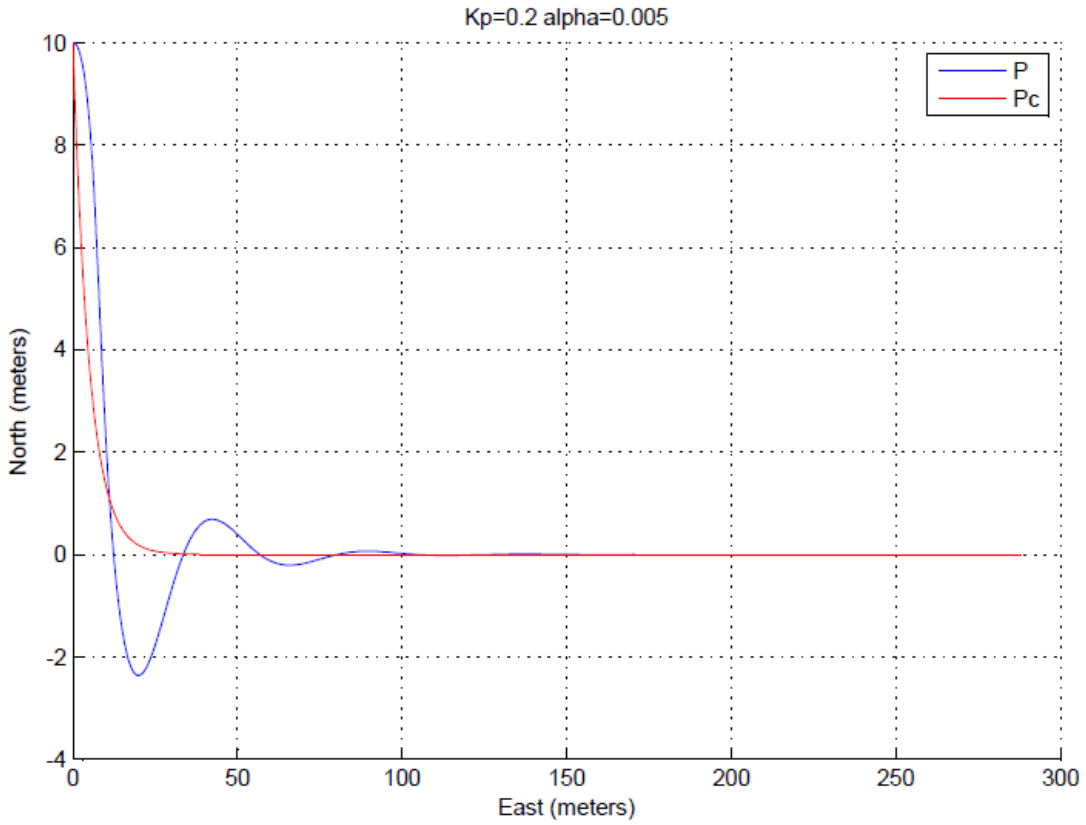
Fig. 289: **Horizontal Position PID**

However, the algorithm is more complex than this simple PID.

For tuning, it is usual **to use only proportional term in the PID**:

- A **high proportional** will **converge faster** to the desired position **but with overshoot**.
- A **lower proportional** will make the **arrival** to the desired position **slower** but it is a **smooth convergence**.

Next figures shows this behavior with $Kp = 0.2$ and $Kp = 0.02$.


Kp=0.2 alpha=0.005
P
Pc
North (meters)
East (meters)
10
8
6
4
2
0
-2
-4
0
50
100
150
200
250
300

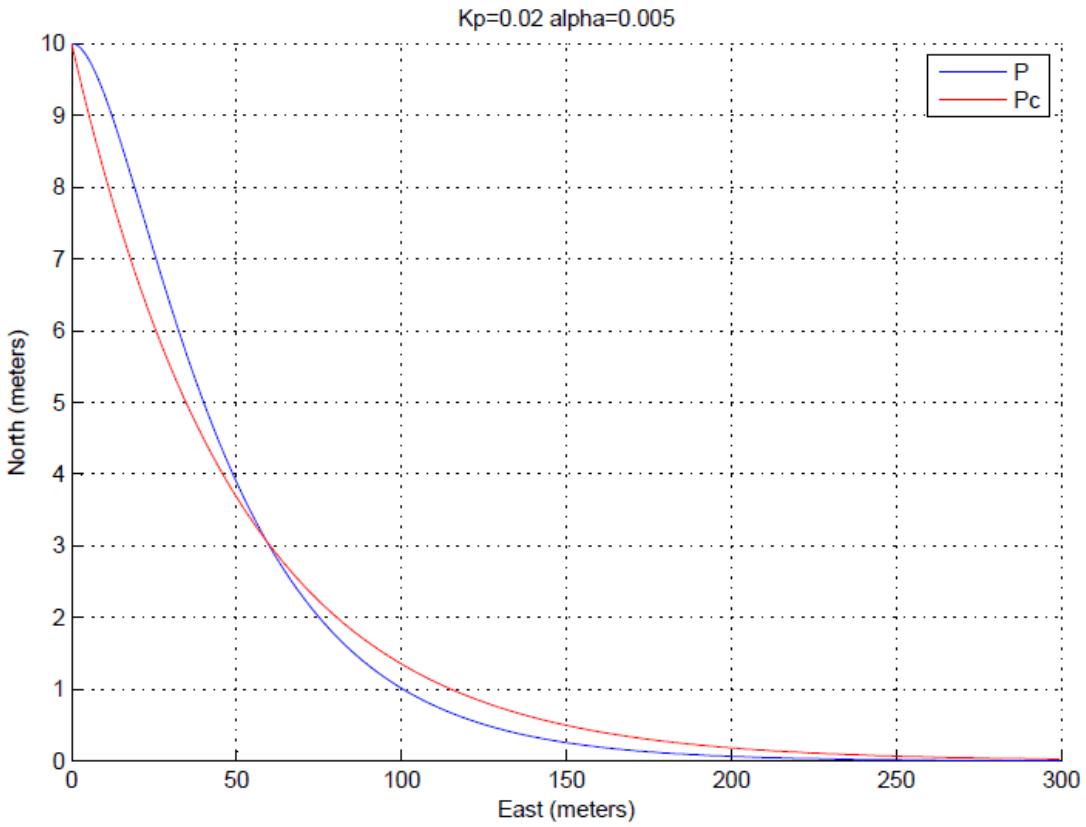


Fig. 290: **PID Proportional gains**

Guidance-generated Variables

The guidances contained within Veronte Autopilot 1x generate a series of variables that are later used in the control loops as the input of the PIDs. Generally, variables named as Desired are used in this Guidance.

The list of variables is the following:

- Desired position.
- Track position.
- Track state (current patch, last patch).
- Desired latitude, desired longitude, desired WGS84, desired MSL, desired AGL.
- Desired velocity.
- Desired front groundspeed, desired lateral groundspeed, desired velocity down.
- Desired tangential acceleration.
- Desired IAS.
- Guidance north error.
- Guidance east error.
- Guidance down error.

- Desired body velocities.
- Desired velocities north, east, down.
- Desired heading, FPA and bank.
- Route-guidance distance - tangential component.
- Route-guidance distance - horizontal component.
- Route-guidance distance - perpendicular component.

2.9.5.2 Climb

Climb guidance is used to make the aircraft climb from the start of the phase to another altitude. Commonly, this guidance is used after the take-off to fly from the ground to cruise altitude through a loiter point, but it can be employed for other purposes.

Climbing guidance generates a three-dimensional trajectory.

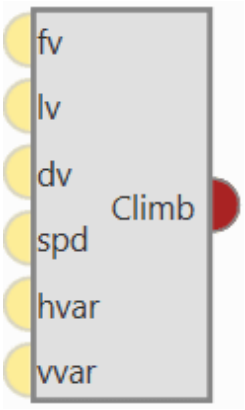


Fig. 291: **Climb block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Configuration menu**:

  The climbing path is automatically generated and is not directly shown to the user until the aircraft enters this phase. This is due to the algorithm recalculating the path each time to take into account the aircraft's actual flight conditions and the user's indicated parameters

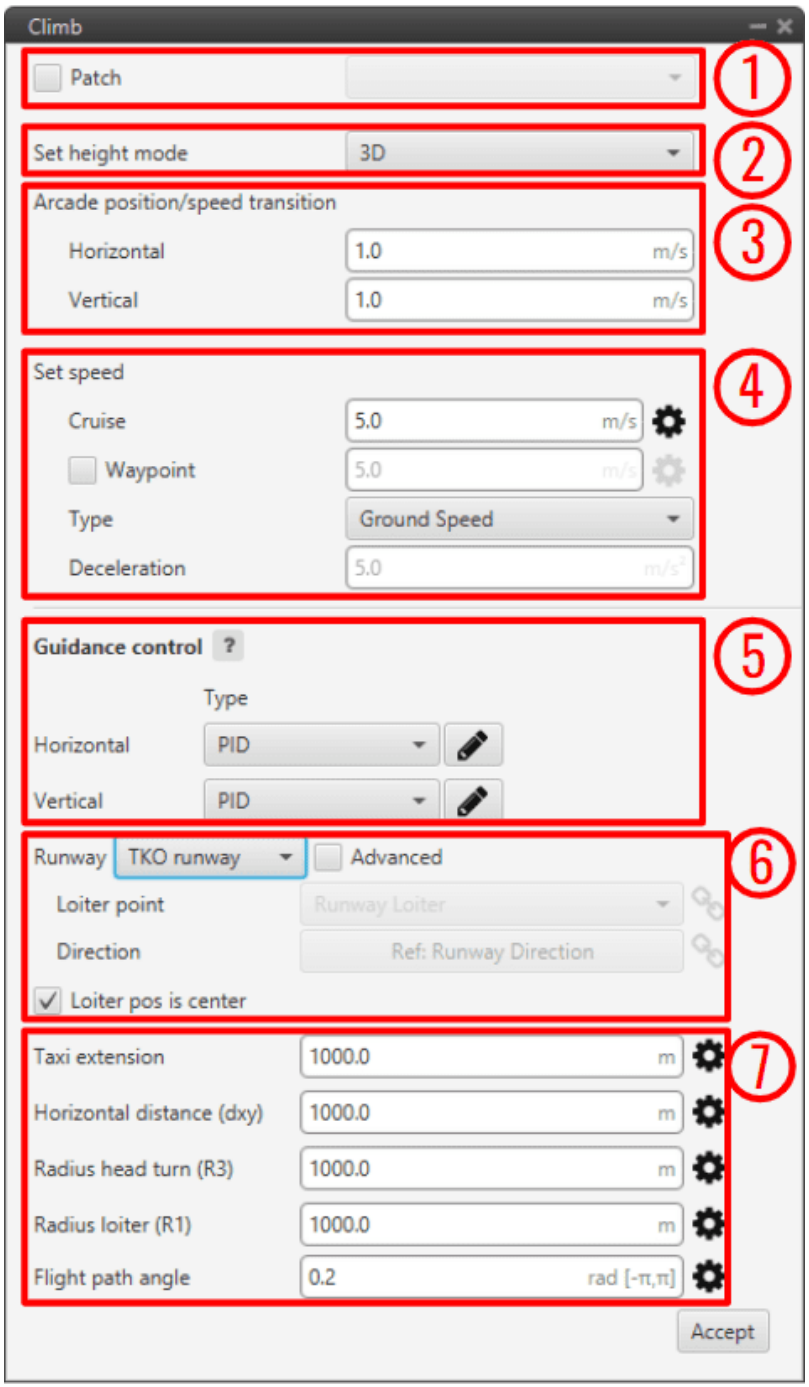


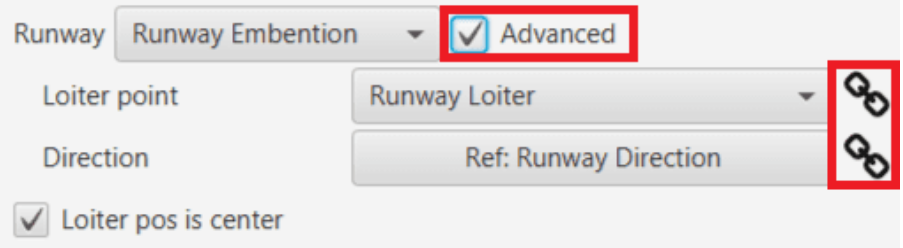
Fig. 292: **Climb block configuration**

Below, the parameters shown above are going to be described. Later, a brief description of the algorithm and its behavior in different possible situations will be presented:

1. **Patch**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
2. **Set height mode**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.

3. **Arcade position/speed transition**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
4. **Set speed**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
5. **Guidance control**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
6. **Runway and Loiter position**: Here the user can define the loiter and runway positions and direction. However, the default option is to define them in the **Runway option** of **Veronte Ops** (for more information, see Veronte Ops manual).

   If the *Advanced* option is chosen, then the user can define these parameters. By clicking on ⚙ or 🔗 different options will be displayed:



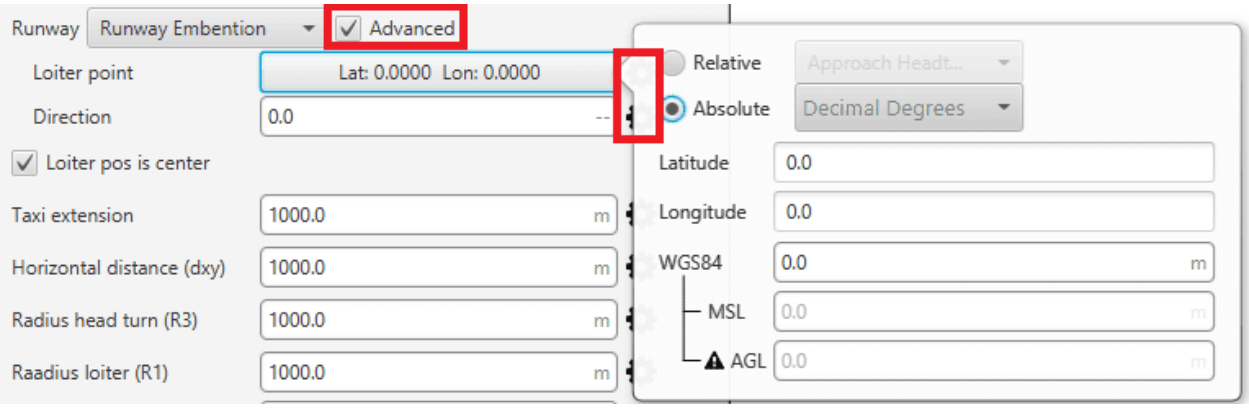


Fig. 293: **Runway and Loiter Position Options**

– **Loiter point**: Defines the loiter point. The two available options are:

  * 🔗 icon selected: By default, this point is the runway's loiter. But the user can select in the drop-down list any other previously defined point. This includes waypoints defined in the Mission (in **Veronte Ops**), among many others.

  * ⚙ icon selected: Alternately, the user can manually define the Loiter point. Then it can be configured in two ways:

    · **Relative**: In this case, the position of the point is relative to another point. That point could be any platform fitted with an Autopilot 1x.

    · **Absolute**: The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress º ” ‘. They are indicated through the latitude, longitude and altitude (being possible to define this last one with respect to the ellipsoid, WGS84, to the sea level, MSL or to the ground, AGL).

- **Direction**: Defines the runway direction. Again, there are two available options:
  - icon selected: By default, it is the same as the selected runway. It can be also chosen from a list of options including runway direction, tailwind direction, etc.
  - icon selected: Alternately, it can also be defined as an angle with respect to the magnetic north.

- **Loiter pos is center**: If this box is enabled, the defined loiter point will be the center of the loiter circular trajectory. In case of not, the circular loiter trajectory will pass through that point.

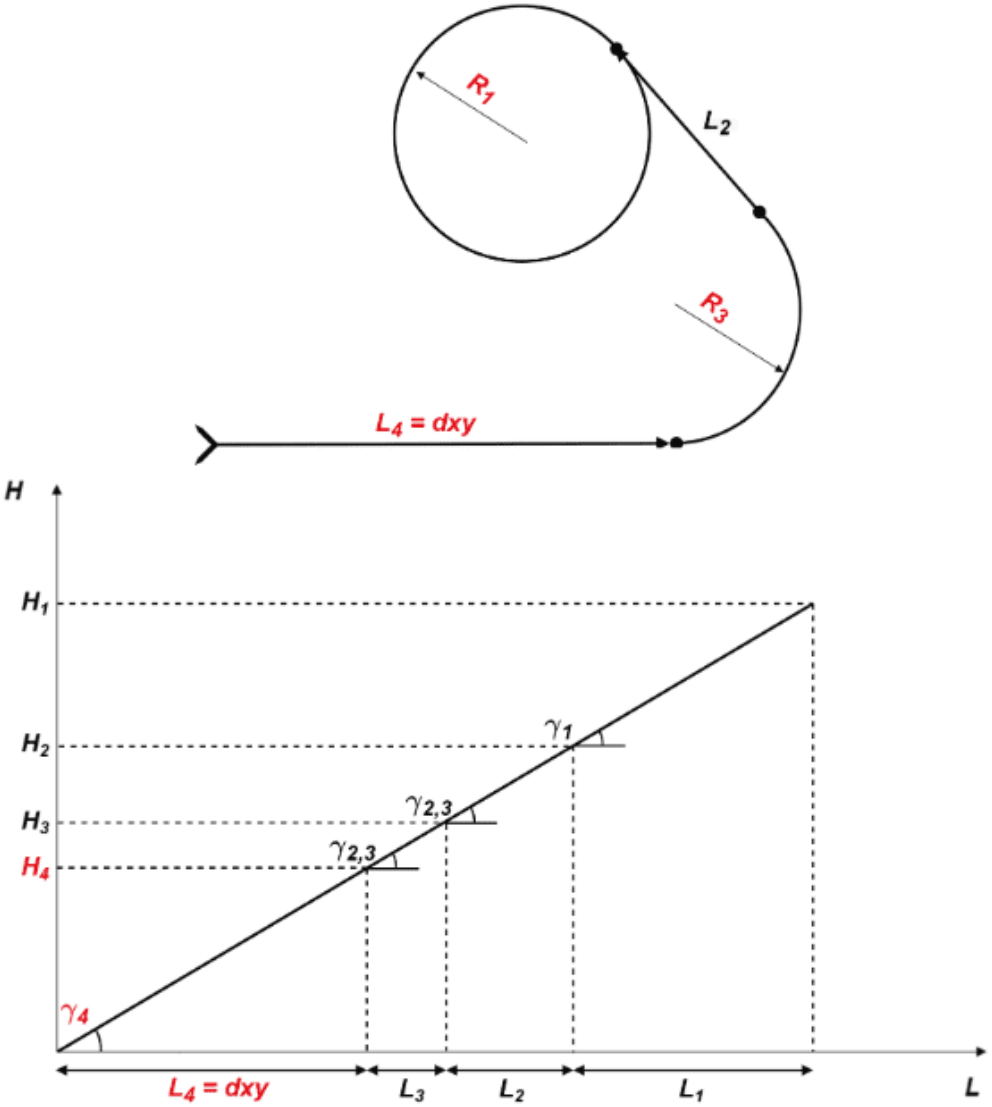


Fig. 294: **Climb route top and front views with parameter identification**

7. **Route**: Here is where the user can set some of the the climbing path parameters (those highlighted in red on the above diagram).

First, the user-defined parameters are described and then, some considerations on the behavior of the climb algorithm are explained:

- **Taxi extension**: This parameter does not apply to this algorithm.
- **Horizontal extension (dxy)**: Absolute ground distance of the first path, L_4 . From the start of the climb to the start of the turn that faces the loiter path.

  This distance will remain fix always and it will also fix L_4 path's final point height, H_4 . More information below.
- **Radius Head Turn (R3)**: Radius of the turn to head the platform towards the loiter.
- **Radius loiter (R1)**: Radius of the ascending helix path to reach the loiter height.
- **Flight Path Angle**: The FPA (γ_4) is the angle at which the aircraft will climb. Before the algorithm execution, all Flight Path Angles, γ_i , are equal: $FPA = \gamma_4 = \gamma_{2,3} = \gamma_1$. The algorithm can modify $\gamma_{2,3}$ and γ_1 . In that case, the Flight Path Angle option will serve as the **upper threshold**.

Note: The rest of the parameters shown in the figure above are calculated automatically by the algorithm ($L_1, L_2, L_3, H_1, H_2, H_3, \gamma_1, \gamma_{2,3}$).

Each of these parameters can be entered manually or linked to an Operation Guidance defined by the user clicking on ⚙ or 🔗.

Climbing guidance parameters behavior

The climbing track is not fix, the algorithm recalculates the paths each time to take into account the aircraft position and the user's parameters. The trajectory usually has 4 paths, excluding the final loiter path:

- **General trajectory description**:
  - L_4 is the first path. The user can set the horizontal length, dxy , the direction (in Runway direction) and the path's final point height, H_4 with the defined Flight Path Angle. This is very relevant, as seen later. **The path length can not be zero**.

$$\gamma_4 = FPA$$
$$H_4 = dxy \cdot \tan(\gamma_4)$$

  - L_3 : Circular turn to head the platform towards the loiter. The user can set the radius, R_3 . It is **possible to set this path to zero** clicking on 🔗⚙. The FPA, $\gamma_{2,3}$, can be modified by the algorithm.
  - L_2 : Straight path that reaches the climbing loiter point. This path is **completely automatic generated**. Its FPA, $\gamma_{2,3}$, can be modified by the algorithm.
  - L_1 : Ascending helix path to reach the loiter height. The user can set the radius, R_1 .

- **Loiter height effect**: Loiter height's, H_c , modifies the algorithm general behavior. Depending on whether H_c is bigger or smaller than H_2 or smaller than H_4 the algorithm will modify some parameters, in particular the flight path angles:
  1. $H_2 < H_c$: This is the general case, in this situation no corrections will be applied as shown below and $\gamma_{2,3} = \gamma_1 = FPA$.

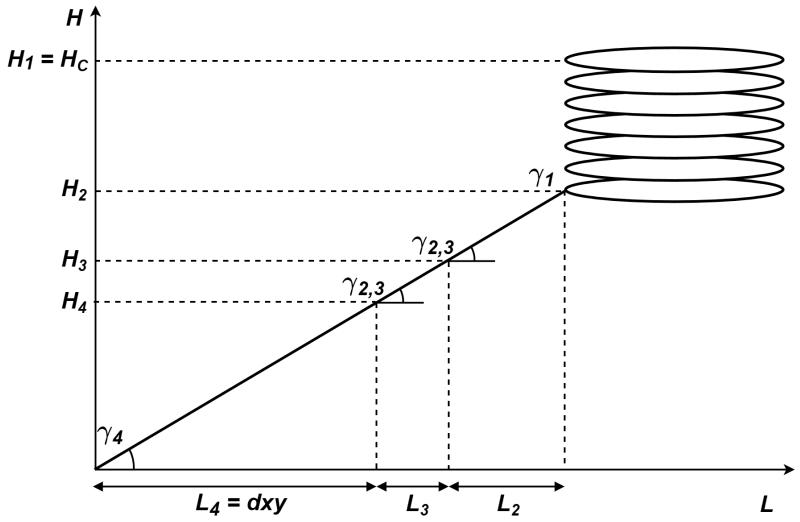


Fig. 295: **Climbing heights when** $H_2 < H_c$

2. $H_4 < H_c < H_2$: In this case, the algorithm will compute a new $\gamma_{2,3}$ to avoid surpassing the loiter's height and γ_1 will be zero.

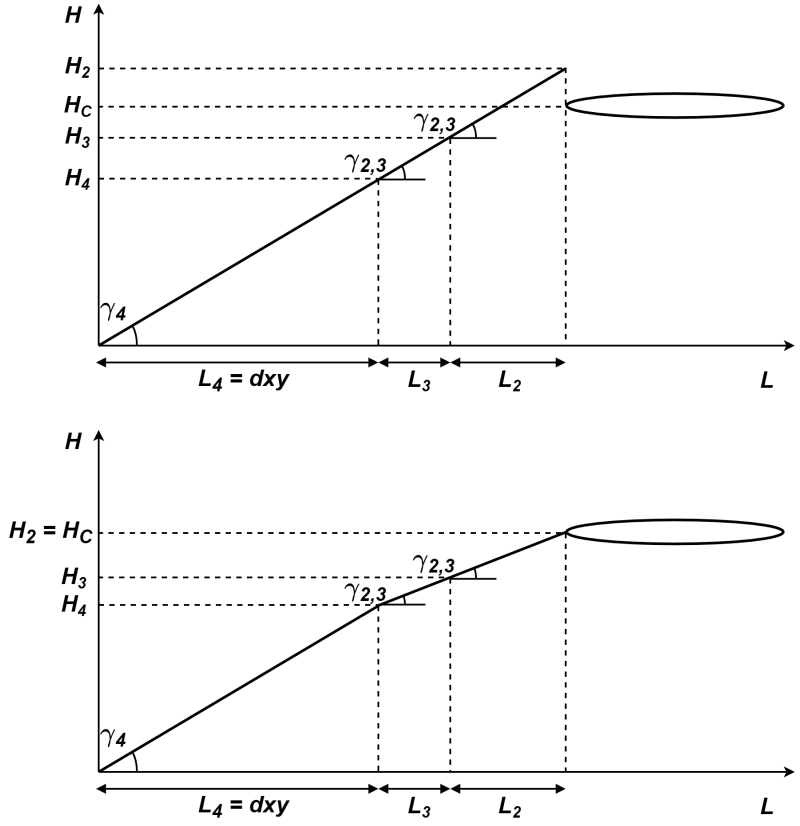


Fig. 296: **Climbing heights when** $H_4 < H_c < H_2$

3. $H_c < H_4$: In this case, the algorithm will force $H_c = H_4$. So H_4 **will be the new loiter height** keeping $\gamma_4 = FPA$ and the other flight paths angles equal to zero, $\gamma_{2,3} = \gamma_1 = 0$.

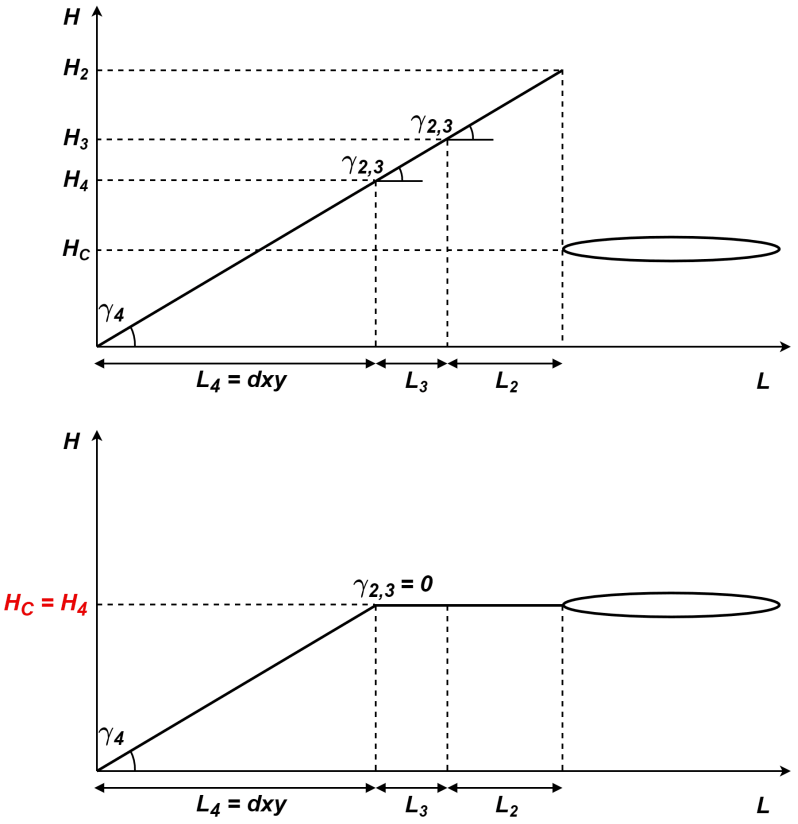


Fig. 297: **Climbing heights when** $H_c < H_4$

2.9.5.3 Cruise

This phase is used to make the aircraft follow a position-based route created by the user in **Veronte Ops** (for more information, see Veronte Ops manual). This is the principal use of this guidance algorithm, but it can also be used to make the aircraft go to a certain location (e.g, a waypoint) without indicating the complete route, thus being a guidance used to command a movement by position.

Cruise guidance generates a three-dimensional trajectory.

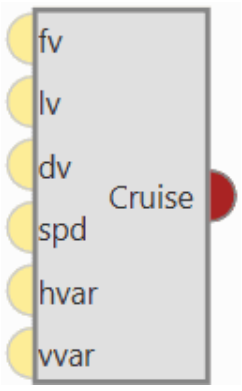


Fig. 298: **Cruise block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Configuration menu**:

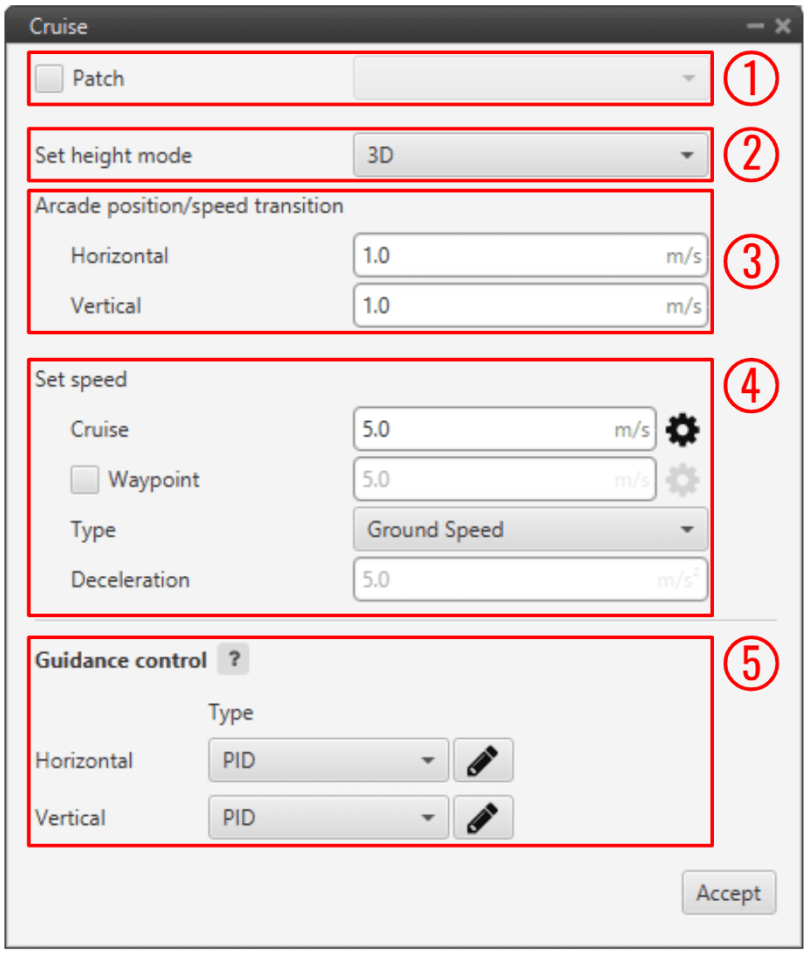


Fig. 299: **Cruise block configuration**

All the parameters that define the cruise guidance are detailed.

1. **Patch**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
2. **Set height mode**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
3. **Arcade position/speed transition**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
4. **Set speed**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
5. **Guidance control**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.

2.9.5.4 Envelope

Envelope block is used to configure the flight envelope of the aircraft. Here the limits that will not be exceeded during the operation are set.

These limits are respected by the guidance and depend on how the control is implemented.

> **Warning:** Although the acceleration has been limited here, if the control is configured so that the pilot in manual mode can control the pitch angle, this acceleration limit will have no effect.

Fig. 300: **Envelope block**

> **Warning:** In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Output**

  Pin 0: Flight envelope configuration.

Important: If users wish to have more than one envelope in their configuration, they must configure as many envelope blocks as they wish and make the connection to the *Guidance Computation block* through a block of type *Swicth*.

To do this, the desired conditions must be set for the Autopilot 1x to make the switch in the **Switch** block.

An example of how to implement more than one envelope is shown below:

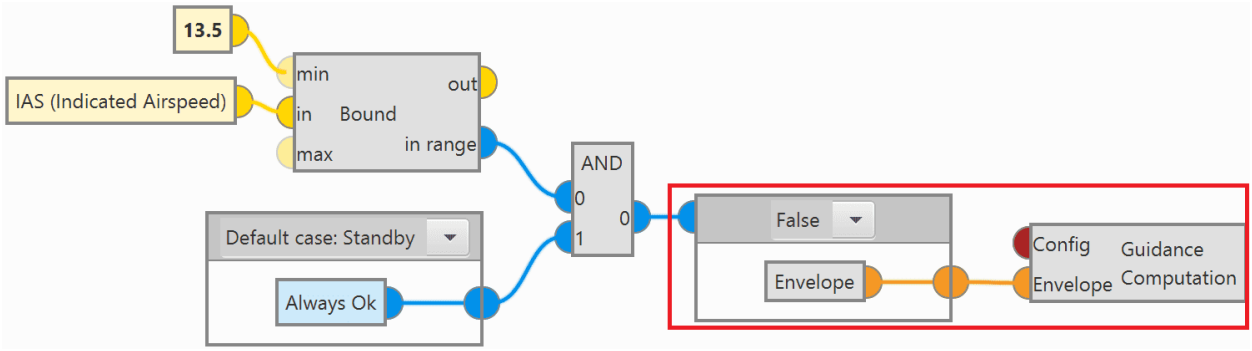


Fig. 301: **Example of more than one Envelope block**

- **Configuration menu**: The configuration of this block is divided into 3 different tabs:

  Envelope

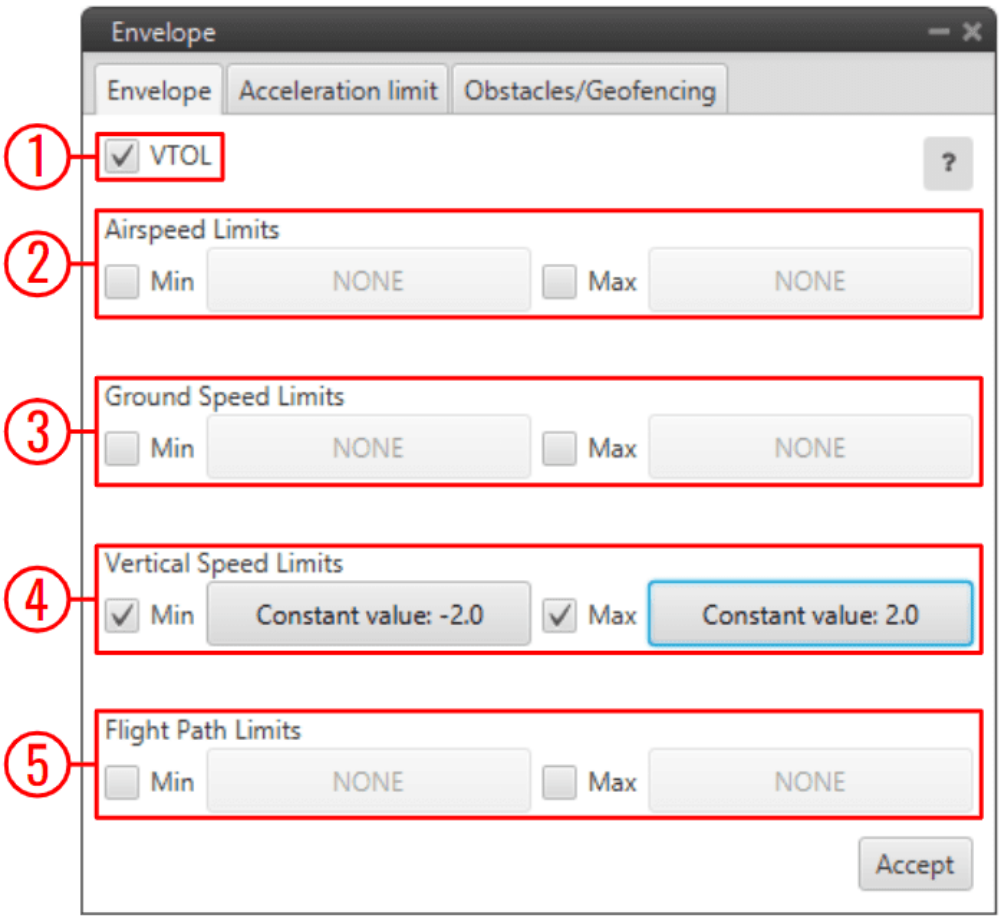


Fig. 302: **Envelope block configuration - Envelope**

1. **VTOL**: If this option is enabled, when the vehicle reaches the last waypoint of its route (and it is an open path), a hover is done instead of a loiter point.
2. **Airspeed Limits**: Limits for the indicated airspeed (IAS). Values indicated here have effect over the "Cruise" guidance, but is **overrided** if there is a **Hold command** on the IAS, so the user must be careful with the velocity commands.
3. **Ground Speed Limits**: Minimum and maximum ground speed of the platform. In case of strong wind, these parameters set the minimum GS that the aircraft can reach, for lower values than this one the thrust will be automatically increased to gain speed and avoid a point where the platform is stopped in the air.
4. **Vertical Speed Limits**: Similar to the previous limit. It sets the minimum and maximun value for the vertical speed of the platform.
5. **Flight Path Limits**: Maximum and minimum values for the flight path angle (angle of climb or descent).

Note:

- Users can manually enter values or select variables instead.
- These limits will modify how guidances command your vehicle to move, but will not apply when the aircraft is not being controlled by a guidance (i.e manual/assisted flight).
- These values should not represent the maximum limits that the platform can withstand, but the maximum values that should be present during a normal operation.

Acceleration limit

In this second tab there are more options to fix the limits (positive and negative direction) of acceleration and jerk in SI units.

Acceleration limits are applied in any phase with position guidance. They modify the desired velocity. The algorithm compares the current desired velocity with thatstored in previous step. There are six values to define.

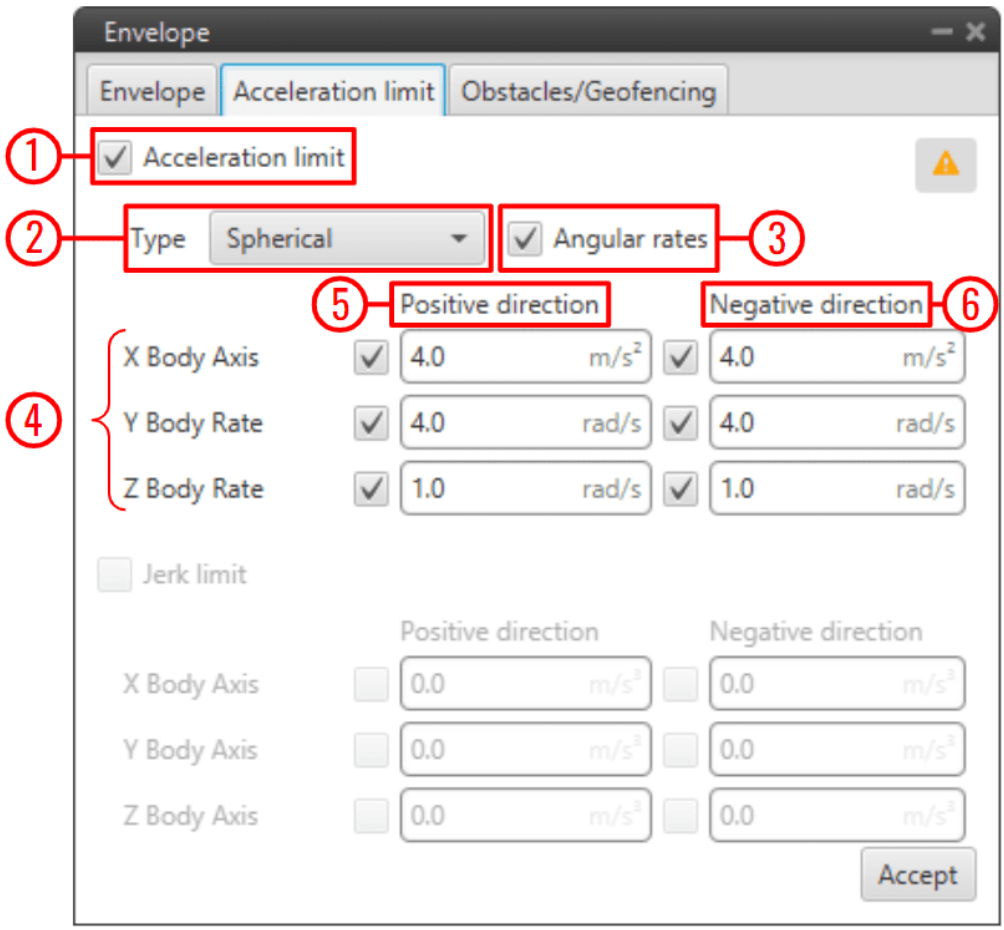


Fig. 303: **Envelope block configuration - Acceleration limit (Spherical frame)**

The configurable options in this menu are:

1. Enable/Disable the acceleration limit.
2. **Type**: User can choose between **Cartesian Body** and **Spherical**.

   Cartesian Body is normally used for **multicopters** or **aircraft that allow 3-dimensional movement**, while the **Spherical** type is used for **conventional aircraft**.
3. **Enable/Disable angular rates**: The user also can selected directly angular rates option which allows him set limits in Y body rate and Z body rates.
4. **Axes**:

  - In **Cartesian Body** the axes refer to the body frame (X Body Axis, Y Body Axis, Z Body Axis).
  - In **Spherical** type, the algorithm internally applies limits in module, inclination and azimuth to maintain the limits set by the user in body frame (body frame limits will be turn into spherical limits).

5. **Positive direction**: The limit for acceleration. If desired velocity has the same sign as in the previous step, and it is lower (in absolute value) this limit is applied.

   For example, if a multicopter is flying in negative X direction, and it has to increase desired velocity in same direction this limit will be applied.

6. **Negative direction**: The limit for deceleration. In the case positive direction limit is not used.

The second derivative of velocity (**Jerk**) imposes another limit in acceleration. It modifies the behavior of the vehicle.

Depending on values, it's possible to get more smoothness during guidance. Algorithm ensures that when desired velocity is reached the acceleration value is near 0. As acceleration limits, user can set 6 values (3 for positive direction limit and 3 for negative direction limit).

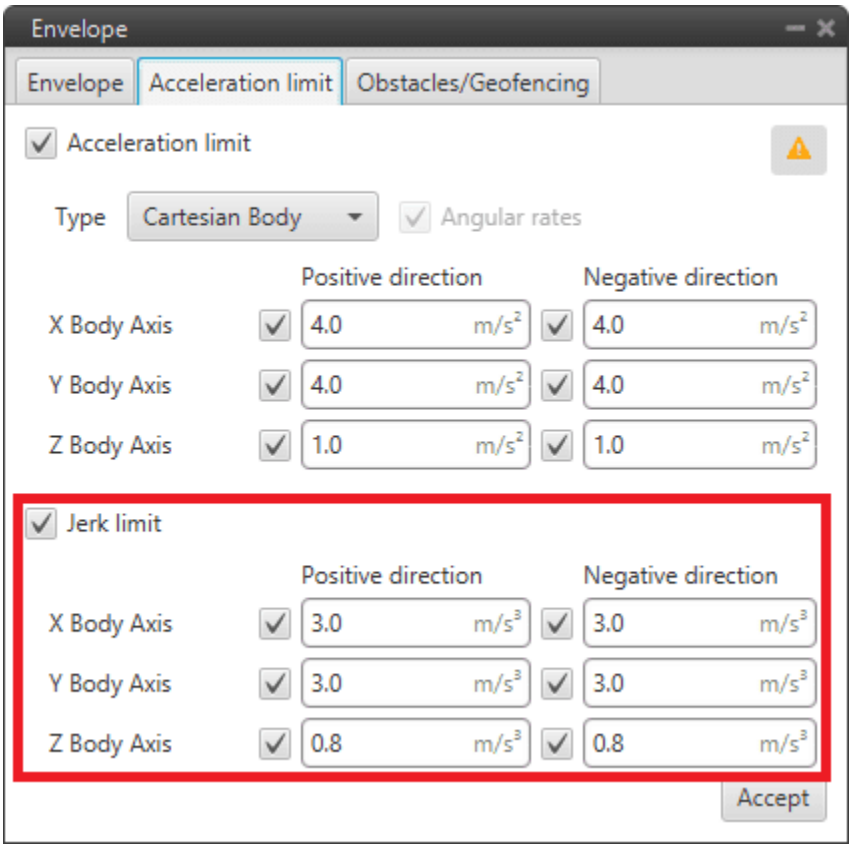


Fig. 304: **Envelope block configuration - Acceleration limit (Cartesian frame)**

The effects of these limitations are explained below.

First, the acceleration limits are disabled. The stick that controls Thrust is moved and desired velocity change according to this stick command. In Desired velocity chart we can see this effect and in bottom acceleration chart is shown how acceleration is not limited (hight values reached).

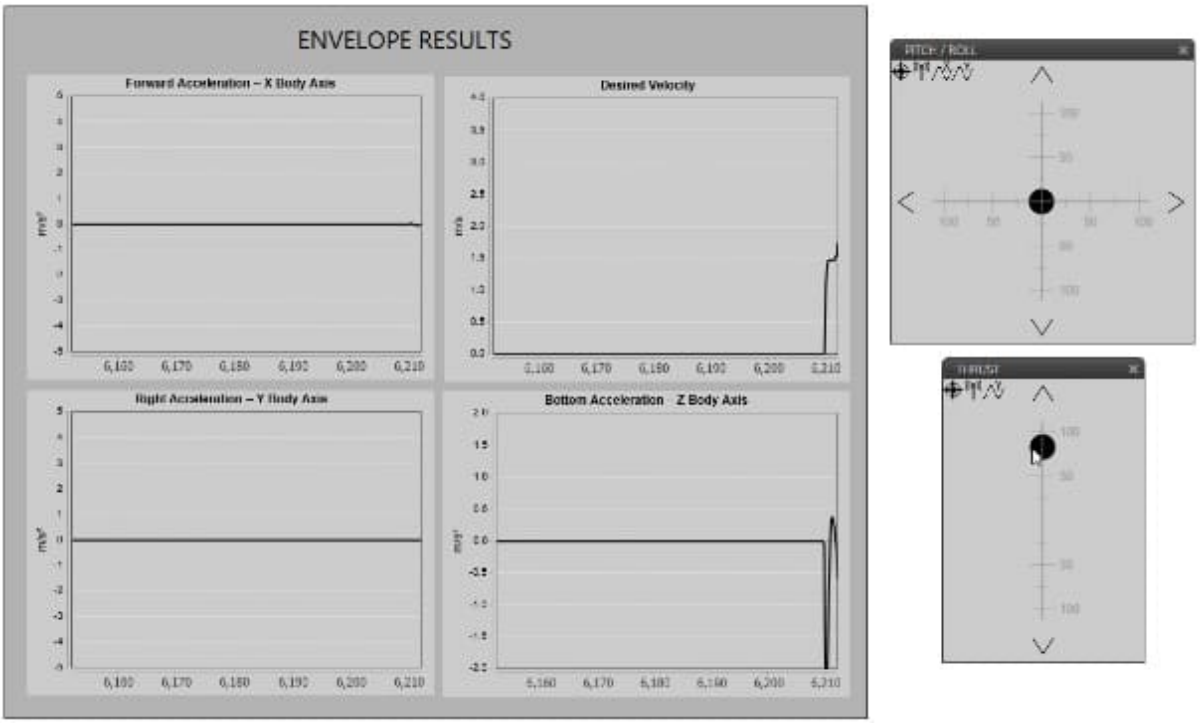


Fig. 305: **Only velocity limit (Thrust)**

Now, a limit in acceleration bottom axis is set to 0.1. Now the desired speed grows with a lower slope due to the imposed limitation. Also in the acceleration bottom chart we can see how the value oscillates within the imposed limit.

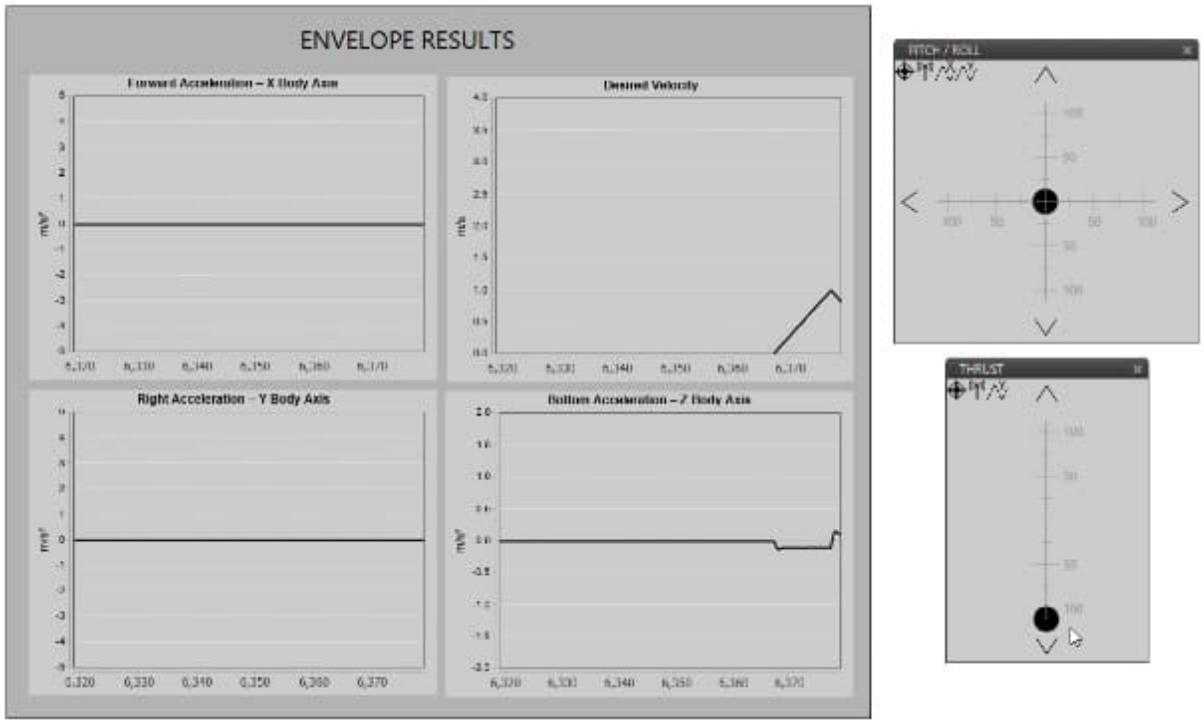


Fig. 306: **Acceleration limit (Thrust)**

To compare acceleration and jerk, roll axis is chosen. In the first gif below only the first limit is applied.

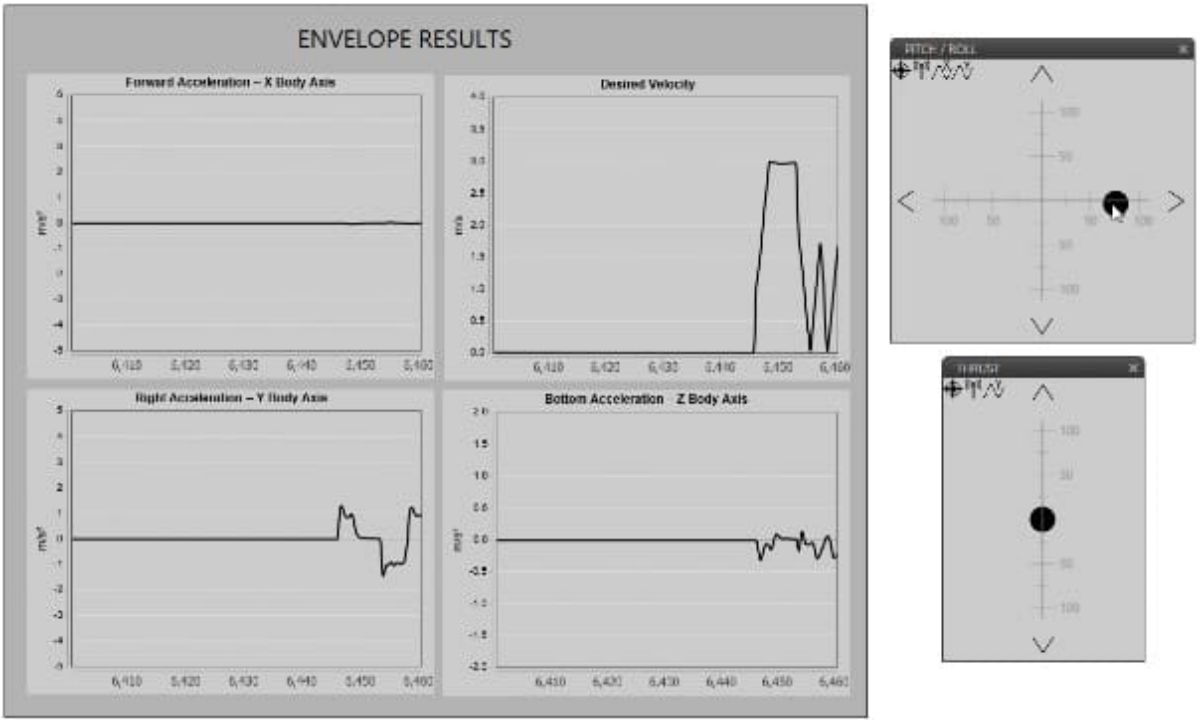


Fig. 307: **Acceleration limit (Cartesian frame)**

When the jerk limit is also enabled we can see how acceleration (in Y Body axis) does not show peaks, and changes in desired velocity are smoother.

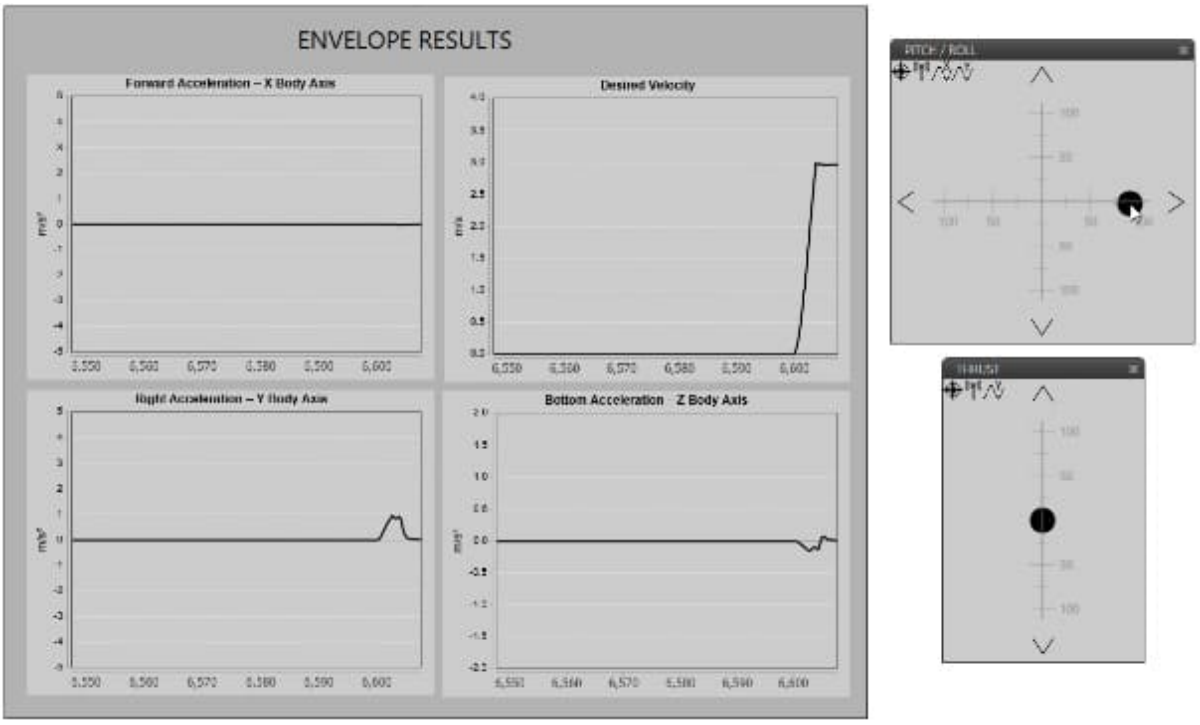


Fig. 308: **Acceleration limit (Cartesian frame)**

Obstacles/Geofencing

Users should configure this last tab when using the **geofencing** functionality of **Veronte Autopilot 1x**.

The **maximum deceleration** desired when approaching the obstacle can be entered manually or by selecting a variable.

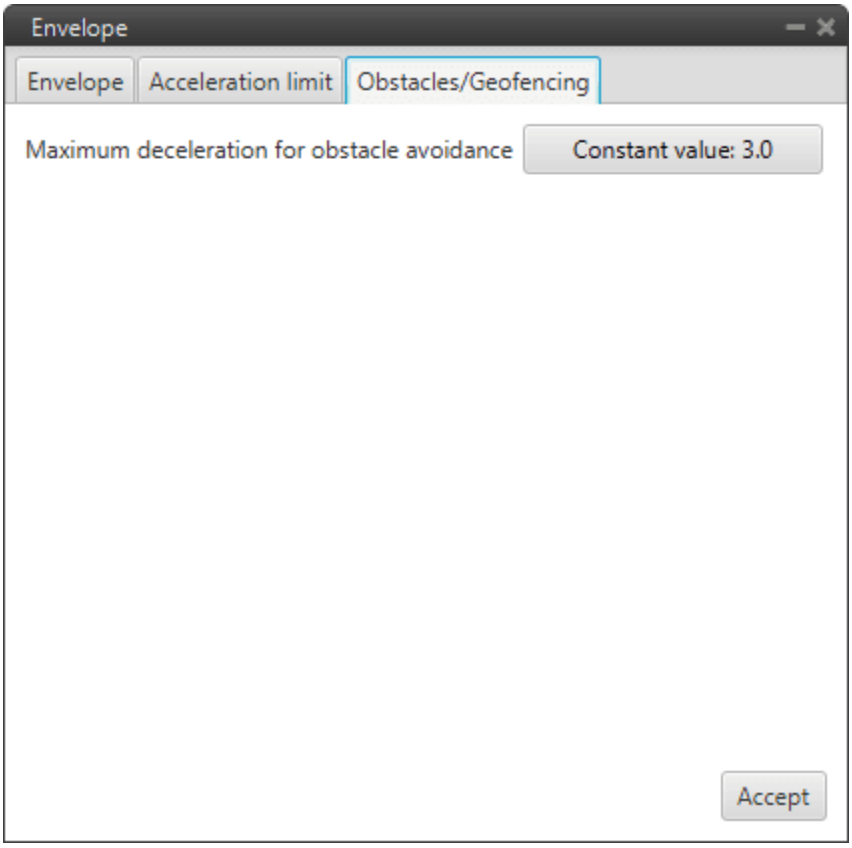


Fig. 309: **Envelope block configuration - Obstacles/Geofencing**

2.9.5.5 Guidance Computation

Guidance Computation block takes the configuration and arcade data from a given type of guidance and computes the guidance parameters. It is always necessary to add it with these Guidance blocks: *Climb*, *Cruise*, *Landing*, *Rendezvous*, *Taxi* and *VTOL*.

In additon, an *Envelope block* must also be connected to this block to compute the flight envelope of the aircraft in the guidance.

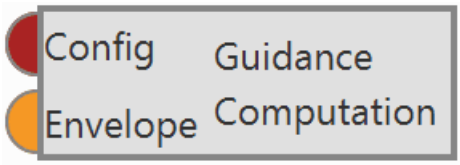


Fig. 310: **Guidance Computation block**

- **Inputs**
  - **Config**: Guidance configuration.
  - **Envelope**: Flight envelope.

2.9.5.6 Landing

Landing guidance is used to generate the flying path the aircraft will follow when landing on a certain runway.

Landing guidance generates a three-dimensional trajectory.

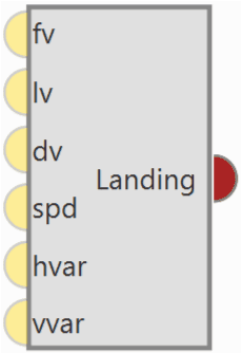


Fig. 311: **Landing block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Configuration menu**:

  The generated path is not directly indicated by the user as in cruise guidance (which is defined in **Veronte Ops**), instead a trajectory is generated based on the parameters detalied later in this section, as in climb guidance.

  Below, find all the information to be defined by the user together with the corresponding Figures showing the location of these parameters in the menu.

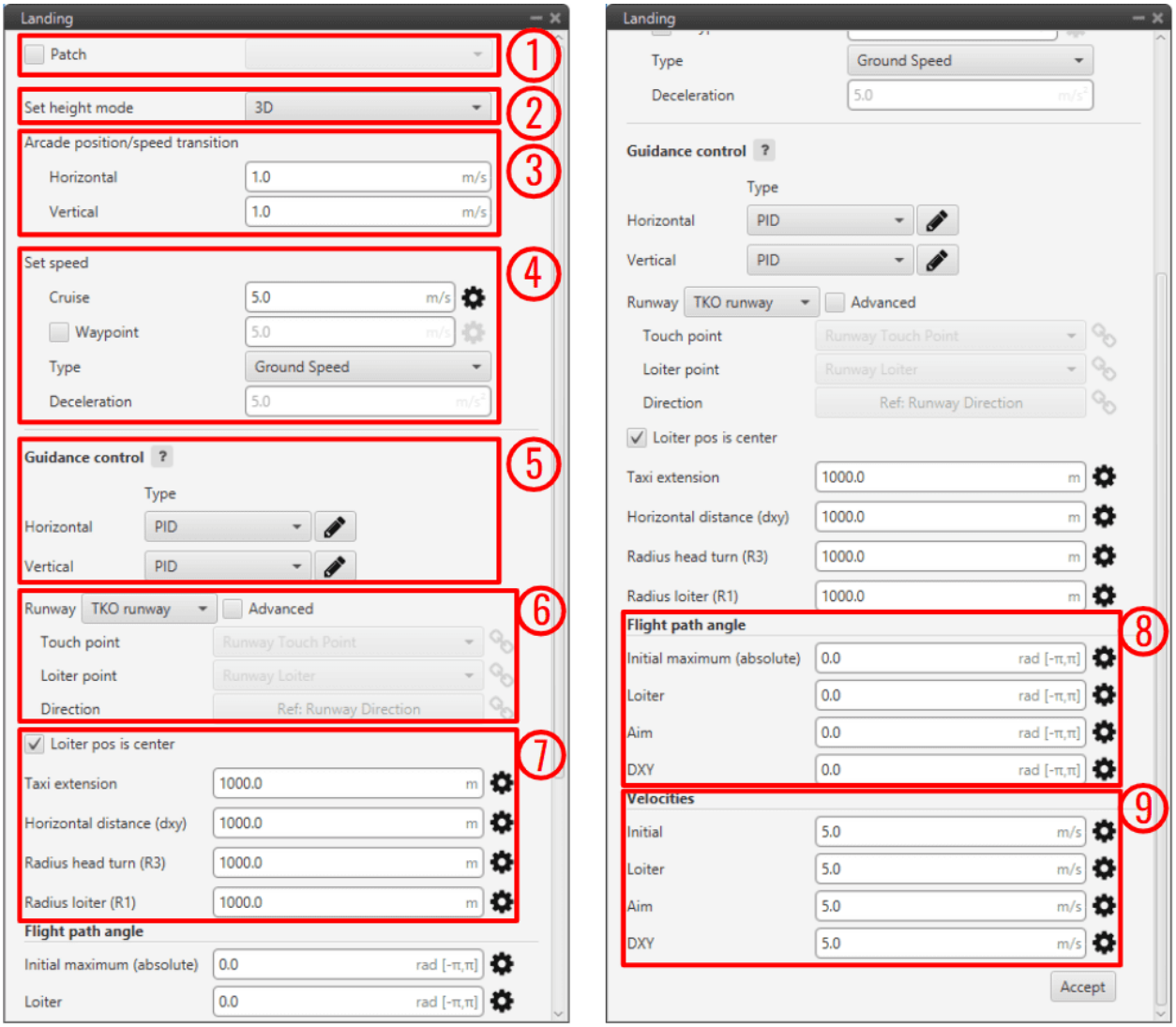
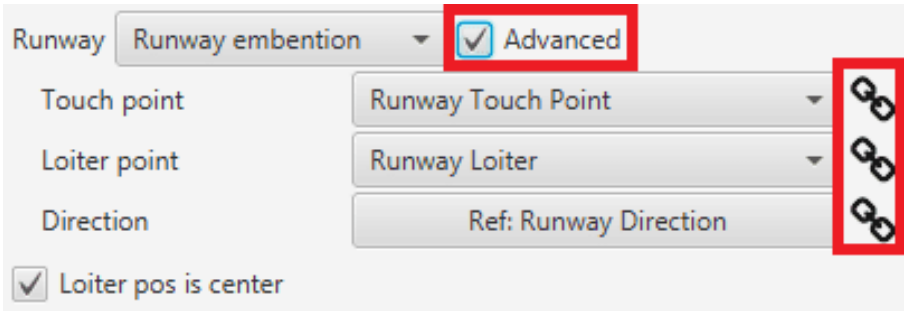


Fig. 312: **Landing block configuration**

1. **Patch**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
2. **Set height mode**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
3. **Arcade position/speed transition**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
4. **Set speed**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
5. **Guidance control**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
6. **Runway and Loiter position**: Here the user can define the loiter and runway positions and direction. However, the default option is to define them in the **Runway option** of **Veronte Ops** (for more information, see Veronte Ops manual).

If the *Advanced* option is chosen, then the user can define three parameters. By clicking on  or  different options will be displayed:



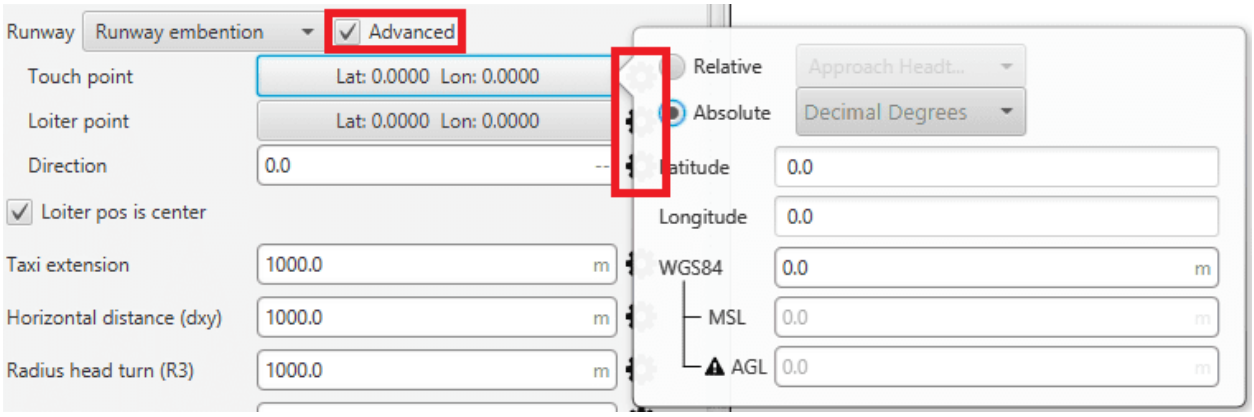


Fig. 313: **Runway and Loiter Position Options**

– **Touch point**: Defines the touch point of the runway. The user can configure it in 2 different ways:

  *  icon selected: By default, this point is the runway's touch point. But the user can select in the drop-down list any other previously defined point. This includes waypoints defined in the Mission (in **Veronte Ops**), among many others.

  *  icon selected: Alternately, the user can manually define this point. Then it can be configured in two ways:

    · **Relative**: In this case, the position of the point is relative to another point. That point could be any platform fitted with an Autopilot 1x.

    · **Absolute**: The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress º ” ‘. They are indicated through the latitude, longitude and altitude (being possible to define this last one with respect to the ellipsoid, WGS84, to the sea level, MSL or to the ground, AGL).

– **Loiter point**: Defines the loiter point. The two available options are:

  *  icon selected: By default, this point is the runway's loiter. But the user can select in the drop-down list any other previously defined point. This includes waypoints defined in the Mission (in **Veronte Ops**), among many others.

    * ⚙ icon selected: Alternately, the user can manually define the Loiter point. Then it can be configured in two ways:
        · **Relative**: In this case, the position of the point is relative to another point. That point could be any platform fitted with an Autopilot 1x.
        · **Absolute**: The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress º ” ‘. They are indicated through the latitude, longitude and altitude (being possible to define this last one with respect to the ellipsoid, WGS84, to the sea level, MSL or to the ground, AGL).

  – **Direction**: Defines the runway direction. Again, there are two available options:
    * 🔗 icon selected: By default, it is the same as the selected runway. It can be also chosen from a list of options including runway direction, tailwind direction, etc.
    * ⚙ icon selected: Alternately, it can also be defined as an angle with respect to the magnetic north.

  – **Loiter pos is center**: If this box is enabled, the defined loiter point will be the center of the loiter circular trajectory. In case of not, the circular loiter trajectory will pass through that point.

7. **Trajectory distances**: Here the user defines some of the trajectory distances. This distances match the trajectory patches lengths L or are proportional to them. See the explanation below for more information on every patch.
   – **Taxi extension**: Distance from touchdown to where the aircraft is brougth to a full stop.
   – **Horizontal extension (dxy)**: Distance before the head of the runway. At the end of this length, touchdown is expected.
   – **Radius Head Turn (R3)**: Radius of the last turn in order to face the runway direction ($L_3 \propto \pi R_3$).
   – **Radius loiter (R1)**: Radius of the descending loiter for the aircraft to reach an altitude suitable to perform the landing manoeuvre ($L_1 \propto \pi R_1$).

   Each of these parameters can be entered manually or linked to an Operation Guidance defined by the user clicking on ⚙ or 🔗.

   Note: Some patches don't have an associated user-defined distance, and are automatically calculated by the landing guidance algorithm, as they depend on some of the above distances and other parameters defined below.

8. **Trajectory flight path angles**: Here the user defines the desired trajectory flight path angles for each of the patches of the trajectory. See the explanation below for more information on every patch.
   – **Initial maximum (absolute)**: Desired flight path angle γ_0 of patch 0.
   – **Loiter**: Desired flight path angle γ_1 of patch 1.
   – **Aim**: Desired flight path angle $\gamma_{2,3}$ of patches 2 and 3.
   – **DXY**: Desired flight path angle γ_4 of patch 4.
9. **Trajectory velocities**: Here the user defines the desired trajectory velocities for each of the patches of the trajectory. See the explanation below for more information on every patch.
   – **Initial**: Desired velocity v_0 of patch 0.

  - **Loiter**: Desired velocity v_1 of patch 1.
  - **Aim**: Desired velocity $v_{2,3}$ of patches 2 and 3.
  - **DXY**: Desired velocity v_4 of patch 4.

  Each of these parameters can be entered manually or linked to an Operation Guidance defined by the user clicking on ⚙ or 🔗.

The generated trajectory of the landing guidance defines the route that the aircraft follows from the point when the phase with this guidance is entered, to the point where it touches the ground, see the Figure below. The landing route has two parts, being decomposed into 6 patches:

- **First part**: Descending loiter used to descend from the cruise altitude to an altitude where the heading manoeuvre towards the runway can be performed.
  - **Patch 0**: This patch is generated from the point the landing phase is entered to where the loiter is located. Variables that influence this patch are γ_0, v_0 , altitudes H_0 and H_1 , and **Loiter point** position.
  - **Patch 1**: The patch length (L_1) will depend on the amount of loops on the loiter. The latter can go from 0 to more than 1 loop, depending on the altitude necessary to descend/ascend. Variables that influence this patch are γ_1, v_1, altitudes H_1 and H_2 , and **Radius loiter (R1)**.

  The loiter exiting point altitude is computed so that patches 2 to 5 can be performed following their desired v and γ. So it exists the possibility of starting the landing manoeuvre at a lower altitude thant the exiting point of the loiter. In that case, the loiter would be used to ascend.

  If the aircraft starts the landing phase at an altitude similar to the one of the loiter (defined in point 7), then the loiter patch is simplified into a turn (during the turn the altitude can still be adjusted) and the turn's length will depend on the latter.

- **Second part**: Final approach of the landing, which consists on turning, facing the runway and touchdown.
  - **Patch 2**: Patches 3 and 4 need to match the distances defined above. Patch number 2 will conect the exit of the loiter patch with the beginning of patch 3. Variables that influence this patch are $\gamma_{2,3}, v_{2,3}$, altitudes H_2 and H_3, **Loiter point** and **Touch point** positions.
  - **Patch 3**: Turning of the aircraft to face the runway. Variables that influence this patch are $\gamma_{2,3}, v_{2,3}$, altitudes H_3 and H_4 and **Radius Head Turn (R3)**.
  - **Patch 4**: At the end of the patch the aircraft lands. Variables that influence this patch are γ_4, v_4 , altitude H_4 and **Horizontal extension (dxy)**.
  - **Patch 5**: Taxi extension for the aircraft to slow down.

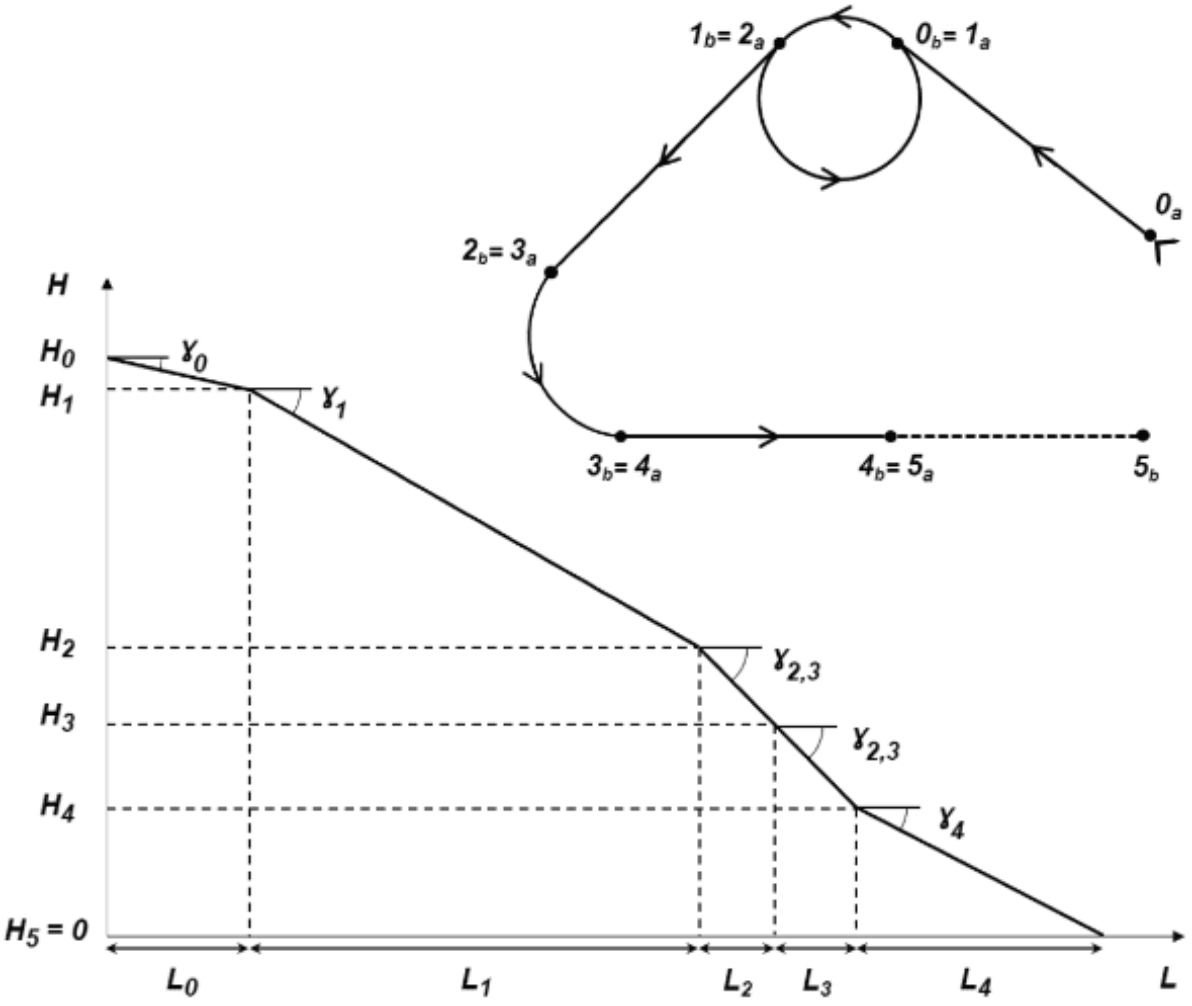


Fig. 314: **Landing route top and front views with parameter identification**

2.9.5.7 Rendezvous

Rendezvous guidance is used to create a meeting point where the 1x air unit will approach a second unit (either another air or base) within a determined offset.

This guidance updates constantly the vehicle attitude in order to track, with the shortest path, the position of the second unit (named as Base hereafter). This guidance works for both **static** and **moving Base**.

Rendezvous navigation is ready for taking **Internest** input to improve the precision from its guidance, being the most suitable kind for Internest integration.

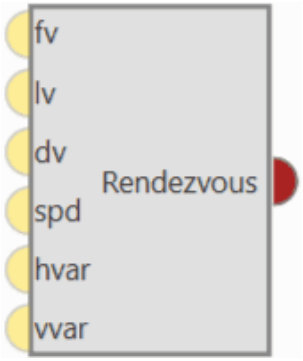


Fig. 315: **Rendezvous block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Configuration menu**:

  All the parameters that define the rendezvous guidance are detailed.

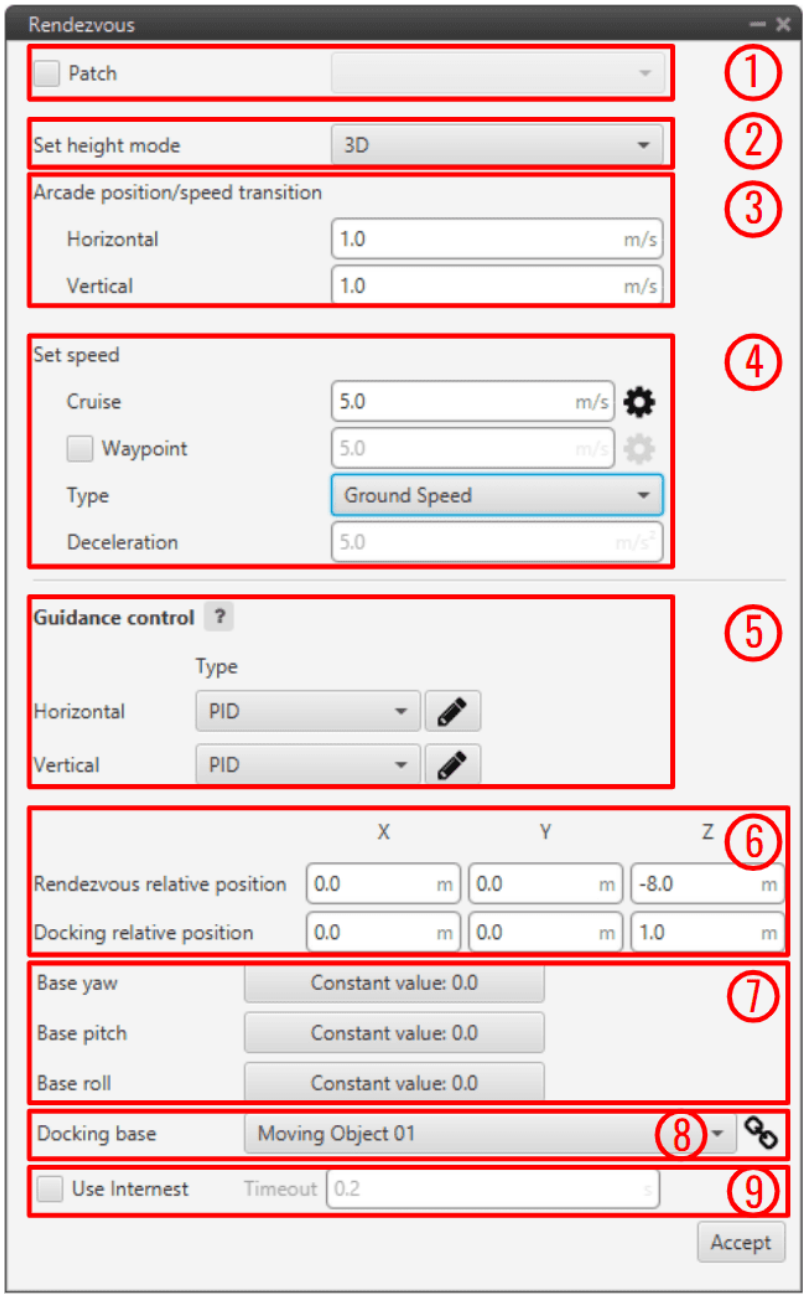


Fig. 316: **Rendezvous block configuration**

1. **Patch**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
2. **Set height mode**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
3. **Arcade position/speed transition**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
4. **Set speed**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.

5. **Guidance control**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
6. **Relative position**: For both Docking and Rendezvous the axes are set according to Autopilot 1x orientation (for more information about 1x orientation visit Orientation - Hardware Installation section of the **1x Hardware Manual**).
   - **Rendezvous relative position**: **3D point** used **to configure the meeting point for the 1x air unit**. This point will be tracked by the vehicle and, once reached, it will start travelling to Docking relative position. For **VTOL**, **X** and **Y components must be equal**.
   - **Docking relative position**: **3D point** used **to configure the offset for the approaching vehicle to the Docking base**. This will be the difference from GNSS position that defines the landing point.

   > **Warning:** Usually, the **docking relative position** is set by slightly **overlapping** the **'Docking base'** in order to ensure that the 1x air unit reaches it, as can be seen in the figure below.

7. **Base yaw, pitch & roll**: Defines the attitude from the Base body. These values affect the navigation by orienting the air unit to be equal to the attitude from the Base unit. To be configured with **telemetry** (example below).
8. **Docking base**: Defines the position of the GNSS antenna connected to the Base unit. If the guidance is being configured for a 'moving' Base, a *Moving Object* must be assigned to it.
9. **Use Internest**: As Rendezvous navigation is prepared to take Internest input, here the user can enable its use and configure a Timeout.

The following figure gives an overview of some parameters introduced (note that the negative Z-coordinate is due to the Autopilot 1x axes convention):

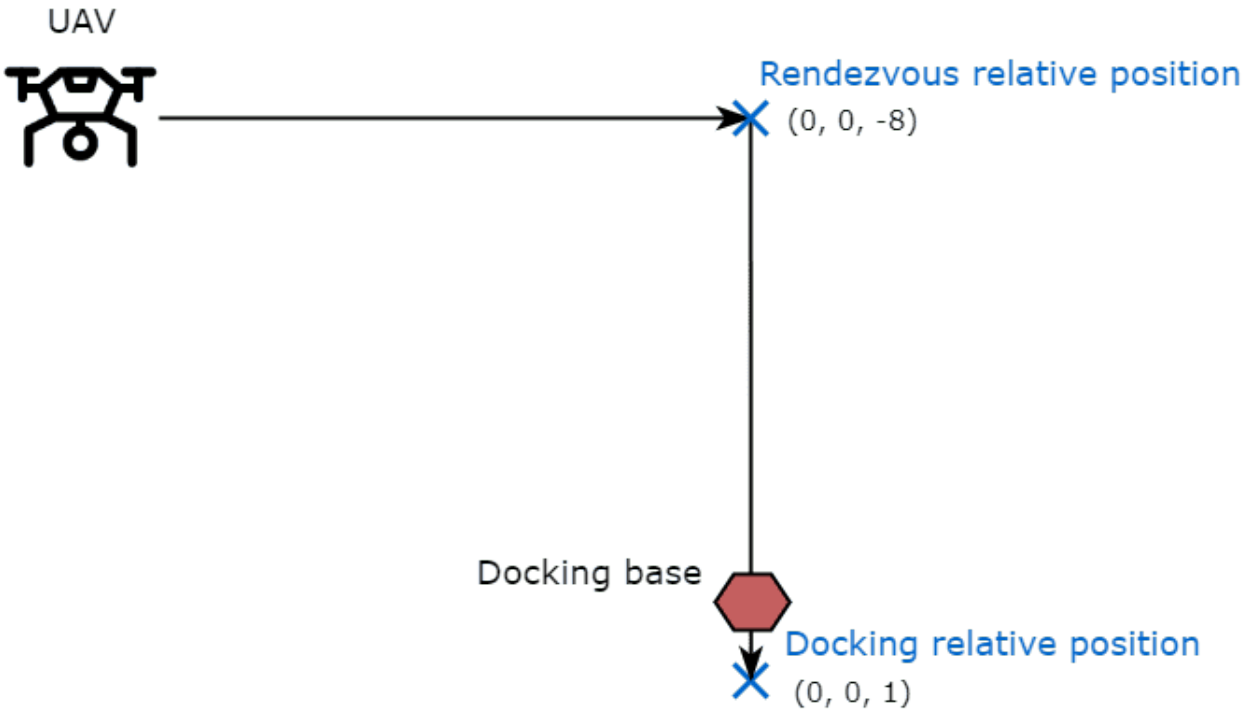


Fig. 317: **Rendezvous guidance parameters**

In order to know how **to configure the Moving Object**, which is assigned to Docking Base and the Base attitude, see *Data transmission between Veronte Autopilots 1x - Integration examples* section of this manual.

Finally, in order **to see the Moving Object position in the Veronte Ops interface**, in the **1x Air unit**:

1. Go to Telemetry menu → **Telemetry** panel → **Data link to VApp**.
2. Add **Moving Object** to the list of variables.

2.9.5.8 Taxi

Taxi guidance is used to create a linear path along the runway that is followed by the aircraft. This command is normally used in the take-off phase, where the airplane is wanted to keep the direction of the runway while is accelerating until the lift-off point.

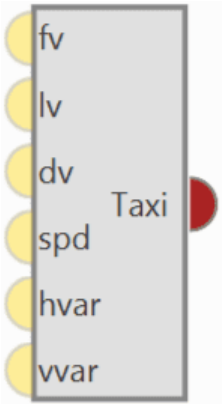


Fig. 318: **Taxi block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Configuration menu**:

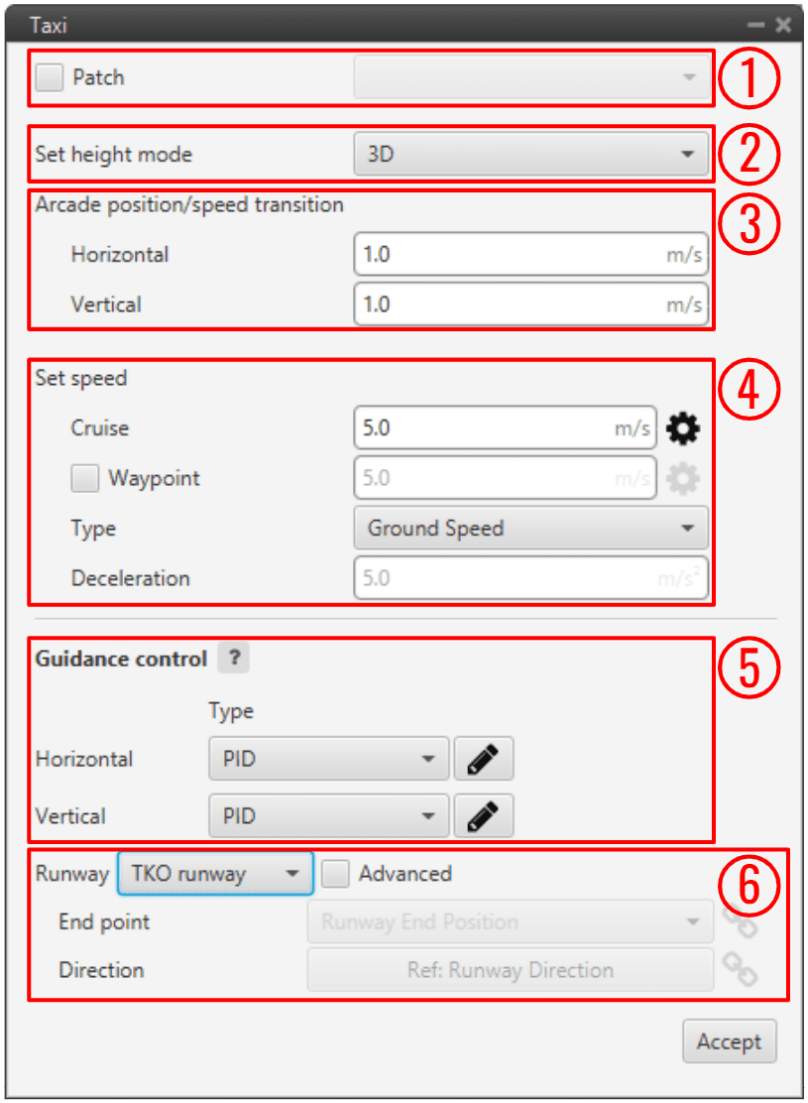


Fig. 319: **Taxi block configuration**

All the parameters that define the taxi guidance are detailed.

1. **Patch**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
2. **Set height mode**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
3. **Arcade position/speed transition**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
4. **Set speed**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
5. **Guidance control**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
6. **Runway**: Here it is selected a runway previously configured, see the **Runway option** of Veronte Ops for more information.

Besides, it is possible to use the *Advanced* mode and select a different end point or direction. By clicking on or different options will be displayed:

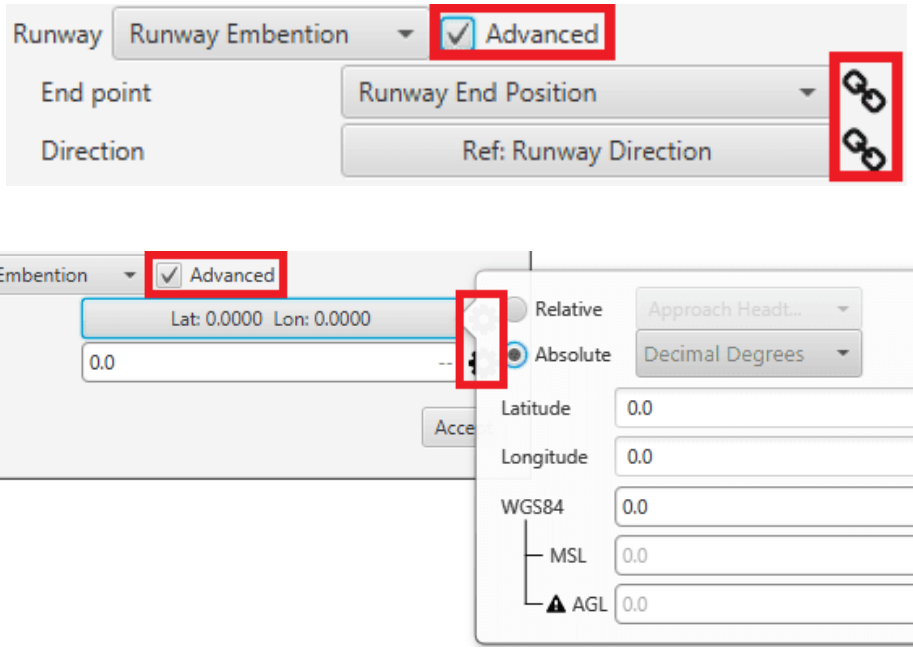


Fig. 320: **Runway parameters**

– **End point**: Defines the end point of the runway. The two available options are:

  * icon selected: By default, this point is the end of the runway. But the user can select in the drop-down list any other previously defined point. This includes waypoints defined in the Mission (in **Veronte Ops**), among many others.

  * icon selected: Alternately, the user can manually define the End point. Then it can be configured in two ways:

    · **Relative**: In this case, the position of the point is relative to another point. That point could be any platform fitted with an Autopilot 1x.

    · **Absolute**: The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress º ” ‘. They are indicated through the latitude, longitude and altitude (being possible to define this last one with respect to the ellipsoid, WGS84, to the sea level, MSL or to the ground, AGL).

– **Direction**: Defines the runway direction. Again, there are two available options:

  * icon selected: By default, it is the same as the selected runway. It can be also chosen from a list of options including runway direction, tailwind direction, etc.

  * icon selected: Alternately, it can also be defined as an angle with respect to the magnetic north.

2.9.5.9 VTOL

VTOL guidance (vertical take-off and landing) is used in **multicopters** for the take-off and landing operations. This guidance consists on the creation of a vertical line that starts at the point where the platform enters in this guidance.

VTOL guidance generates a vertical straight trajectory.

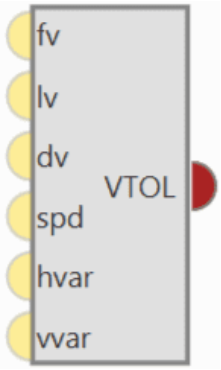


Fig. 321: **VTOL block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidace Computation* block.

- **Configuration menu**:

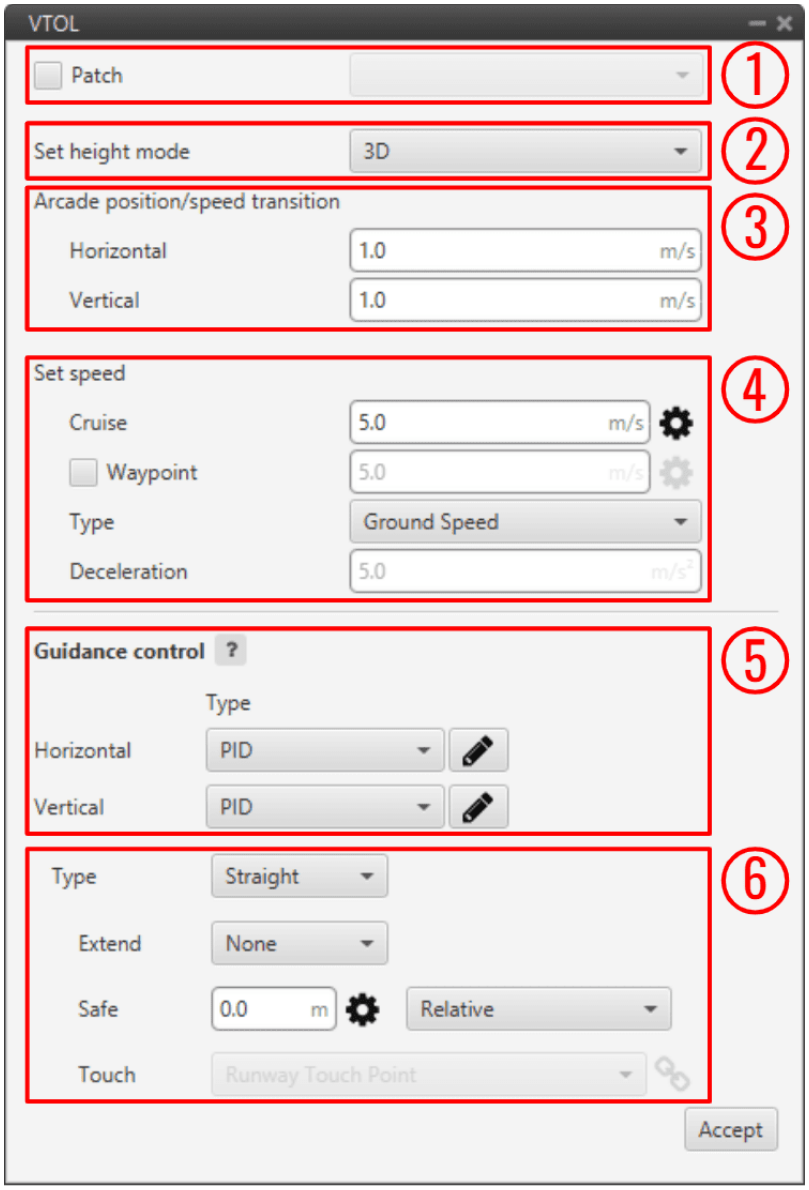


Fig. 322: **VTOL block configuration**

All the parameters that define the VTOL guidance are detailed.

1. **Patch**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
2. **Set height mode**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
3. **Arcade position/speed transition**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
4. **Set speed**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.
5. **Guidance control**: This is detailed in the common guidance block parameters, described at the beginning of this section in *Guidance blocks common configuration*.

6. **Type**: These parameters are used to indicate how the multicopter follows the route during the take-off and landing.
   - The path **Straight** consists on a vertical line from the point where the vehicle enters in this phase. In the case of a take-off, the line goes from the ground to an altitude indicated by the user.
   - The second option, **Hangman**, the path consists on a vertical and horizontal line.

   - **Extend**: When **Up** or **Down** are selected, the value set in Safe will be discard, and the platform will ascend or descend, until a next change.
   - **Safe**: This parameter defines the altitude the aircraft reach. The user can select an Operation Guidance point from the drap-down list (icon selected) or manually enter a value (icon selected), this latter value can be:
     * **Relative**: Starting from the initial point of the route (current platform position).
     * **Absolute altitude**: MSL, AGL or WGS84.

       As an example, in a Take-Off operation, an altitude of -10000 meters can be indicated as the final point of the route, so it is sure that the multicopter will keep climbing until another phase is commended (via automation or manually).

       The same procedure is done in the landing, indicating a big relative distance (for example 100 meters from the starting point), so it is sure that the vehicle reaches the ground, and an automation is set to stop the platform when it touches the surface.

     Note: When the option relative is selected, a positive value will made the aircraft descend. Therefore, this value is Positive down.

   - **Touch**: Additional parameter **to be configured when the type Hangman is selected**. It defines a point that the aircraft has to reach. For instance, after go Up/Down the set value, the aircraft will perform an horizontal movement according to the defined point. Finally, when the aircrafts is over the point, it will descend until reaches that point.

     Usually, this option is used to land at the same point where it took-off (**Return to Take-Off point**) or when there are obstacles in the area and by performing this movement the platform can avoid them and land safely.

     There are 2 ways to configure it:
     * icon selected: By default, this point is the touch point of the runway. But the user can select in the drop-down list any other previously defined point. This includes waypoints defined in the Mission (in **Veronte Ops**), among many others.
     * icon selected: Alternately, the user can manually define this point. Then it can be configured in two ways:
       - **Relative**: In this case, the position of the point is relative to another point. That point could be any platform fitted with an Autopilot 1x.
       - **Absolute**: The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress º ” ‘. They are indicated through the latitude and longitude.

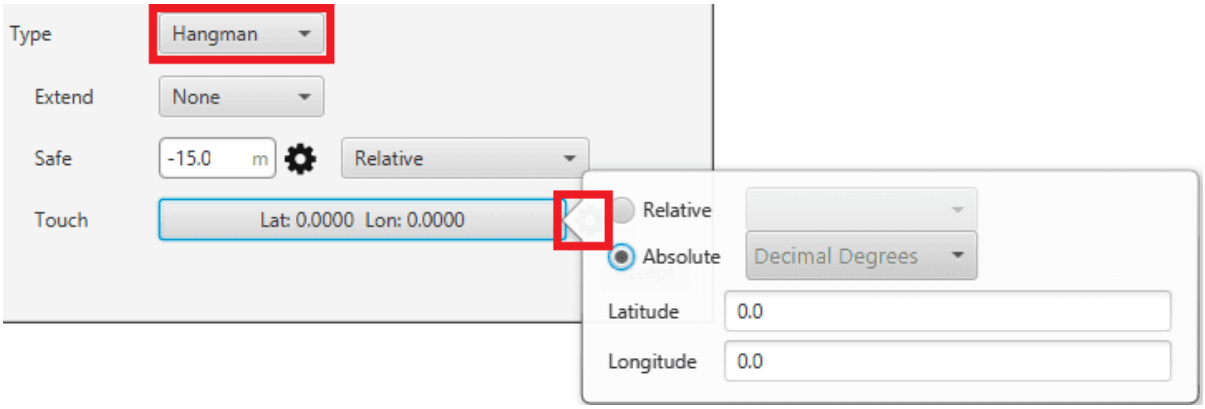


Fig. 323: **Touch options**

The following image gives an overview of some parameters introduced:

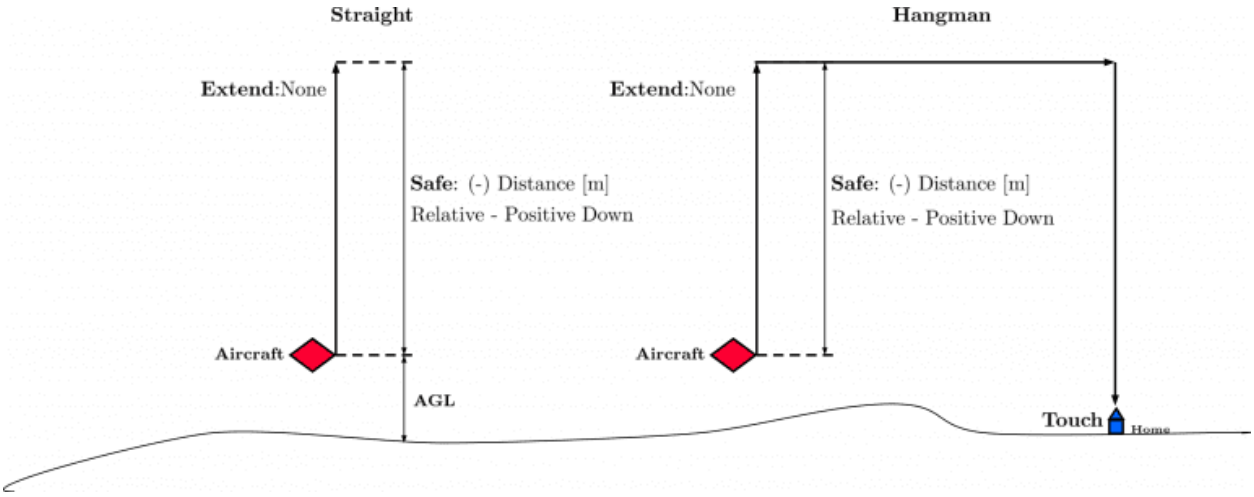


Fig. 324: **Parameters Overview**

2.9.5.10 Yawing current

Yaw guidance is used in multicopters to indicate the behavior of the platform in the yaw axis. This option is normally used during the cruise phase of the multicopters, because the route can be carried out with the aircraft without rotating in the yaw axis, or rotate it to point its longitudinal axis parelel to the path.

Yawing current block produces the desired yaw by **keeping the yaw reading in on focus**. That is, the multicopter will **keep the yaw angle it has when entering in the phase** that contains this guidance. **Desired Yaw = Current Yaw**.

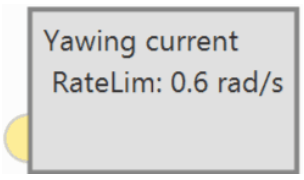


Fig. 325: **Yaw current block**

- **Input**

  (Optional) **Commanded yaw offset with respect to reference**: It is the desired yaw increment with respect to the yaw reference set in the block.

- **Configuration menu**:

Fig. 326: **Yaw current block configuration**

  - **Limit enabled**: When activated, the yaw rate limit will be set to a constant value indicated below.
  - **Limit rate value**: The user can enter this value in different units. The available units are: **rad/s**, **rad/m**, **rad/h**, **rps**, **rpm**, **rph** and **°/s**.

2.9.5.11 Yawing heading

Yaw guidance is used in multicopters to indicate the behavior of the platform in the yaw axis. This option is normally used during the cruise phase of the multicopters, because the route can be carried out with the aircraft without rotating in the yaw axis, or rotate it to point its longitudinal axis parelel to the path.

Yawing heading block produces the desired yaw as **an angle offset from heading**.

Heading represents the direction of the velocity vector and, when it is very small, its estimation is more complex and the direction is constantly changing. Because of this, the **approximation Yaw = Heading** is introduced when the **estimated velocity is close to 0**.

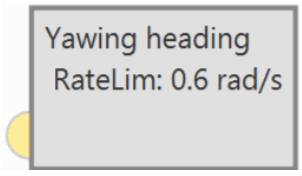


Fig. 327: **Yaw heading block**

- **Input**

  (Optional) **Commanded yaw offset with respect to reference**: It is the desired yaw increment with respect to the yaw reference set in the block.

- **Configuration menu**:

Fig. 328: **Yaw heading block configuration**

- **Limit enabled**: When activated, the yaw rate limit will be set to a constant value indicated below.
- **Limit rate value**: The user can enter this value in different units. The available units are: **rad/s**, **rad/m**, **rad/h**, **rps**, **rpm**, **rph** and **°/s**.

2.9.5.12 Yawing north

Yaw guidance is used in multicopters to indicate the behavior of the platform in the yaw axis. This option is normally used during the cruise phase of the multicopters, because the route can be carried out with the aircraft without rotating in the yaw axis, or rotate it to point its longitudinal axis parelel to the path.

Yawing north block produces the desired yaw as **an angle offset from north**. That is, the yaw of the multicopter will be rotated so that its longitudinal axis always has **north as a reference**.

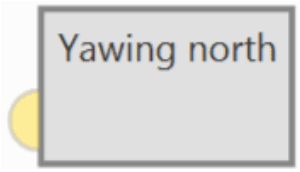


Fig. 329: **Yaw north block**

- **Input**

  (Optional) **Commanded yaw offset with respect to reference**: It is the desired yaw increment with respect to the yaw reference set in the block.

- **Configuration menu**:

Fig. 330: **Yaw north block configuration**

- **Limit enabled**: When activated, the yaw rate limit will be set to a constant value indicated below.
- **Limit rate value**: The user can enter this value in different units. The available units are: **rad/s**, **rad/m**, **rad/h**, **rps**, **rpm**, **rph** and **°/s**.

On the other hand, there are 3 more blocks that help the navigation guidance.

2.9.5.13 Navigation guidance blocks

These blocks, based on the position of a target, calculate the acceleration required to reach that position.

- **PNav**: Proportional navigation block. The algorithm implemented in this block is based on the fact that two vehicles are on a collision course when their direct line-of-sight does not change direction as the range closes.

  So, PNav **dictates that the missile velocity vector should rotate at a rate proportional to the rotation rate of the line of sight** (LOS-rate), and in the same direction.

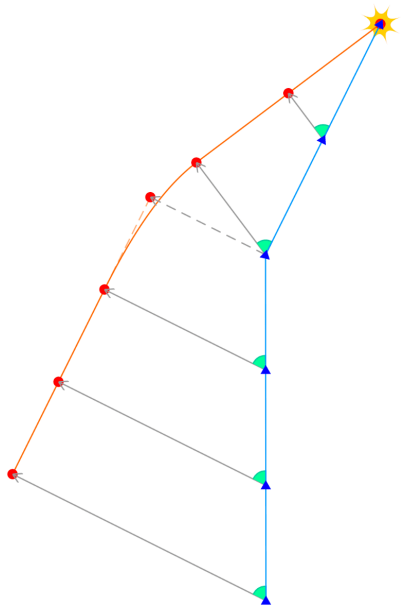


Fig. 331: **PNav algorithm**: A missile (blue) intercepts a target (red) by maintaining constant bearing to it (green)

- **Modified PNav**: Modified proportional navigation guidance (Old Miura).
- **GENEX**: Generalized Vector Explicit Guidance (GENEX) block.

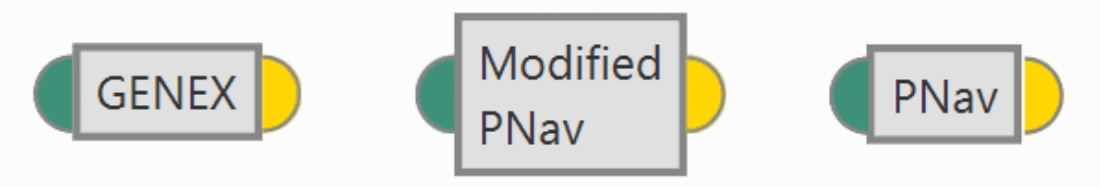


Fig. 332: **Navigation guidance blocks**

They are configured the same, but with slightly different algorithms:

- **Input**
  - **Pin 0**: Target position.
- **Output**
  - **Pin 0**: Desired acceleration in body axis.
- **Configuration menu**:

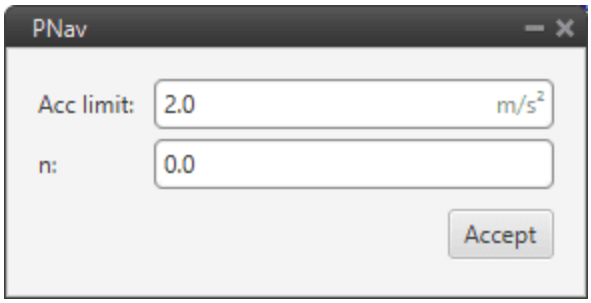


Fig. 333: **PNav block configuration**

The following parameters are configurable:

- **Acc limit**: The user can fix a limit for the acceleration. The units available for this value are: **m/s²**, **ft/s²**, **in/s²** and **g**.
- **n**: It is a proportional parameter.

2.9.6 Library blocks

Library blocks are custom blocks. They are usually a combination of blocks that are used many times in the configuration of block programs, so for ease of configuration, the user can group them into a single block.

There are 2 types of Library blocks: Custom and Default blocks.

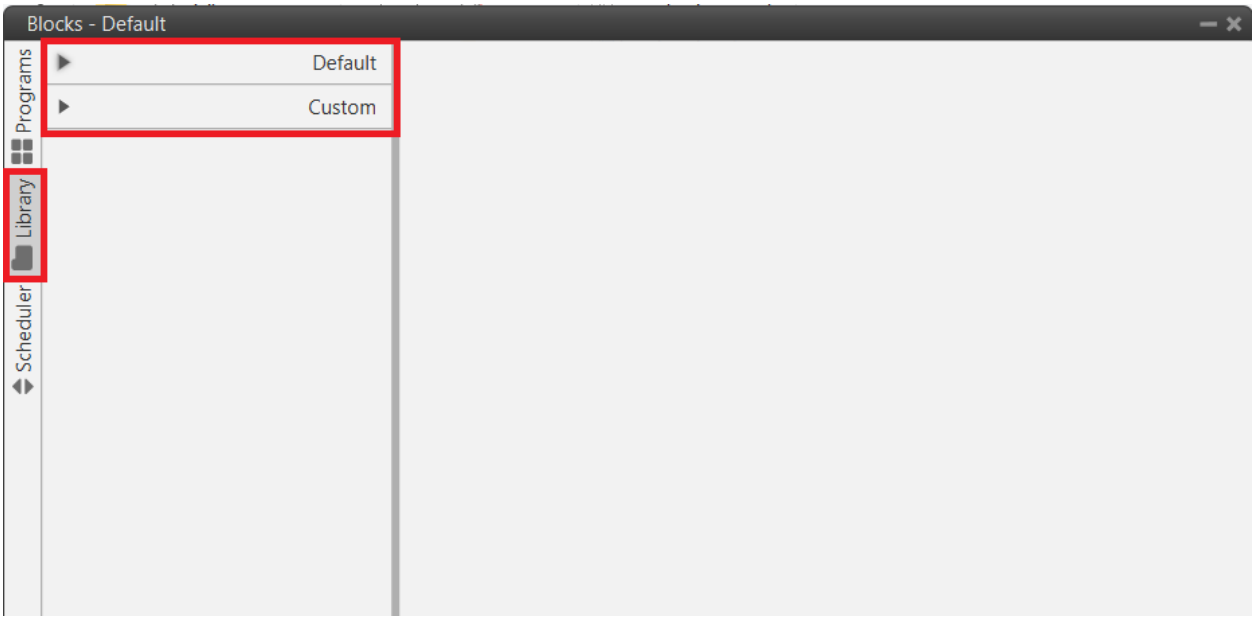


Fig. 334: **Library blocks menu**

- **Custom**

  These are custom blocks created by the user. They can be added by simply clicking on ✚. An example of how to create one of these blocks is shown below:

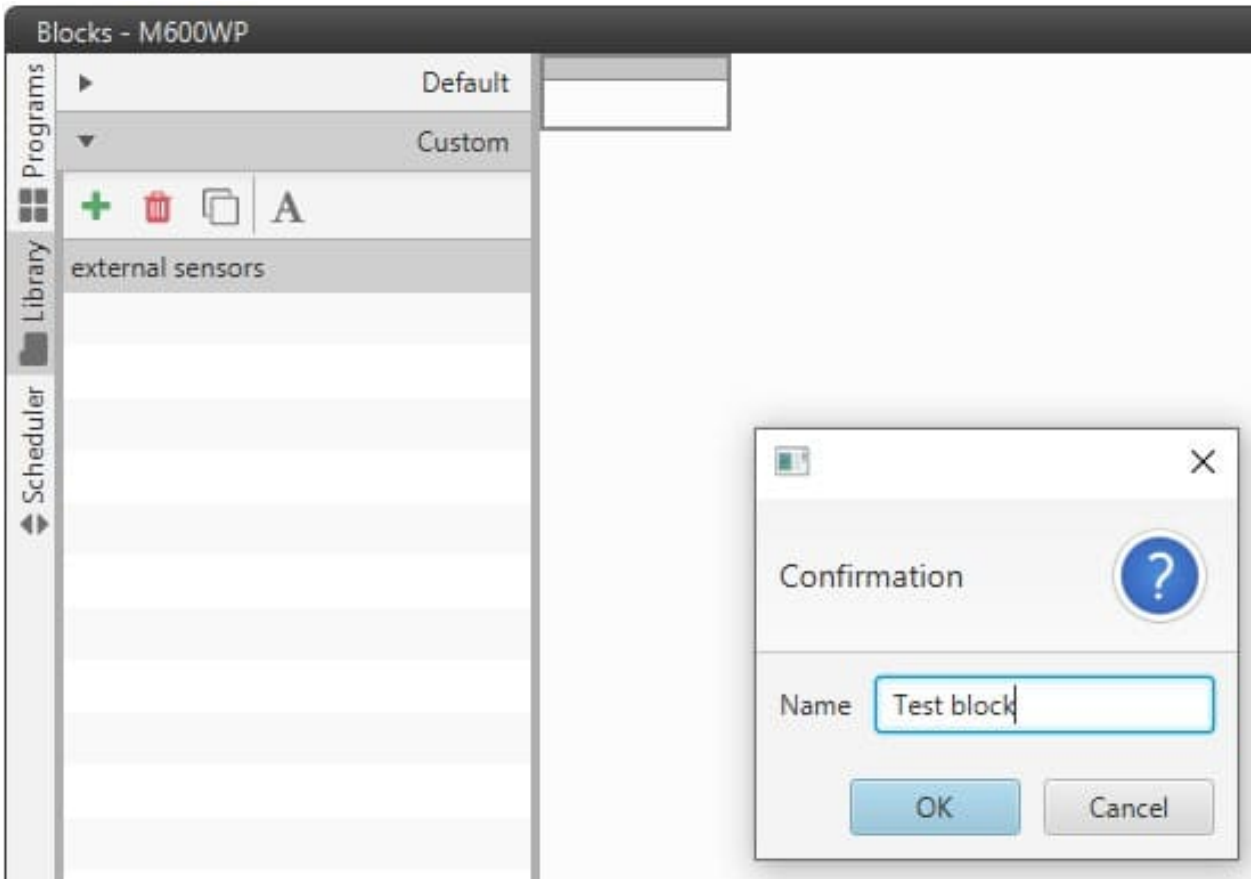


Fig. 335: **Example of custom library block**

Examples of such blocks are those related to the Stick, which have been named “Center Stick” and “Trim Stick”:

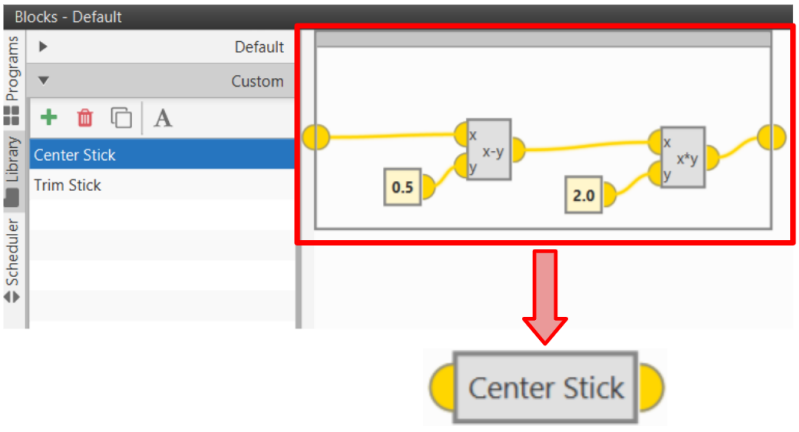


Fig. 336: **Example of custom library block - Center Stick**

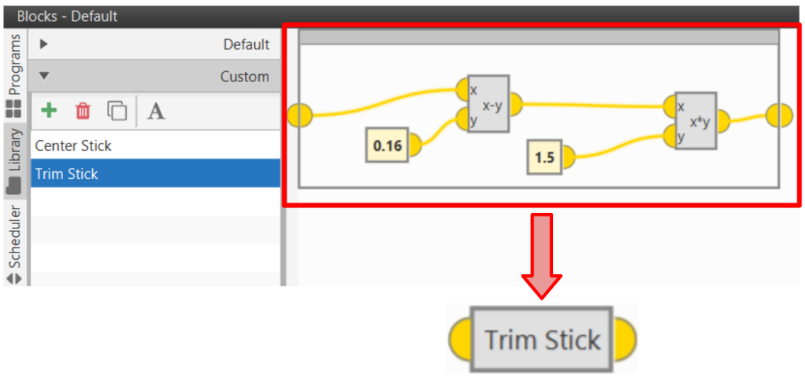


Fig. 337: **Example of custom library block - Trim Stick**

- **Default**

  There is only one 'Default' library block, it is called: **Sagetech ADSB**. This is a block created for the Transponder/ADS-B "**Sagetech MXS**".

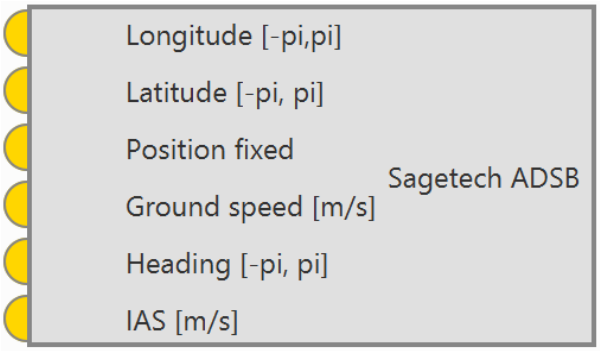


Fig. 338: **Default library block - Sagetech ADSB block**

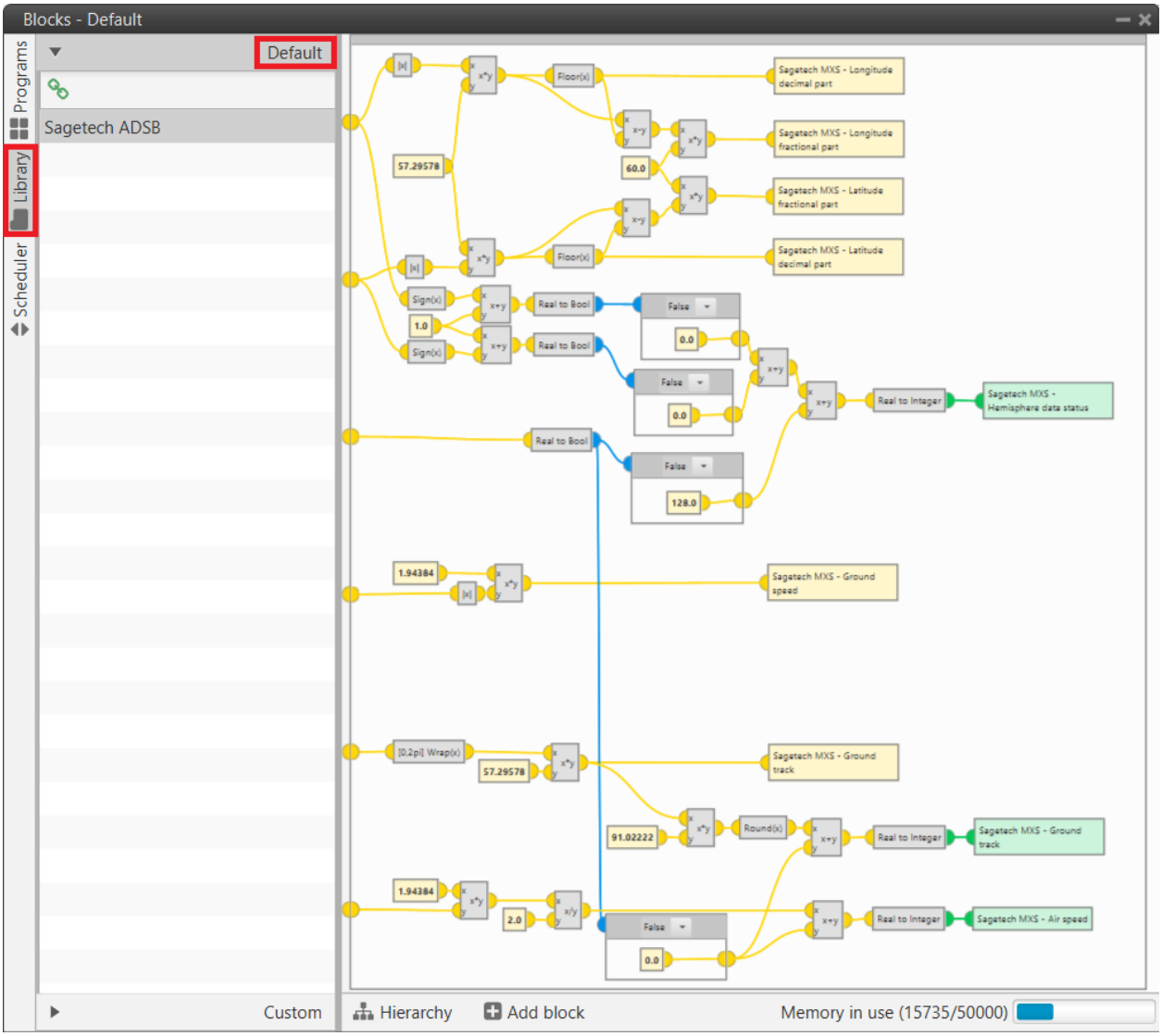


Fig. 339: **Default library block - Sagetech ADSB block configuration**

By clicking on the icon, it will appear as 'Custom' block and will be available in the Library blocks, so that users will be able to use it in their programs.

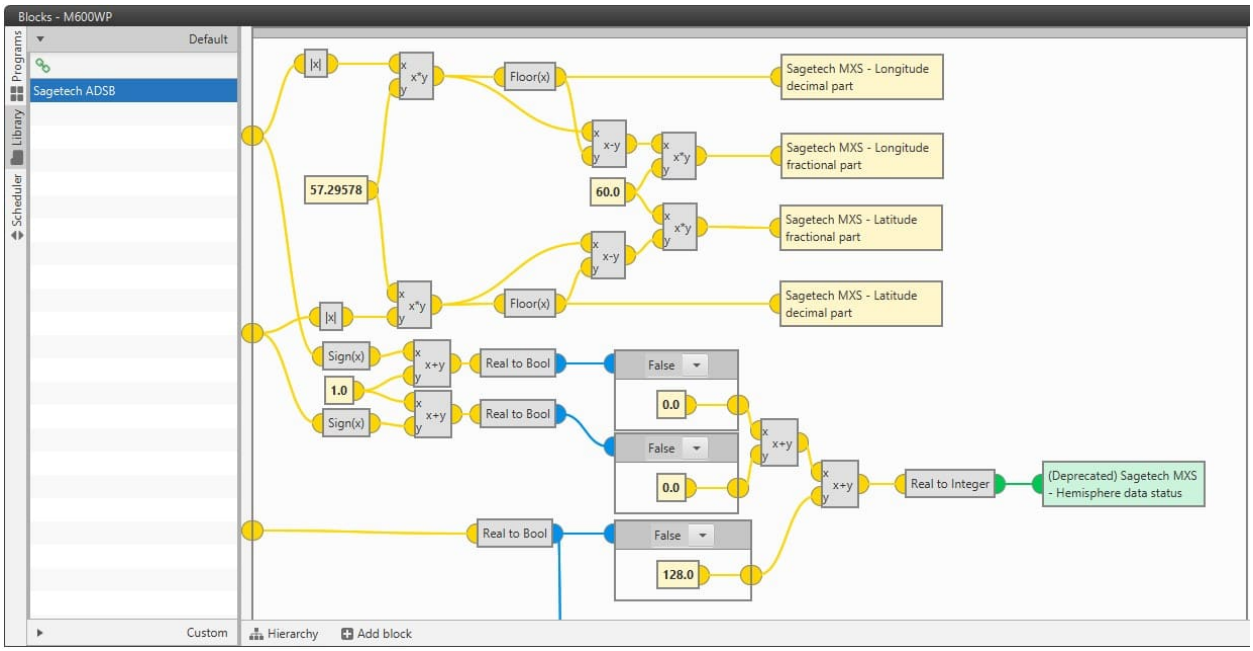


Fig. 340: **Default library block - Sagetech ADSB**

2.9.7 Logic blocks

Logic gates for operating with boolean variables.

2.9.7.1 AND

Returns true if ALL inputs are true, else return false.

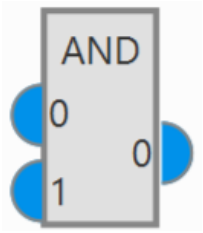


Fig. 341: **AND block**

- **Inputs**
  - **0**: Input bit.
  - **1**: Input bit.
- **Output**
  - **0**: Output bit.
- **Configuration menu**

Fig. 342: **AND block configuration**

Here the user can configure the number of inputs. The units available units for this value are: **bin**, **octal**, **dec** and **hex**.

2.9.7.2 NOT

Logical complement (negation) computation.

Fig. 343: **NOT block**

- **Input**
  - **0**: Input bit.
- **Output**
  - **0**: Output negated bit.

2.9.7.3 OR

Returns true if ANY of inputs are true, else return false.

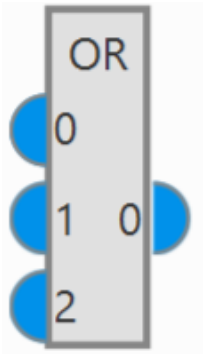


Fig. 344: **OR block**

- **Inputs**
  - **0**: Input bit.
  - **1**: Input bit.

- **2**: Input bit.

- **Output**
  - **0**: Output bit.
- **Configuration menu**

Fig. 345: **OR block configuration**

Here the user can configure the number of inputs. The units available units for this value are: **bin**, **octal**, **dec** and **hex**.

2.9.8 Math blocks

Math blocks allow to perform a wide variety of mathematical operations.

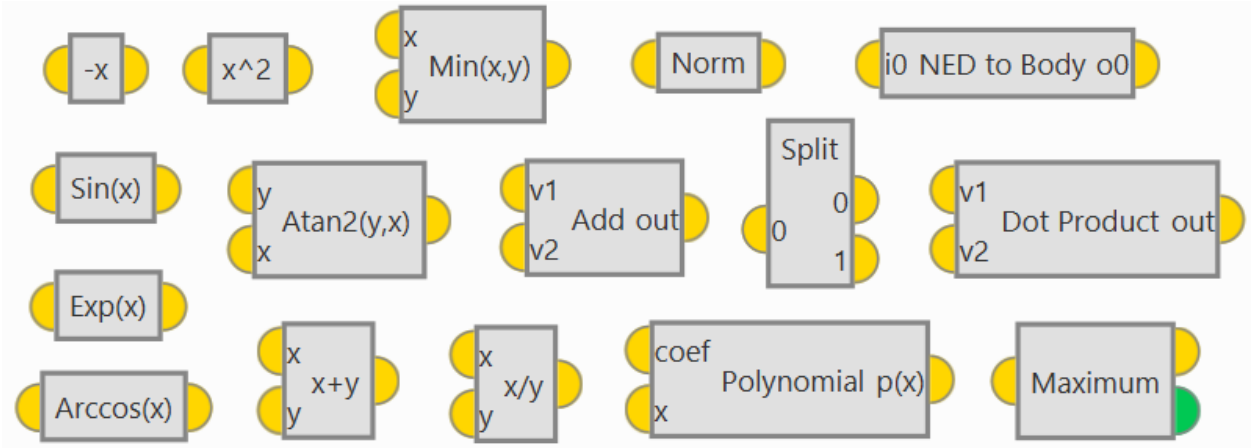


Fig. 346: **Math blocks**

2.9.8.1 f(x)

Math blocks with 1 input and 1 output.

All f(x) math blocks have the same input and output:

- **Input**
  - : Input value, real variable or constant.
- **Output**
  - : Output value.

Warning: All these blocks express the **output value in radians**.

Users will find the following blocks:

- **-x**: Change of sign.
- **1/x**: Inverse of the input (1/x).
- **[-0.5,0.5] Wrap**: Wrapping to the range [-0.5, 0.5].
- **[-pi,pi] Unwrap**: Angle unwrap from [-pi, pi] limits. This block converts an angle signal in the range [-pi, pi] to a continuous signal in the range [-inf, +inf] assuming the smallest angle change between execution steps.
- **[-pi,pi] Wrap**: Angle wrapping to the range [-pi, pi] radians.
- **[0,1] Wrap**: Wrapping to the range [0, 1].
- **[0,2pi] Wrap**: Angle wrapping to the range [0, 2*pi] radians.
- **Arccos(x)**: Arccos function.
- **Arcsin(x)**: Arcsin function.
- **Arctan(x)**: Arctangent function.
- **Ceil(x)**: Closest intger rounding towards plus infinity.
- **Cos(x)**: Cosine function.
- **Exp(x)**: Natural exponent (e number to the power of the input of the block).
- **Floor(x)**: Closest integer rounding towards minus infinity.
- **Log(x)**: Natural logarithm.
- **Round(x)**: Rounding to closest integer.
- **Sign(x)**: Sign of the input. It returns '**1**' if the input is **positive or zero** and '**-1**' if **negative**.
- **Sin(x)**: Sine function.
- **Sqrt(x)**: Square root.
- **Tan(x)**: Tangent function.
- **x^2**: Square of the input.
- **|x|**: Absolute value.

2.9.8.2 f(x,y)

Math blocks with 2 inputs and 1 output.

All f(x,y) math blocks have the same inputs and output:

- **Inputs**
  - : Input value, real variable or constant.
  - : Input value, real variable or constant.
- **Output**
  - : Output value.

Users will find the following blocks:

- **Atan2(y,x)**: Calculates one unique arc tangent value, where the signs of both arguments are used to determine the quadrant of the result.
- **Max(x,y)**: Returns the maximum value of the two inputs.
- **Min(x,y)**: Returns the minimum value of the two inputs.
- **Remainder(x/y)**: Remainder block computes the remainder of the division with the first input as numerator and second input as denominator.
- **x*y**: Mulitplier block.
- **x+y**: Adder block.
- **x-y**: Subtract block computes the subtraction of the first input minuts the second input.
- **x/y**: Divider block computes the division with the first input as numerator and second input as denominator.
- **x^y**: Computes the first input raised to the power of the second input.

2.9.8.3 Polynomial

This block performs a polynomial evaluation, it returns the value of the polynomial defined by the coefficients for the value of x.

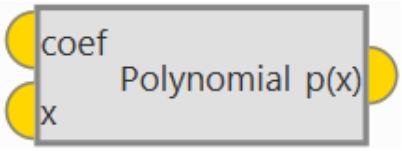


Fig. 347: **Polynomial block**

- **Inputs**
  - **coef**: Array of polynomial coefficients. The order is in increasing order of powers, that is first element is the independent term, second term is the proportional term, third the quadratic term, etc.
  - **x**: Value of evaluation of the polynomial.
- **Output**
  - **p(x)**: Result of the polynomial evaluation.

2.9.8.4 Vectors

These are blocks that perform operations with vectors.

- **Add**: Adds two vectors together.
- **Add Elements**: Adds all the element of the input vector.
- **azeld -> xyz**: Conversion from azimuth, elevation and distance to NED (North, East, Down).

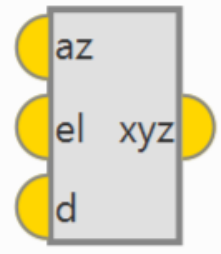


Fig. 348: **azeld -> xyz block**

  - **Inputs**
    - **az**: Azimuth in radians.
    - **el**: Elevation in radians.
    - **d**: Distance in meters.
  - **Output**
    - **xyz**: NED (North, East, Down) vector in meters.

- **Body to NED**: Rotates a vector from the Body frame of reference to North, East, Down.

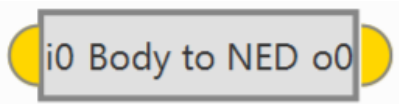


Fig. 349: **Body to NED block**

  - **Input**
    - **i0**: Vector in Body frame.
  - **Output**
    - **o0**: Vector in NED frame.

- **Bundle**: Returns a vector whose components are the inputs of the block. In its configuration, the user can set the number of inputs.

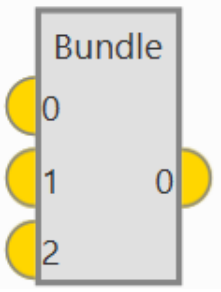


Fig. 350: **Bundle block**

- **Cross product**: Produces the cross product multiplication of the input vectors.
- **Dot Product**: Returns the dot product of the input vectors.

- **Linear Transformation**: Returns the input vector multiplied by the transformation matrix. In order to edit the transformation matrix, double click on the block.

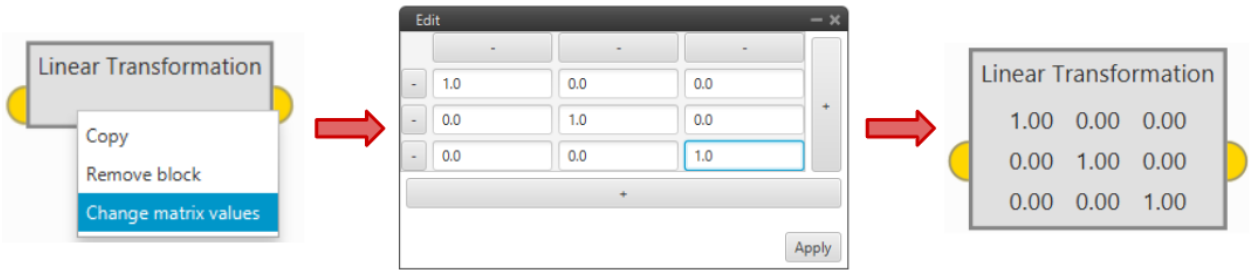


Fig. 351: **Linear Transformation block**

- **m x v**: Multiplies a 3x3 matrix by a vector of size 3.
  - **Inputs**
    - **m**: Matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.

      That is, the input vector must be: [00 10 20 01 11 21 02 12 22]

      Where the matrix will be: $\begin{pmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{pmatrix}$
    - **v**: Column vector with 3 elements.
  - **Output**
    - **Pin 0**: Product of the matrix and the vector, with 3 elements.
- **m1 x m2**: Multiplies two 3 by 3 matrices.
  - **Inputs**
    - **m1**: First matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
    - **m2**: Second matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
  - **Output**
    - **Pin 0**: Matrix product of both matrices, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
- **m1 x m2T**: Multiplies a 3x3 matrix by the transpose of another 3x3 matrix.
  - **Inputs**
    - **m1**: First matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
    - **m2**: Second matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
  - **Output**
    - **Pin 0**: Matrix product of the two inputs, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.

- **m1T x m2**: Multiplies the transpose of a 3x3 matrix by another 3x3 matrix.
  - **Inputs**

    m1: First matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.

    m2: Second matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
  - **Output**

    Pin 0: Product of both matrices, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
- **m1T x m2T**: Multiplies the transpose of two 3x3 matrices.
  - **Inputs**

    m1: First matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.

    m2: Second matrix, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
  - **Output**

    Pin 0: Matrix product of both matrices, as an array of 9 elements, where the element 0 is 00 of the matrix, the 1 is 10 and so on.
- **Mat2quat**: Transforms a 3-by-3 rotation matrix (**input**) into its equivalent quaternion (**output**).

  In its configuration users can select whether to conjugate the quaternion or not.
- **Max**: Returns the value and position (integer output) of the highest component of the input vector.
- **Min**: Returns the value and position (integer output) of the lowest component of the input vector.
- **Multiply Elements**: Returns the product of the components of the input vector.
- **NED to Body**: Rotates a vector from the North, East, Down frame of reference to Body.

Fig. 352: **NED to Body block**

  - **Input**

    i0: Vector in Body frame.
  - **Output**

    o0: Vector in NED frame.
- **Norm**: Computes the norm of the input vector.
- **Quat2mat**: Transforms a quaternion (**input**) into its equivalent 3-by-3 rotation matrix (**output**).

  In its configuration users can select whether to transpose the matrix or not.
- **Scale**: Multiply the input vector (**vIN**) by a scalar value (**k**).

- **Split Bool**: This block takes a boolean vector as input and splits it into as many outputs as the user wishes, each one of them can be a vector or a single value from the input.

  Each output produces a vector with the size (number of elements) indicated in **Size** from the element of the **input vector** indicated in the **Index** parameter.

  In the following example:

  - The first output produces a vector of size 4 with the first four elements (from 0 to 3).
  - The second output only generates a vector of size 1 with the fifth element (4).
  - The third output produces a vector of size 3 with the third to fifth elements (from 2 to 4).

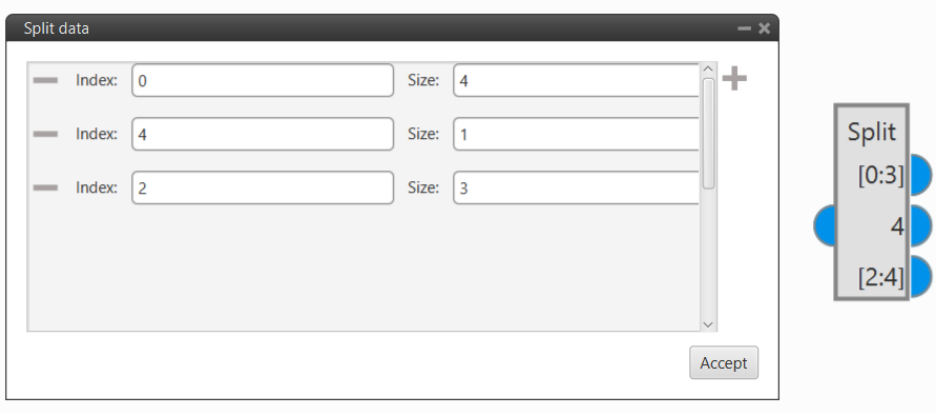


Fig. 353: **Split Bool block example**

- **Split Real**: This block works in the same way as **Split Bool** block does. Nonetheless, **Split Real** operates with real instead of boolean variables.
- **Substract**: Subtracts from the first input vector (**v1**) the other input vector (**v2**).
- **Vector rotation**: Rotates a given vector of 3 elements (**vector to rotate**) by the provided rotation angles (**rotation angles**).
- **xyz -> azeld**: Conversion from NED (North, East, Down) to azimuth, elevation and distance.

Fig. 354: **xyz -> azeld block**

  - **Input**

    xyz: NED (North, East, Down) vector in meters.

  - **Outputs**

    az: Azimuth in radians.

- **el**: Elevation in radians.
- **d**: Distance in meters.

2.9.9 Mode/AP Selection blocks

Mode/AP Selection blocks allow to interact with flight modes and redundancy (Autopilot 4x).

2.9.9.1 AP Selection

AP selection block is a **Veronte Autopilot 4x** consensus. This block behaves differently depending on the selected AP:

- If **this AP is the selected one** then the **input is copied into the output** and the **input is sent** to the other APs **via CAN** (needed to have correct CAN ports configuration).
- If **this AP is not the selected one** then the **outputs** are the **received values via CAN** from the selected AP.

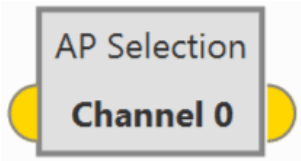


Fig. 355: **AP Selection block**

- **Input**
  - **Pin 0**: Values computed by this AP.
- **Output**
  - **Pin 0**: Values computed by the selected AP.
- **Configuration menu**: The user must select the channel through which information is shared between the APs.

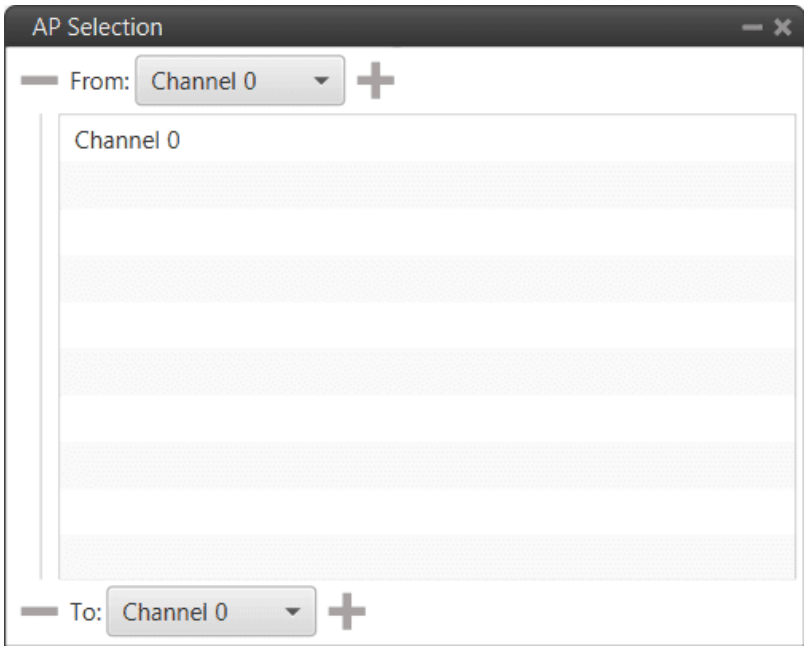


Fig. 356: **AP Selection block configuration**

This block **normally is used for the control outputs**. The APs share the control outputs with each other so that they all use the control output of the selected AP. Therefore, the **input** is the **control output of the AP in the configuration** and the **output** is the **control output of the selected AP**. This can actually be used with any type of variable that you want to share and use the selected AP's variable.

This block is intended to be used as **input** for the inner loop PID (at **control rate**), so that the Integral term of the 3 Autopilots 1x of the Autopilot 4x remains the same. This way, if the selected autopilot from the 4x is switched to one of the other 2 available autopilots, a smooth response without significant 'jumps' in control will still be obtained. An example of this use is shown below:

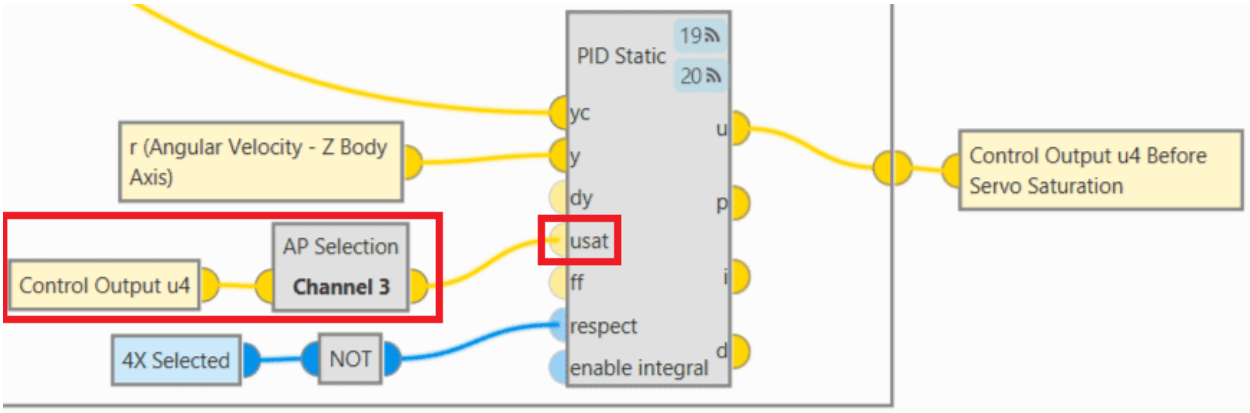


Fig. 357: **AP Selection block - Example of use**

2.9.9.2 Arcade

Arcade Pure block switches between two input signals according to the current mode of the configured channel. Refer to the *modes configuration table* to check the configuration (go to *Modes panel* in the **Control menu**).

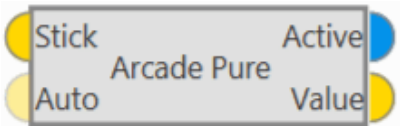


Fig. 358: **Arcade block**

- **Inputs**
  - **Stick**: **Arcade value** to be applied when the configured channel is **in arcade mode**.
  - (Optional) **Auto**: **Auto value** to be applied when the configured channel is **not in arcade mode**.

  The default value if not connected is zero.
- **Outputs**
  - **Active**: **TRUE** if the configured channel is **in arcade mode**, FALSE otherwise.
  - **Value**: Value to apply. In **arcade mode**, this value is equal to the first input (**arcade value**) and in **any other mode** is equal to the second input (**auto value**).
- **Configuration menu**:

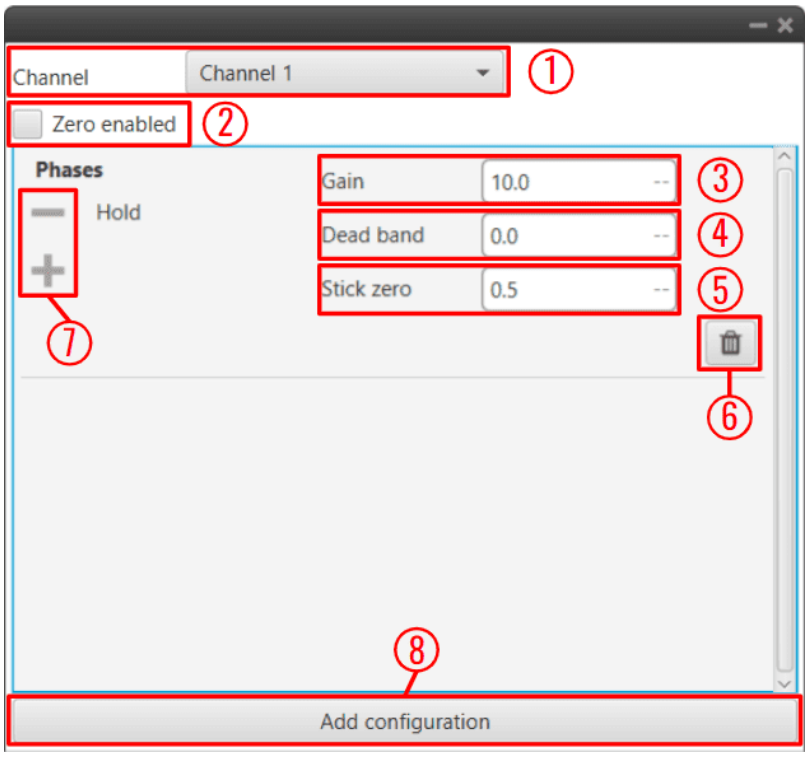


Fig. 359: **Arcade block configuration**

The following parameters must be configured:

1. **Channel**: Channel controlled by this block. Be careful, each channel is related to a control output previously defined in the *Modes panel* of the **Control menu**.
2. **Zero enabled**: It can be enabled by the user.
   - **Disabled**: If the **stick is in position "0"**, even if it is in arcade mode, the autopilot processes it as if it were **in auto mode**, and consequently, the value of the output **active** will be **false** and the value of the output **value** will be that of the **auto input**.
   - **Enabled**: If the **stick is in position "0"** and in arcade mode, the autopilot still processes it as being **in arcade mode**, so the output **active** value will be **true** and the output **value** will be that of the **stick input**.
3. **Gain**: The output value is the result of multiply the **stick input** by this gain.
4. **Dead band**: Creates a zone where the movement of the stick is not sent to the system.
5. **Stick zero**: Output value when the value of the stick input is 0.

Note: The following diagram details the differences between the parameters **Dead band** and **Stick zero**:

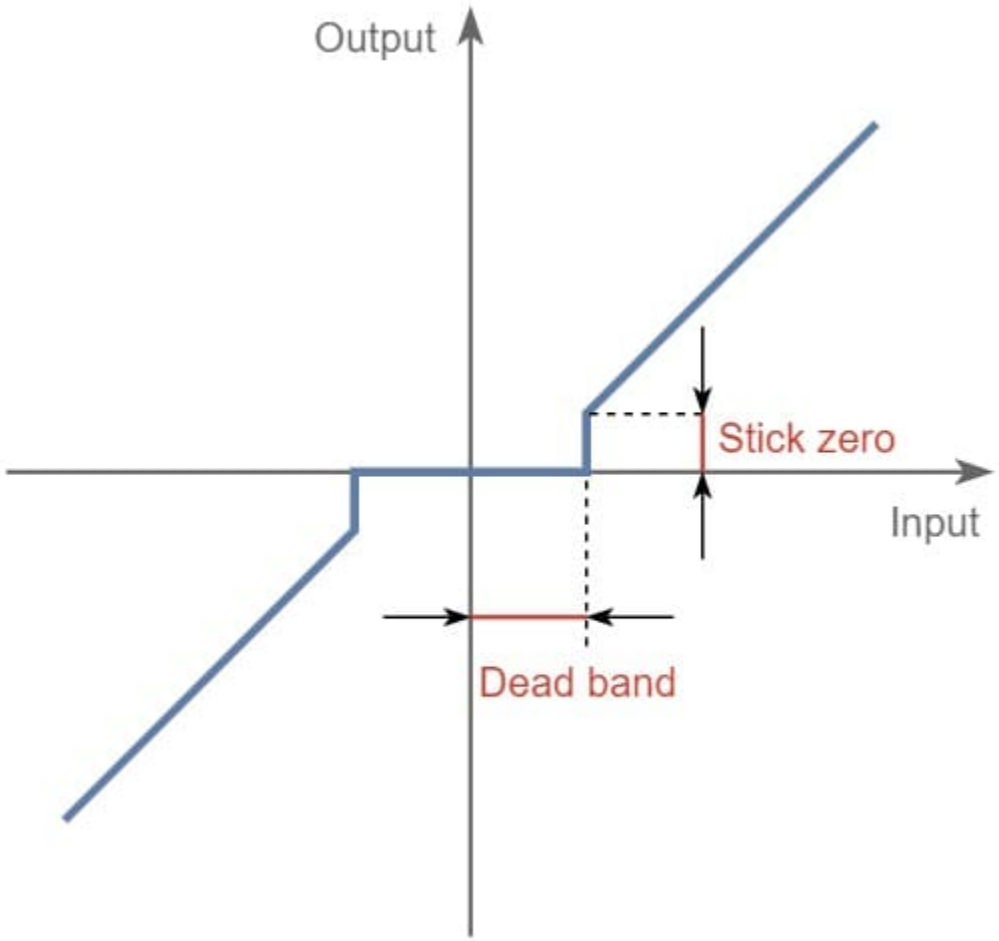


Fig. 360: **Arcade block configuration - Dead band VS Stick zero**

---

6. Delete a group of phases.
7. Delete/Add a phase to a group.
8. **Add configuration**: Add a new group of phases affected by this block.

2.9.9.3 Arcade Bounce

Arcade Bounce block switches between two input signals according to the current mode of the configured channel preventing bounces in transitions from arcade to auto. Refer to the *mode configuration table* to check the configuration (go to *Modes panel* in the **Control menu**).

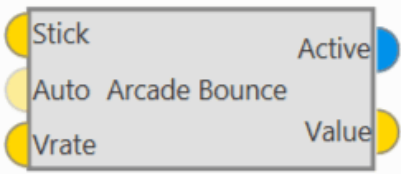


Fig. 361: **Arcade Bounce block**

In the pictures below there is an example controlling the yaw.

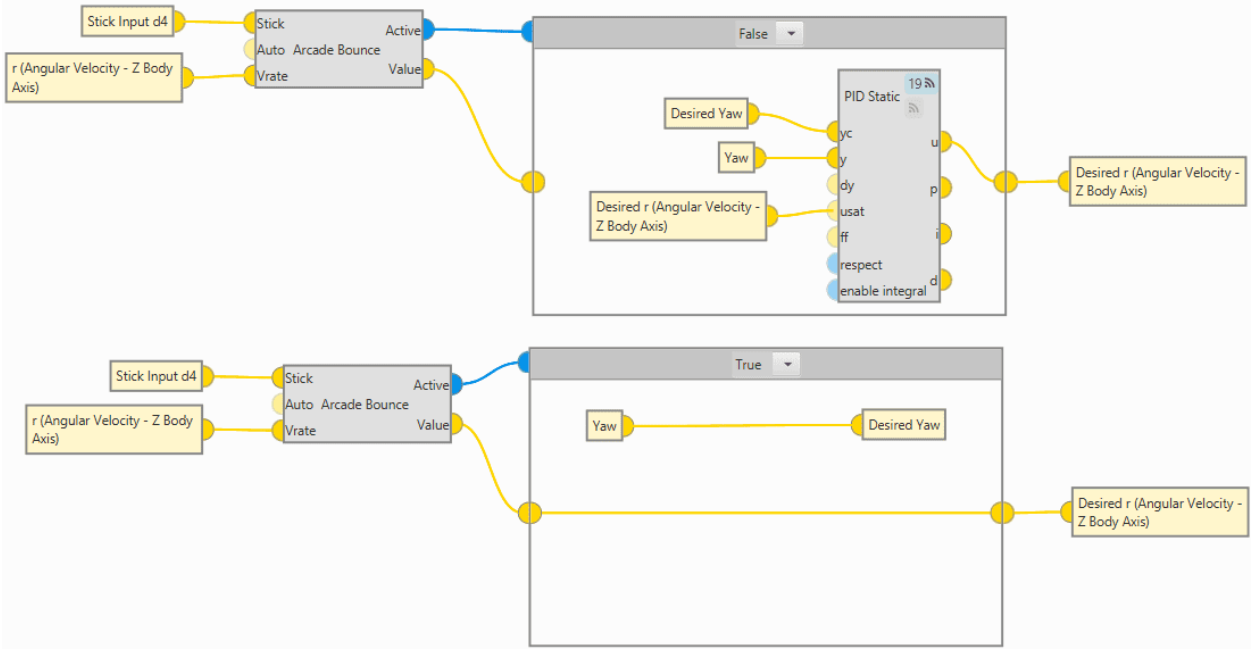


Fig. 362: **Arcade Bounce block example**

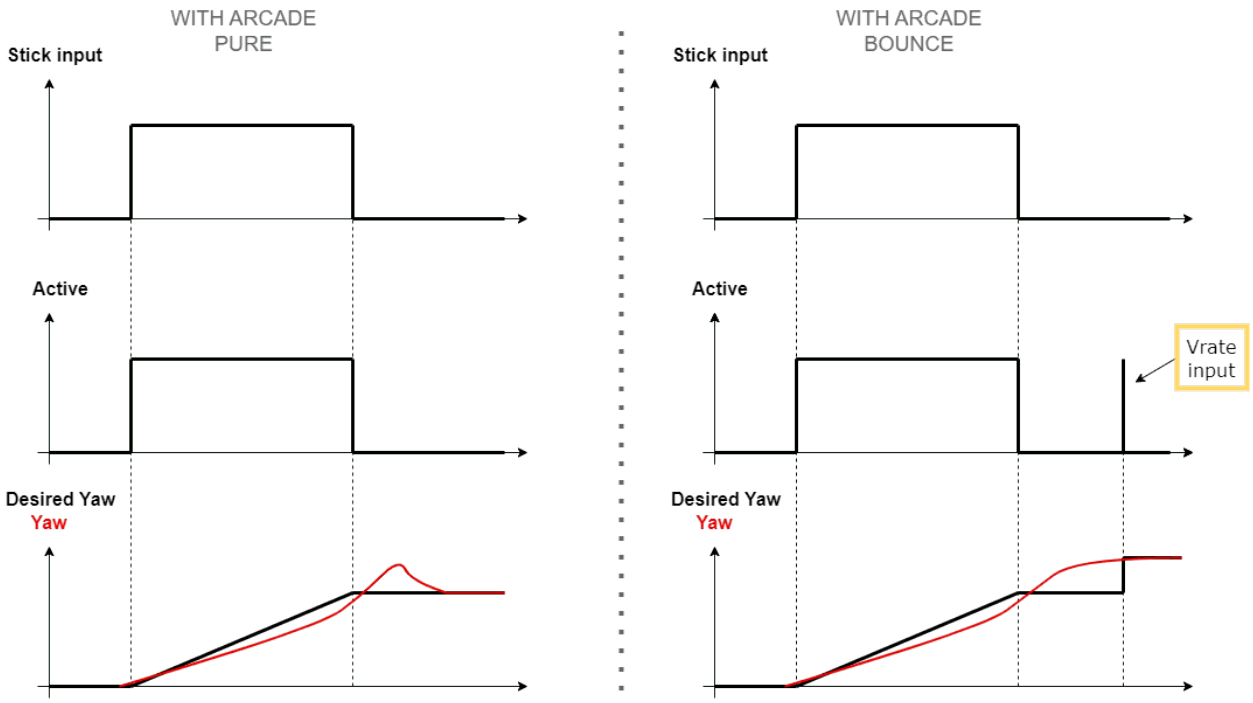


Fig. 363: **Arcade Pure VS Arcade Bounce**

This example is explained below:

When the stick is in its zero position, the command sent is 0, so the status of the Active BIT is FALSE (low level), and the desired yaw is the last yaw saved when the status was TRUE (high level).

However, the platform can still have a **yaw rate** (*r (Angular Velocity)* variable in the block example) and in an **arcade pure block** it could experiment this **bounce**.

Therefore, with the **Arcade Bounce block**, when the Yaw rate (**Vrate input**) approaches 0 (**changes its sign**), this BIT reverts to TRUE (for a moment), and even though a new yaw rate is not being commanded with the stick, the desired yaw is being updated with the current yaw.

As it is very similar to the Arcade Pure block, the inputs and outputs are the same, except that an additional input is added to this block:

- **Input**
  - **Vrate**: Controlled variable used to prevent bounces.
- **Configuration menu**:

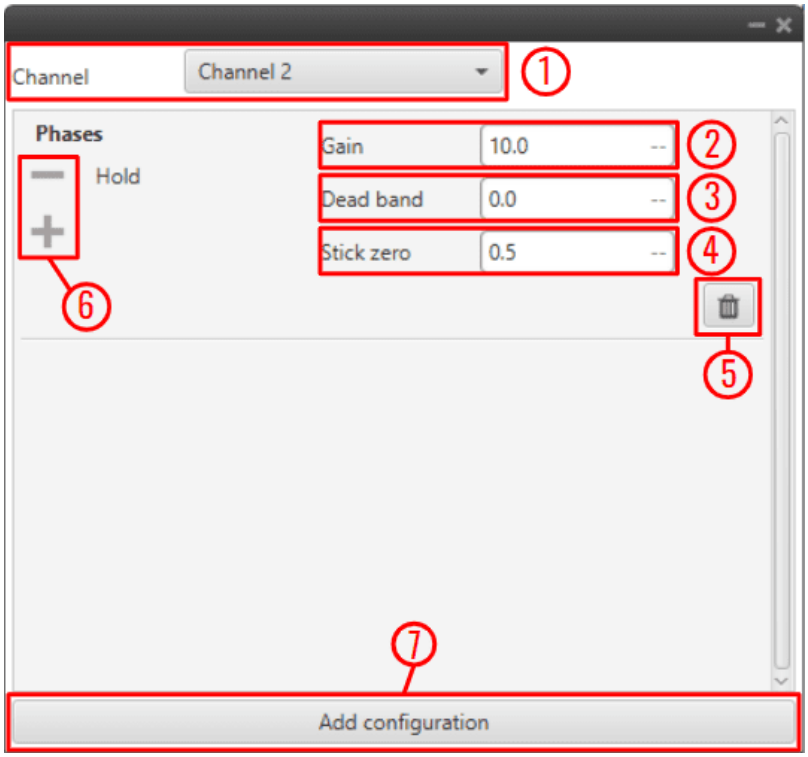


Fig. 364: **Arcade Bounce block configuration**

The following parameters must be configured:

1. **Channel**: Channel controlled by this block. Be careful, each channel is related to a control output previously defined in the *Modes panel* of the **Control menu**.
2. **Gain**: The output value is the result of multiply the **stick input** by this gain.
3. **Dead band**: Creates a zone where the movement of the stick is not sent to the system.
4. **Stick zero**: Output value when the value of the stick input is 0.
5. Delete a group of phases.
6. Delete/Add a phase to a group.
7. **Add configuration**: Add a new group of phases affected by this block.

2.9.9.4 Arcade Extend

Arcade Extend block switches between two input signals according to the current mode of the configured channel smoothing the transition by **extending the arcade mode until the input 'Val' goes below the configured margin**. (Similar to Arcade Bounce block). Refer to the *mode configuration table* to check the configuration (go to *Modes panel* in the **Control menu**).

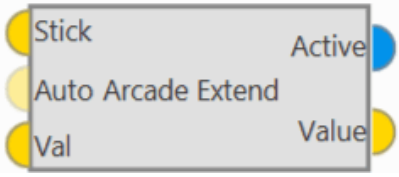


Fig. 365: **Arcade Extend block**

As it is very similar to the Arcade Pure block, the inputs and outputs are the same, except that an additional input is added to this block:

- **Input**

  Val: Controlled variable used to extend the arcade to auto transitions.

- **Configuration menu**:

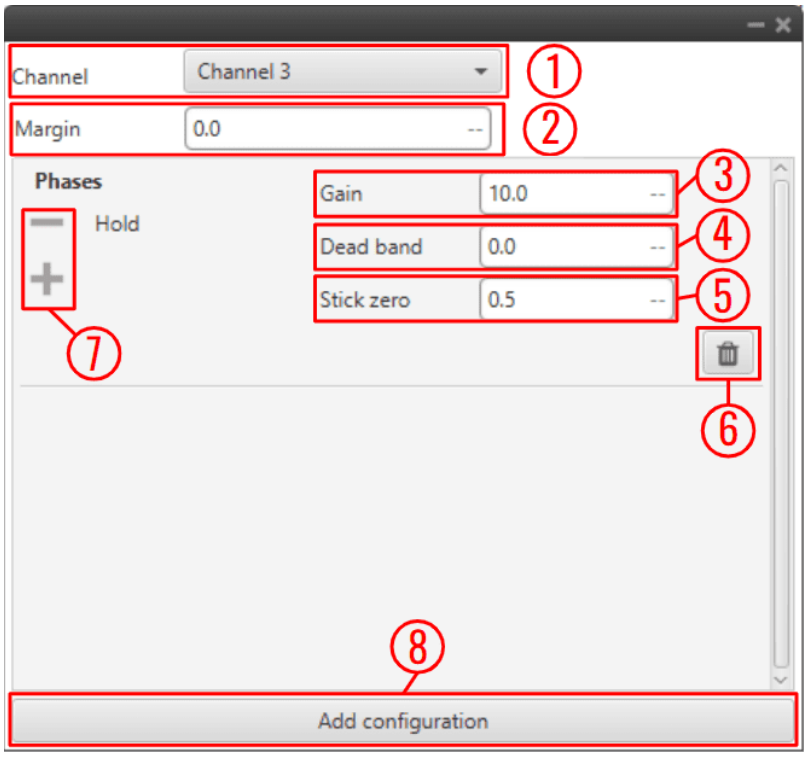


Fig. 366: **Arcade Extend block configuration**

The following parameters must be configured:

1. **Channel**: Channel controlled by this block. Be careful, each channel is related to a control output previously defined in the *Modes panel* of the **Control menu**.
2. **Margin**: If the 'Val input' is higher than the margin set here, the autopilot will remain in **arcade mode**.
3. **Gain**: The output value is the result of multiply the **stick input** by this gain.
4. **Dead band**: Creates a zone where the movement of the stick is not sent to the system.
5. **Stick zero**: Output value when the value of the stick input is 0.

6. Delete a group of phases.
7. Delete/Add a phase to a group.
8. **Add configuration**: Add a new group of phases affected by this block.

2.9.9.5 Manual

Manual block switches between two input signals according to the current mode of the configured channel. Refer to the *mode configuration table* to check the configuration (go to *Modes panel* in the **Control menu**).

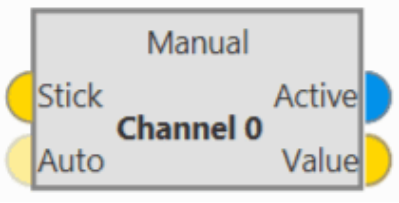


Fig. 367: **Manual block**

- **Inputs**

  Stick: **Manual value** to be applied when the configured channel is **in manual mode**.

  (Optional) **Auto**: **Manual value** to be applied when the configured channel is **not in manual mode**.

  The default value if not connected is zero.

- **Outputs**

  Active: TRUE if the configured channel is in manual mode, FALSE otherwise.

  Value: Value to apply, which **in manual mode** is equal to the first input (**manual value**) and in **any other mode** is equal to the second input (**auto value**).

- **Configuration menu**: The user must select the channel to be controlled.

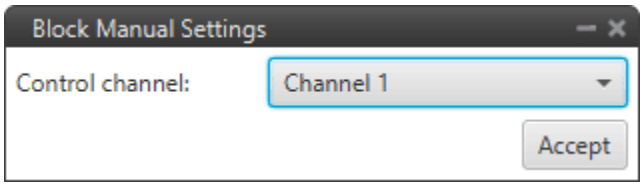


Fig. 368: **Manual block configuration**

2.9.9.6 Mix

Mix block adds a 'Stick' signal to an 'Auto' signal if the **current mode** for the configured channel is **MIX**, otherwise the output is directly the 'Auto' signal. In other words, it allows a variable offset to be applied to the input using one of the stick channels.

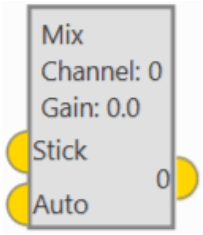


Fig. 369: **Mix block**

- **Inputs**
  - **Stick**: Value to be added to 'Auto' when the configured channel is in MIX mode.
  - **Auto**: Auto value.
- **Output**
  - **0**: If the **configured mode** of the configured channel is **MIX** the **output** is the **addition of 'Stick' and 'Auto'**, otherwise the output is directly 'Auto'.
- **Configuration menu**:

Fig. 370: **Mix block configuration**

The following parameters can be configured:

- **Channel**: The user must select the channel to be controlled.
- **Gain**: This gain will multiply the 'Stick' input (before the addition with the 'Auto' input) if the current mode is MIX.

2.9.10 Navigation blocks

2.9.10.1 EKF Adapters

These blocks allow the connection between the autopilot sensors (internal or external) and the calculation of the navigation algorithm. That is, they "convert" the sensor data into EKF data in order to implement them in the navigation block (Extended Kalman Filter algorithm).

For this reason, EKF Adapters blocks normally work with the *Sensor blocks* as inputs.

2.9.10.1.1 Altitude

Altitude block adapts an exteral altimeter sensor, like LIDAR, Sonar, etc., to the EKF input.

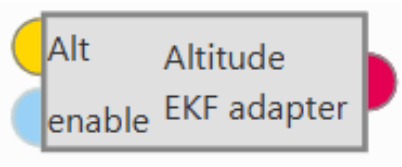


Fig. 371: **Altitude block**

- **Inputs**

  Alt: Altitude measurement as a 3-dimensional real array with the following components:

  - 0: Update flag

  - 1: Altitude measurement

  - 2: Variance

  This input corresponds to the *Altitude Sensor block*.

  (Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

  Not connected means enabled.

- **Output**

  Pin 0: EKF input data (H, R, y).

  Note:

  - **H**: Observation matrix.
  - **R**: Measurement covariance matrix.
  - **y**: Measurement.

- **Configuration menu**:

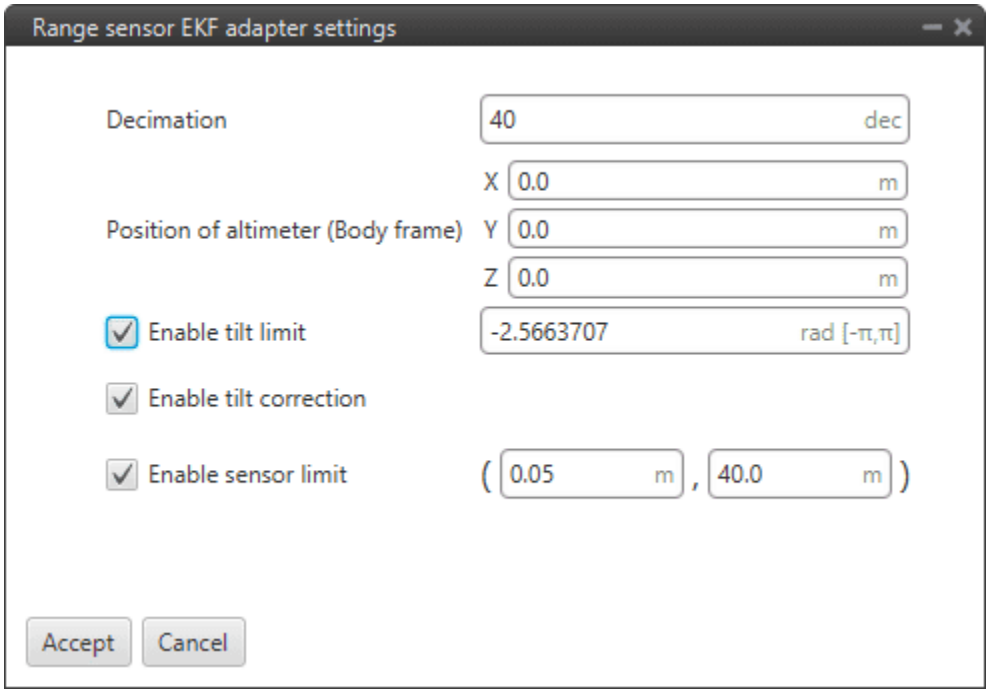


Fig. 372: **Altitude block configuration**

The following acceletometer parameters must be configured:

- **Decimation**: Parameter that defines the group of measurements for which 1 value will be stored. For example, if the decimation is 10, every 10 measurements will be counted as 1. This procedure is used **to reduce the number of samples**. **Recommended value 40** (in decimal units).

  If users have any questions about decimation, please refer to the *What does decimation mean? - FAQ* section of this manual.

- **Position of altimeter (Body frame)**: Parameter to indicate the distance from the altimeter to the center of gravity of the platform. This is used to take into account the weight of the altimeter in the aircraft control.

- **Enable tilt limit**: The altimeter is normally installed in a fixed position having a constant direction with respect to the platform. Taking a LIDAR as an example, it is used to measure altitude so it has to point towards the ground, in a direction parallel to the Z body axis. When the vehicle is not level on its X or Y axis (has a pitch or roll angle different from zero), the LIDAR will not point in a direction perpendicular to the ground, and the measurement taken will not be the real altitude of the aircraft. This option is a safe condition to discard the measure of an altimeter when its tilt angle exceeds a certain value defined here.

- **Enable tilt correction**: Allows the correction of the altimeter sensor measurement, normally AGL measurent, with internal pitch and roll measurements.

$$correction = (AGL\ measurement) \cdot \cos(pitch) \cdot \cos(roll)$$

- **Enable sensor limits**: It is the range in which the sensor measurement is taken to be processed by **1x PDI Builder**. Any external value will be discarded by the system.

The following figure shows a diagram with the values of maximum and minimum sensor limits altitude, and the maximum tilt angle.

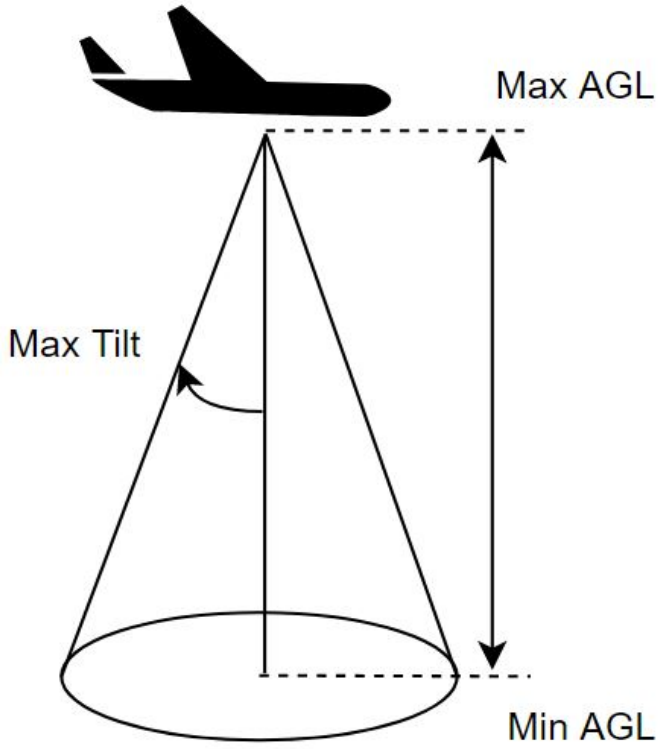


Fig. 373: **Altitude block - Limits**

2.9.10.1.2 GNSS compass

GNSS compass block takes two relative position measurements and converts them to **misalignment vectors** to use in the EKF **for attitude correction**.

Fig. 374: **GNSS compass block**

- **Inputs**: These inputs correspond to the *GNSS Sensor block*.

  Drn1: **Relative position measurement** 0 as a **10-dimensional real array** with the following components:

  - 0: Update flag
  - 1: Rover flag: 1 is rover, 0 is base
  - 2: Time stamp
  - 3: North relative distance
  - 4: East relative distance
  - 5: Down relative distance

- 6: Relative distance variance
- 7: X body antenna position
- 8: Y body antenna position
- 9: Z body antenna position

Drn2: Relative position measurement 1 as a 10-dimensional real array with the same components as described above.

- **Outputs**

NED: **Baseline vector in NED frame** as a **5-dimensional real array** with the following components:

- 0: Update flag (always 1)
- 1: North component
- 2: East component
- 3: Down component
- 4: Variance

Body: **Baseline vector in body frame** as a **5-dimensional real array** with the following components:

- 0: Update flag (always 1)
- 1: X body component
- 2: Y body component
- 3: Z body component
- 4: Variance.

An example of how to implement this block is presented below:

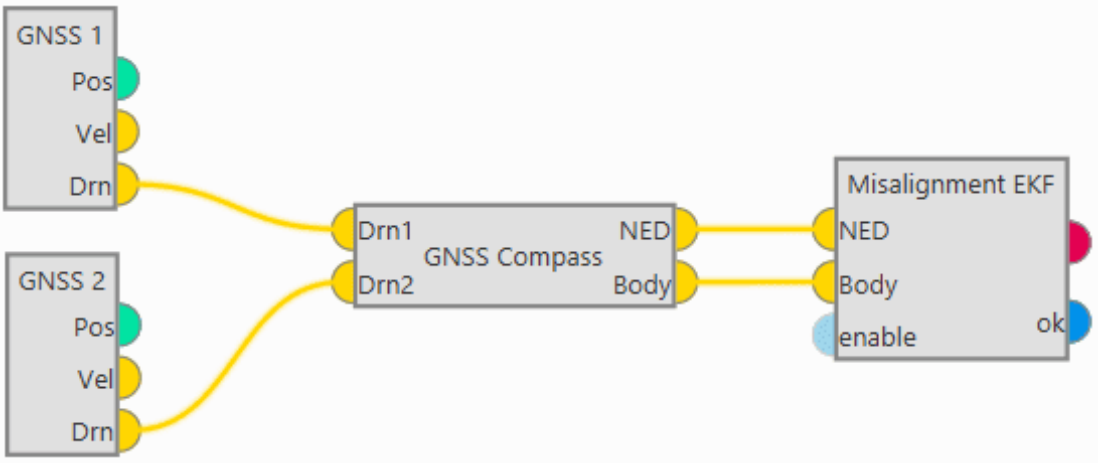


Fig. 375: **GNSS compass block example**

2.9.10.1.3 Misalignment

Misalignment block transforms from **two vectors expressed in NED and body frames** to **EKF misalignment data for attitude correction**.

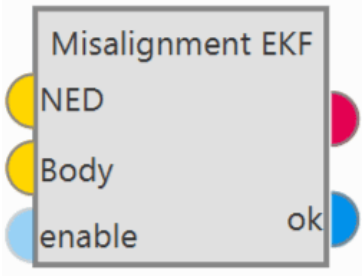


Fig. 376: **Misalignment block**

- **Inputs**

  NED: **Vector measured in NED frame as a 5-dimensional real array** with the following components:

  - 0: Update flag (always 1)
  - 1: North component
  - 2: East component
  - 3: Down component
  - 4: Variance

  Body: **Vector measured in body frame as a 5-dimensional real** array with the following components:

  - 0: Update flag (always 1)
  - 1: X body component
  - 2: Y body component
  - 3: Z body component
  - 4: Variance.

  (Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

  Not connected means enabled.

- **Outputs**

  Pin 0: EKF input data (H, R, y).

  Note:

  – **H**: Observation matrix.

  – **R**: Measurement covariance matrix.

  – **y**: Measurement.

  ---

  ok: Measurement check bit, returns **true** if the **measurements pass the module checks**, false otherwise.

- **Configuration menu**:

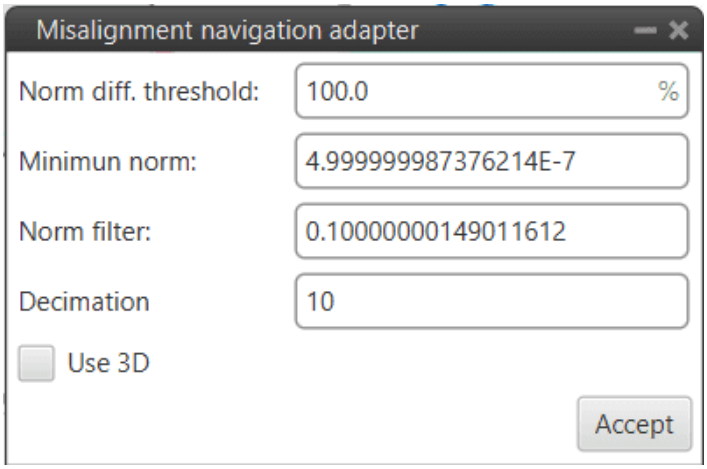


Fig. 377: **Misalignment block configuration**

The following parameters are configurable:

– **Norm diff. threshold**: In order to use the measures that enter the block, the moduli of both measurements must be similar according to the equation:

$$\left| \frac{nmn - nmbn}{(nmn + nmbn) \cdot ndthr} \right| < 1$$

Where,

* nmn: This is the norm of the first input vector.
* $nmbn$: This is the norm of the second input vector.
* $ndthr$: **Norm diff. threshold** as a percentage of one, users should look at the absolute value.

– **Minimum norm**: In addition, it must also be fulfilled that the moduli of the measures are **greater than the Minimum norm defined here**.

– **Norm filter**: This parameter is to filter the measurement of $\left| \frac{nmn-nmbn}{(nmn+nmbn)\cdot ndthr} \right|$ to use it in the Madgwich algorithm.

– **Decimation**: Parameter that defines the group of measurements for which 1 value will be stored. For example, if the decimation is 10, every 10 measurements will be counted as 1. This procedure is used **to reduce the number of samples**. **Recommended value 10** (in decimal units).

If users have any questions about decimation, please refer to the *What does decimation mean? - FAQ* section of this manual.

– **Use 3D**: If enabled, the attitude correction will be in 3D, as by default, only the attitude correction in the horizontal plane is activated.

2.9.10.1.4 Position

Position block adapts **absolute position** data to **EKF data for position update**.

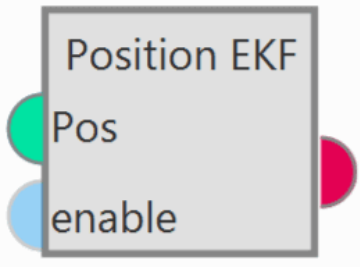


Fig. 378: **Position block**

- **Inputs**

  Pos: Absolute position measurement. This input corresponds to the *GNSS Sensor block* or *Relative position sensor block*.

  (Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

  Not connected means enabled.

- **Output**

  Pin 0: EKF input data (H, R, y).

  Note:

  – **H**: Observation matrix.

  – **R**: Measurement covariance matrix.

  – **y**: Measurement.

- **Configuration menu**:

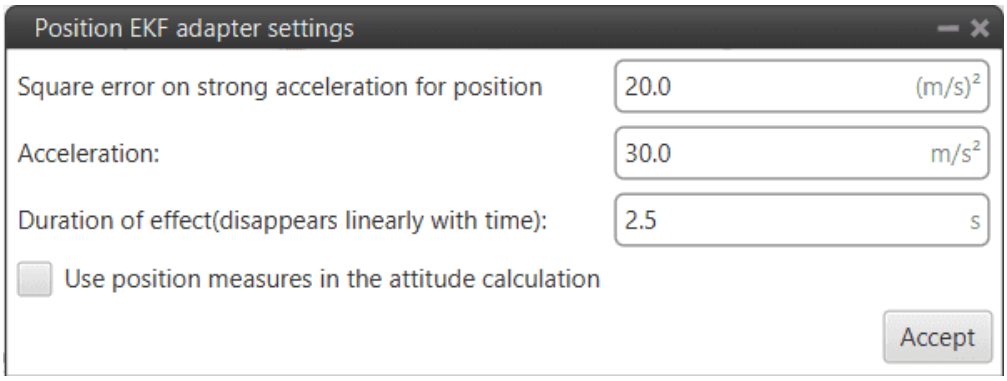


Fig. 379: **Position block configuration**

Users must configure the following parameters:

- **Square error on strong acceleration for position**: Under strong acceleration the variance for the position solution is changed to the specified value.
- **Acceleration**: Threshold definition. When this threshold is exceeded, strong acceleration variances are considered.
- **Duration of effect (disappears linearly with time)**: Time needed to restore the default variances of the GNSS solution.
- **Use position measures in the attitude calculation**: When enabled, the position data from the GNSS solution is considered for the attitude estimation.

2.9.10.1.5 Static Pressure

Static Pressure block converts **static pressure** measurement into **EKF data for altitude update**.

Fig. 380: **Static Pressure block**

- **Inputs**

  mea: Static pressure measurement as a **3-dimensional real array** with the following components:

  - 0: Update flag

  - 1: Pressure

  - 2: Variance

  This input corresponds to the *Static Pressure sensor block*.

  (Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

  Not connected means enabled.

  (Optional) **gec-y**: Enable **ground effect measurement correction** (true) or disable (false).

  Not connected means disabled.

  (Optional) **gec-r**: Enable **ground effect variance increment** (true) or disable (false).

  Not connected means disabled.

- **Outputs**

  Pin 0: EKF input data (H, R, y).

Note:

- **H**: Observation matrix.
- **R**: Measurement covariance matrix.
- **y**: Measurement.

- **hQNH**: **MSL** right from **actual QNH and pressure** measurement.
- **hISA**: **MSL** for **ISA and pressure** measurement.

- **Configuration menu**:

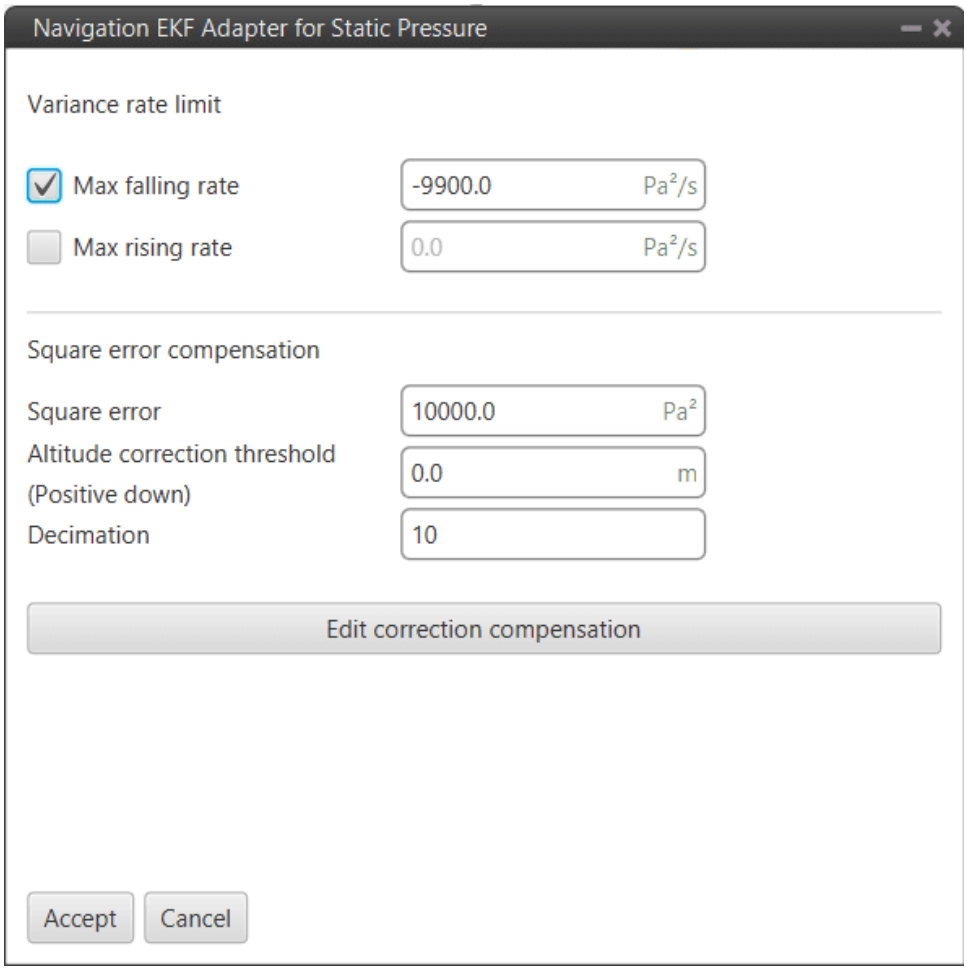


Fig. 381: **Static Pressure block configuration**

The following parameters must be configured:

Variance rate limit

- **Max falling rate**: Defines the maximum falling rate of the system.
- **Max rising rate**: Defines the maximum rising rate of the system.

Square error compensation: Compensation in the pressure square error applied when **Veronte Autopilot 1x** is close to the ground. This is to correct the Ground Effect for landing purposes.

- **Square error**: This value is automatically calculated from the square error of the static pressure sensor (see *Static Pressure sensor block*).

- **Altitude correction threshold (Positive down)**: The user has to define an Altitude Difference for the system to apply the Ground Effect. While landing, the aircraft will feel a increase in the static pressure due to the Ground Effect, and this pressure difference (transformed into meters) is the Threshold that can be configured here. If set to 0, whenever Ground Effect is enabled, it will make its effect.

  In other words, Altitude error is the measurement from the static pressure sensor in meters minus the estimated state of the UAV.

Explanation

The ground effect creates a high pressure below the UAV when it is close to the ground, this increase in pressure readings produces the navigation to "go down". Veronte Autopilote 1x can mitigate the ground effect in two ways:

- 1: Increasing the static pressure sensor variance (R) used for the Extended Kalman filter. This means that other height sources, for example GNSS will be used more strongly to estimate the altitude near the ground. This is configured with the boxes "Square error" and "Altitude correction threshold (Positive down)". If enabled, the R used in the EKF for the static pressure sensor will be increased to be the value configured in the box when the "altitude error" measured by the static pressure sensor is higher than the configured threshold in the down direction. Example, if the current estimated state of the UAV is 10 meters MSL and the measurements from the static pressure sensor tell that the UAV is at 5 meters MSL. If the "Altitude correction threshold (Positive down)" is less than 5 then the R used for this sensor will be the one configured in the "Square error". Please notice that ground effect correction must only be enabled when close to the ground so that the navigation performance is not negatively impacted when there is no ground effect.
- 2: Modifying the actual measurement of the static pressure sensor. When this function is enabled the "altitude error" corresponding to the static pressure sensor is modified according to the **Correction compensation** table presented below. The idea is that altitude errors that would make the estimation of the height to go down are changed to reduce the altitude error. It is also important to only enable this when close to the ground. Using the same example as before, the 5 meters down of altitude error for this sensor would be transformed to only 4 meters for example, that way this sensor would pull down the estimated altitude a little bit less.

Note: Please note that these two ways of compensating the ground effect can be enabled/disabled separately and that they only have to be used when close to the ground. It is recommended to test in controlled conditions.

- **Decimation**: Parameter that defines the group of measurements for which 1 value will be stored. For example, if the decimation is 10, every 10 measurements will be counted as 1. This procedure is used **to reduce the number of samples**. **Recommended value 10** (in decimal units).

  If users have any questions about decimation, please refer to the *What does decimation mean? - FAQ* section of this manual.

Correction compensation: Users can edit the correction compensation by clicking here:

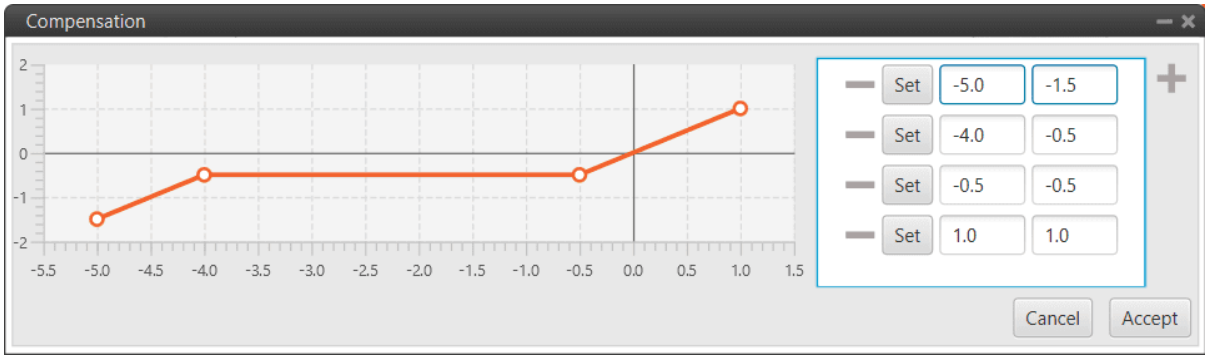


Fig. 382: **Static Pressure block configuration - Correction compensation table**

The user can add or remove points from the correction compensation table. The correction will have the 'shape' made by these points.

The **altitude error** used is computed from the actual altitude error by interpolating in the table (with extrapolation).

- X-axis: Actual altitude error
- Y-axis: Actual error used in EKF

Note: The value entered in column Y is an adjustment of how "strong" the ground effect compensation is.

The unit of measurement is meters.

2.9.10.1.6 Terrain height

Terrain height block transforms from **terrain height** measurement to **EKF data for terrain height update**.

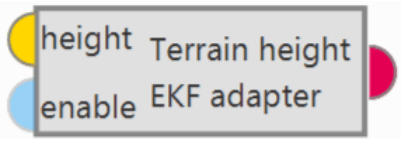


Fig. 383: **Terrain height block**

- **Inputs**

  height: **Terrain altitude as a 4-dimensional real array** with the following components:

  - 0: Update flag (always 1)

  - 1: Valid flag (inside mesh)

  - 2: Terrain height

  - 3: Variance

  This input corresponds to the *SRTM height sensor block*.

  (Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

  Not connected means enabled.

- **Output**

  Pin 0: EKF input data (H, R, y).

  Note:

  - **H**: Observation matrix.
  - **R**: Measurement covariance matrix.
  - **y**: Measurement.

- **Configuration menu**: The user must configure the decimation of this sensor.

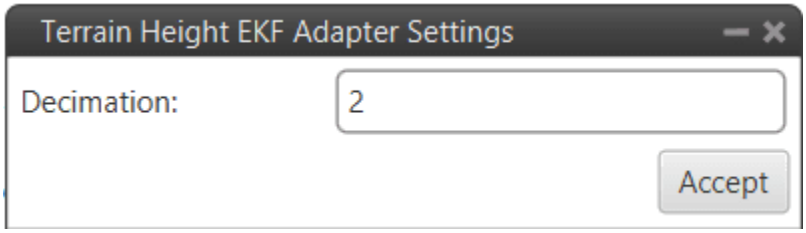


Fig. 384: **Terrain height block configuration**

This parameter defines the group of measurements for which 1 value will be stored. For example, if the decimation is 10, every 10 measurements will be counted as 1. This procedure is used **to reduce the number of samples**. **Recommended value 2** (in decimal units). If users have any questions about decimation, please refer to the *What does decimation mean? - FAQ* section of this manual.

2.9.10.1.7 Velocity

Velocity block converts **velocity** measurement into **EKF data for velocity update**.

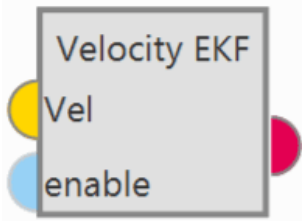


Fig. 385: **Velocity block**

- **Inputs**

  Vel: Velocity measurement as a **12-dimensional real array** with the following components:

  - 0: Update flag
  - 1: Fix flag
  - 2: North velocity
  - 3: East velocity
  - 4: Down velocity

- 5: North velocity variance
- 6: East velocity variance
- 7: Down velocity variance
- 8: X body antenna position
- 9: Y body antenna position
- 10: Z body antenna position
- 11: Measurement delay

This input corresponds to the *GNSS Sensor block*.

(Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

Not connected means enabled.

- **Output**

Pin 0: EKF input data (H, R, y).

Note:

- **H**: Observation matrix.
- **R**: Measurement covariance matrix.
- **y**: Measurement.

- **Configuration menu**:

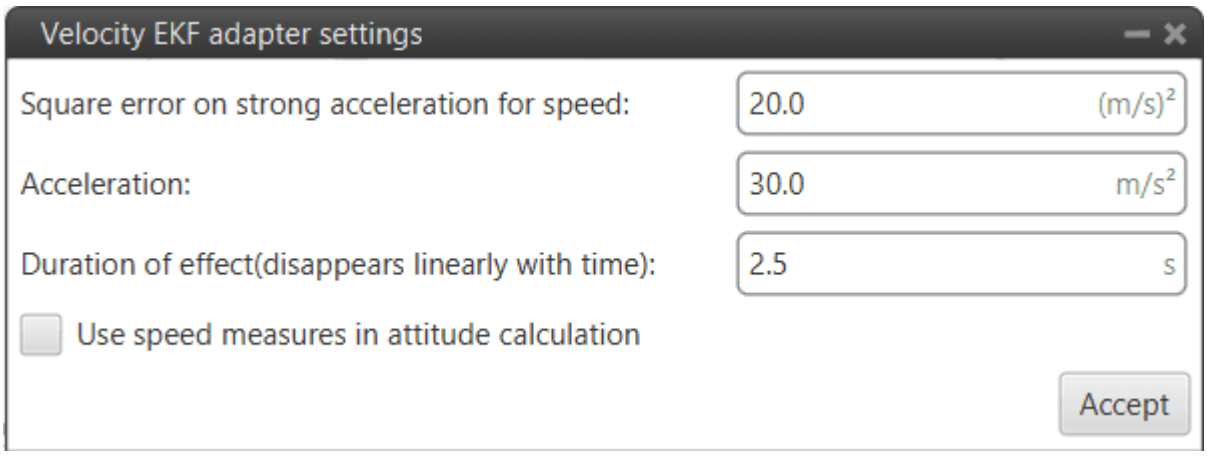


Fig. 386: **Velocity block configuration**

Users must configure the following parameters:

- **Square error on strong acceleration for speed**: Under strong acceleration the variance for the speed solution is changed to the specified value.
- **Acceleration**: Threshold definition. When this threshold is exceeded, strong acceleration variances are considered.
- **Duration of effect (disappears linearly with time)**: Time required to restore the default variances of the GNSS solution.

– **Use speed measures in the attitude calculation**: When enabled, the speed data from the GNSS solution is considered for the attitude estimation.

2.9.10.1.8 Velocity down

Velocity down block adapts the **velocity down** measurement to **EKF data for velocity down update**.

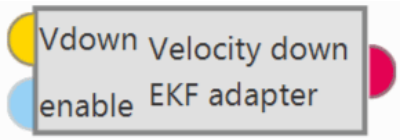


Fig. 387: **Velocity down block**

- **Inputs**

  Vdown: Velocity measurement down measurement.

  (Optional) **enable**: Optional boolean input to enable (true) or disable (false) the input in the EKF.

  Not connected means enabled.

- **Output**

  Pin 0: EKF input data (H, R, y).

  Note:

  – **H**: Observation matrix.

  – **R**: Measurement covariance matrix.

  – **y**: Measurement.

- **Configuration menu**:

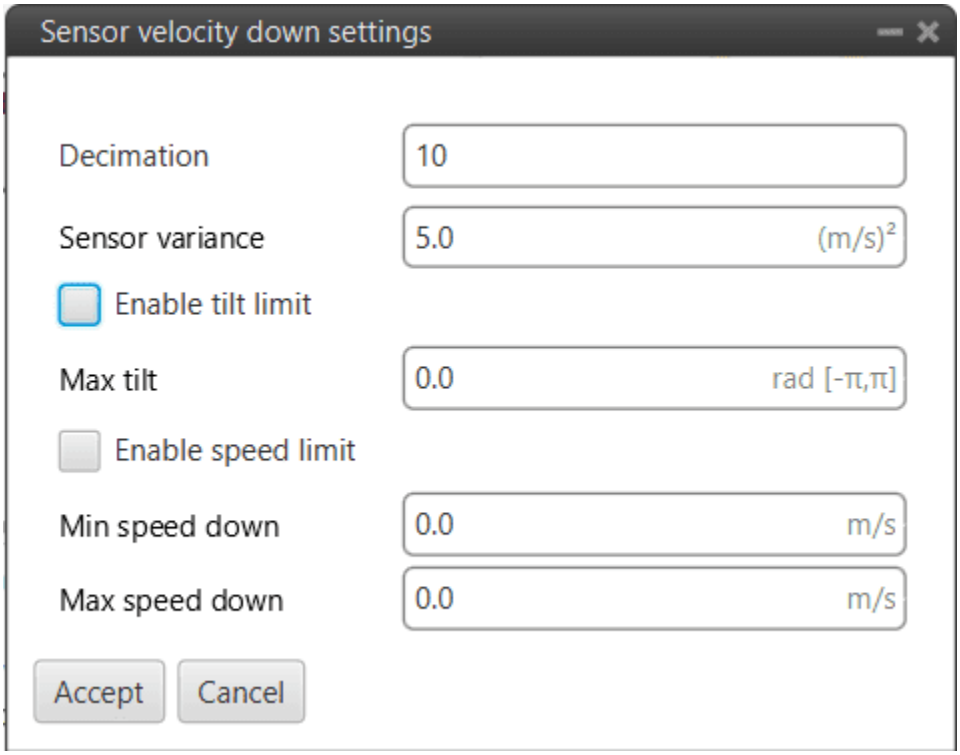


Fig. 388: **Velocity down block configuration**

The following parameters must be configure:

- **Decimation**: Parameter that defines the group of measurements for which 1 value will be stored. For example, if the decimation is 10, every 10 measurements will be counted as 1. This procedure is used **to reduce the number of samples**. **Recommended value 10** (in decimal units).

  If users have any questions about decimation, please refer to the *What does decimation mean? - FAQ* section of this manual.

- **Sensor variance**: Sensor error variance.
- **Enable tilt limit**: The sensor measures the variable in a direction perpendicular to the longitudinal axis of the platform, so when it is tilted the reading will not be reliable. This option allows the definition of a tilt limit, so that if the limit is reached, the sensor reading will be discarded.
- **Max tilt**: A maximum tilt can be defined.
- **Enable speed limit**: This option allows a speed limit to be set, so that if the limit is reached, the sensor reading will be discarded.
- **Min speed down**: Defines the minimum limit of the speed measured by the sensor.
- **Max speed down**: Defines the maximum limit of the speed measured by the sensor.

2.9.10.2 EKF Split

EKF Split block shows all the **sensor information** that is going into the EKF algorithm.

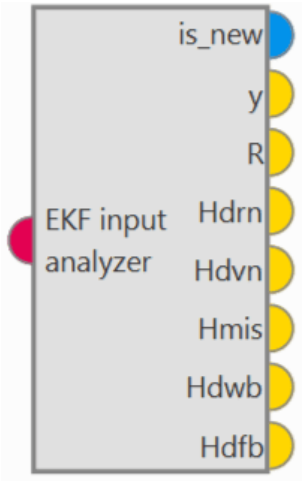


Fig. 389: **EKF Split block**

- **Input**
  - **Pin 0**: EKF input data **from any EKF input adapter block** (H, R, y).
- **Outputs**
  - **is_new**: Update flag (**true** when a **new measurement is generated**, false otherwise).
  - **y**: Measurement. (3-dimensional real array)
  - **R**: Measurement **covariance matrix**. (3-dimensional real array)
  - **Hdrn**: Observation matrix for the **position increment states**. (9-dimensional real array)
  - **Hdvn**: Observation matrix for the **velocity increment states**. (9-dimensional real array)
  - **Hmis**: Observation matrix for the **misalignment states**. (9-dimensional real array)
  - **Hdwb**: Observation matrix for the **gyroscope bias increment states**. (9-dimensional real array)
  - **Hdfb**: Observation matrix for the **terrain altitude increment state**. (9-dimensional real array)

This block is designed for any EKF adapter block, i.e. a standard block for all EKF adapter blocks, therefore some outputs only correspond to a certain type of EKF adapter block. And, consequently, **the outputs that do not correspond to the connected block, will get 0 as value**.

> **Attention:** Be careful! Check that the **blocks to which the outputs are connected match in size with the data**, i.e. if the output is a 9-dimensional real array, a split block or a multiple user real variable with size 9 must be connected to it in order to display the data.

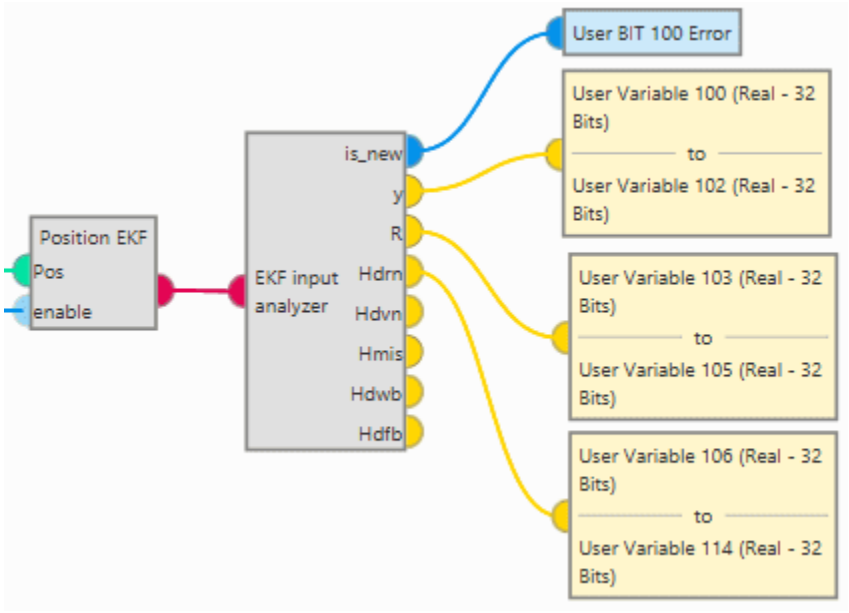


Fig. 390: **EKF Split block - Example**

2.9.10.3 Navigation

Navigation block **updates the Veronte navigation variables** (position, velocity, attitude, etc.) based on the **current selected navigation source**.

Note: This block has by default 1 input, in its configuration the user must set the desired size of inputs.

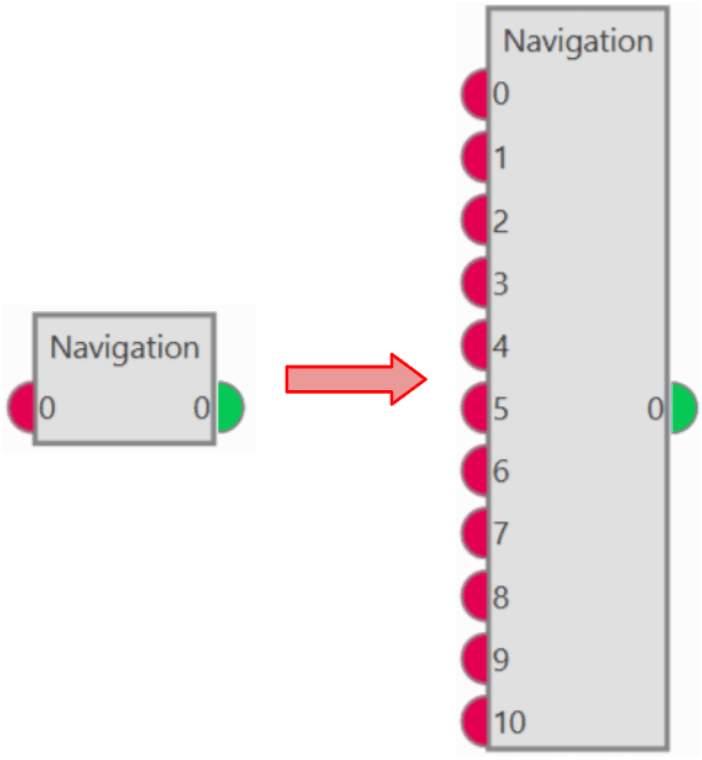


Fig. 391: **Navigation block**

Veronte Autopilot 1x integrates a navigation system that can operate with GPS and without GPS coverage.

In the navigation with GPS, the system uses it to fly the aircraft along a route or to a given waypoint. It is possible to control the aircraft's position (longitude and latitude) and the altitude. This is the navigation used by default by the system when everything is working properly.

In case the GPS signal is lost, the navigation can easily measures the attitude angles with a greater precision than using a simple IMU. With these measures, it is possible for the system to control pitch, roll and yaw and then maintain a safe attitude when the GPS signal is lost, avoiding any possible malfunctions. It is recommended to create an automation to change to a phase where the attitude angles are controlled, in case of loss of GPS signal. For more information visit *Automations*.

Note: The yaw can be measured in the navigation without GPS only if the magnetometer is activated in the navigation window.

- **Input**

  0: EKF input data **from any EKF input adapter block**.

- **Output**

  0: Index + 1 of the current input measurement used to update the EKF state. A value of **zero means that no measurements have been input to the EKF in the current step**.

- **Configuration menu**: This menu contains the parameters used in the Kalman Filter algorithm to fuse the information provided by the different sensors. This data is used in the navigation system to generate the commands sent to the aircraft.

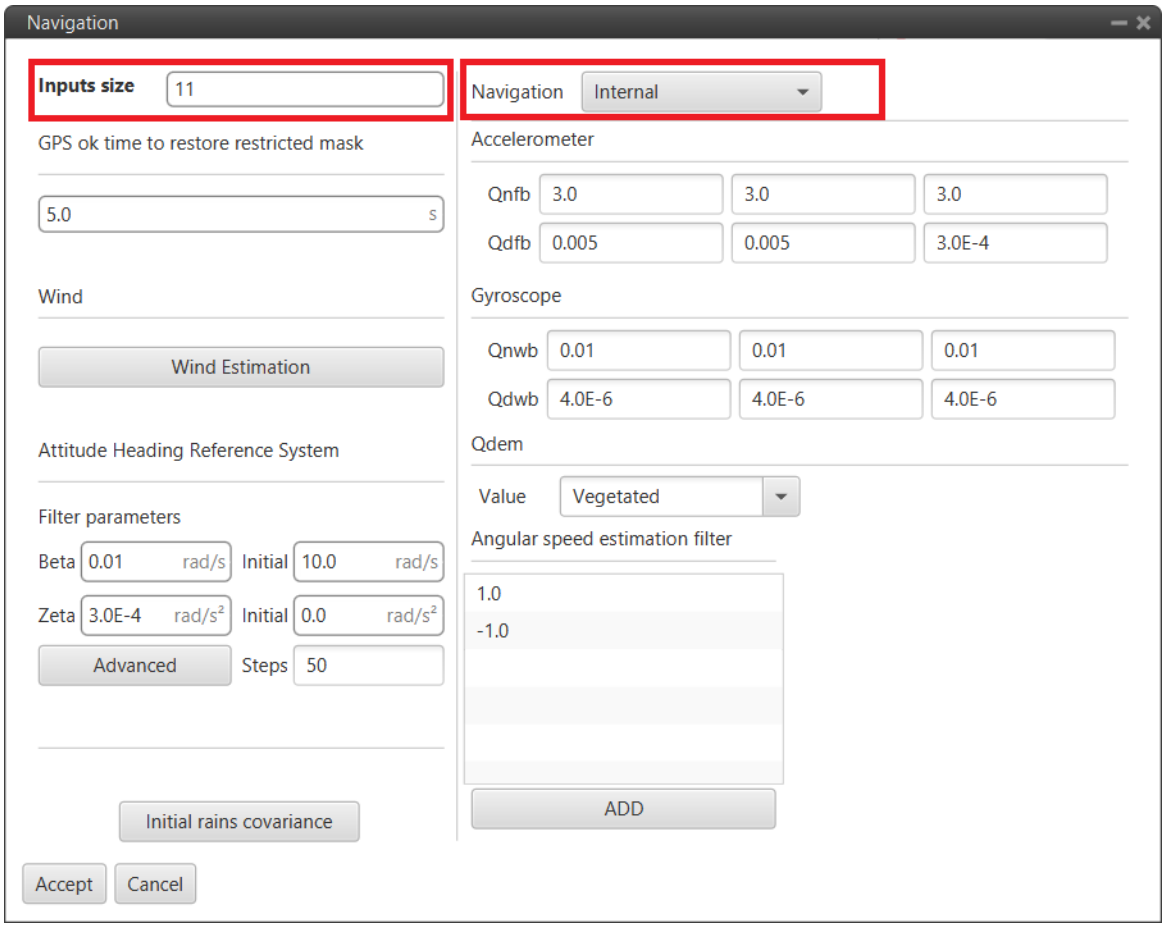


Fig. 392: **Navigation block configuration**

> **Warning: The values that appear here should only be changed by advanced users**. If the user is not familiar with the Kalman Filter algorithm and Sensor Fusion, do not change the default parameters.
>
> If further information is required, please contact the support team (create a ticket in the customer's **Joint Collaboration Framework**; for more information, see Tickets section of the **JCF** manual).

Although this configuration menu must be set by default or modified by advanced users, the **2 following parameters** must be **configured by all users**:

- **Inputs size**: Users must set the number of inputs.
- **Navigation**: The type of navigation must be selected, tha available options are:
  - **Internal**: Uses internal data for navigation. Data (position, attitude, etc.) is processed into 1x unit from sensor measures.
  - **External VCP**: Uses external data for navigation. Data (position, attitude, etc.) is provided by Veronte Communication Protocol (VCP).
  - **External Var**: Uses external data for navigation.

    It takes directly the **attitude, velocity and acceleration data** of the following real variables from the memory:

- **ID 259**: External yaw
- **ID 260**: External pitch
- **ID 261**: External roll
- **ID 262**: External roll rate
- **ID 263**: External pitch rate
- **ID 264**: External yaw rate
- **ID 265**: External velocity north
- **ID 266**: External velocity east
- **ID 267**: External velocity down
- **ID 268**: External acceleration x body axis
- **ID 269**: External acceleration y body axis
- **ID 270**: External acceleration z body axis
- **ID 271**: External GPS Time of Week

Position data is read from the **Moving Feature 00**.

* **Vectornav VN-300**: Uses external data for navigation. Data (position, attitude, etc.) is provided by Vectornav VN-300. For more information, see the *Vectornav VN-300 - Integration examples* section of this manual.

Wind

By clicking on the 'Wind Estimation' button, a configuration menu will be displayed:

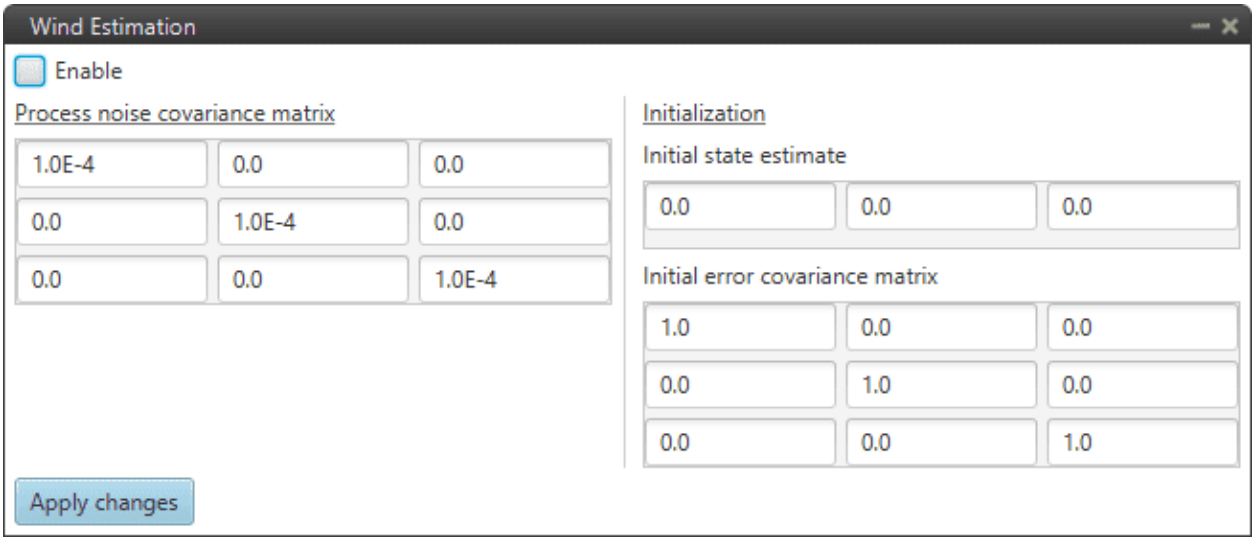


Fig. 393: **Wind Estimation configuration**

To make a proper estimate, the system needs to collect as much wind information as possible, so missions with a trajectory involving changes in directions will result in a better wind estimate compared to a straight trajectory mission.

The computed result is displayed in the variables: **Wind Velocity Down**, **Wind Velocity East**, **Wind Velocity North**.

Warning: **The values that appear here should only be changed by advanced users**.

2.9.11 Positions blocks

Position blocks allow to operate with position variables.

Note: In **1x PDI Builder**, position variables are also refered to as **Features**.

2.9.11.1 Constant Position

Constant Position block defines a position on Earth using Latitude, Longitude and WGS84 Height.

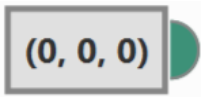


Fig. 394: **Constant Position block**

- **Output**
  - **Pin 0**: Output of the configured position.
- **Configuration menu**:

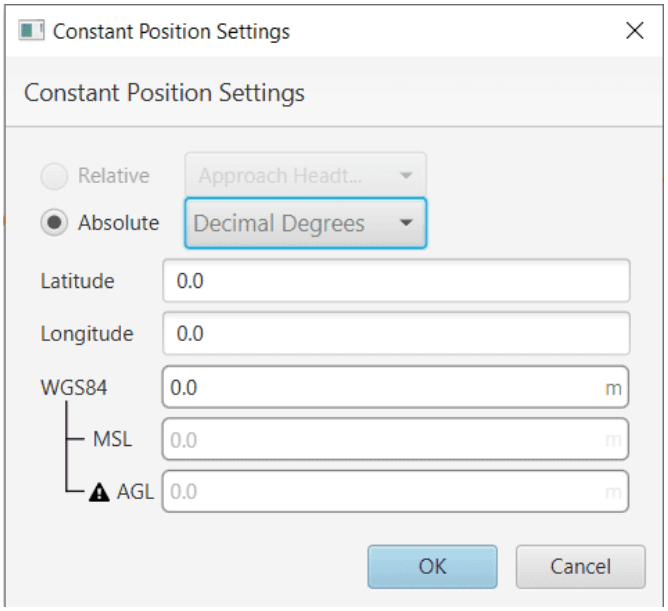


Fig. 395: **Constant Position block configuration**

The following parameters must be configured to define the desired **absolute position**:

- The coordinates can be set in **UTM**, **MGRS** (Military Grid Reference System), **Decimal Degrees** or **Degress º ‘ ”**.

- They are indicated through the **latitude**, **longitude** and **altitude**.
- **WGS84**: The first time, the altitude must be defined with respect to the ellipsoid, WGS84. After this, MSL and AGL values will be calculated automatically and the user will also be able to define the altitude with respecto to the sea level, MSL.

2.9.11.2 Move

Move block outputs the position which is the result of moving the input position with the displacement of the input vector.

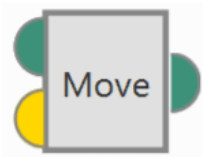


Fig. 396: **Move block**

- **Inputs**
  - **Pin 0**: Input position.
  - **Pin 1**: Input displacement vector in **NED frame**. It must be a vector of 3 elements, i.e. a 3x1 vector.
- **Output**
  - **Pin 0**: Output position.

2.9.11.3 Relative Vector

Relative Vector block calculates the relative vector in NED frame from the two input positions.

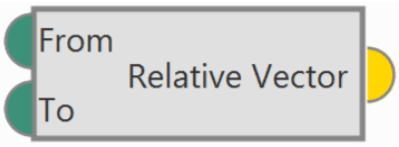


Fig. 397: **Relative Vector block**

- **Inputs**
  - **From**: First position.
  - **To**: Second position.
- **Output**
  - **Vector**: Distance vector from the first position to the second position.

2.9.11.4 Read Feature

Read Feature block reads a position from a FID (feature) variable.

Fig. 398: **Read Feature block**

- **Output**

  Pin 0: Position to read.

- **Configuration menu**:

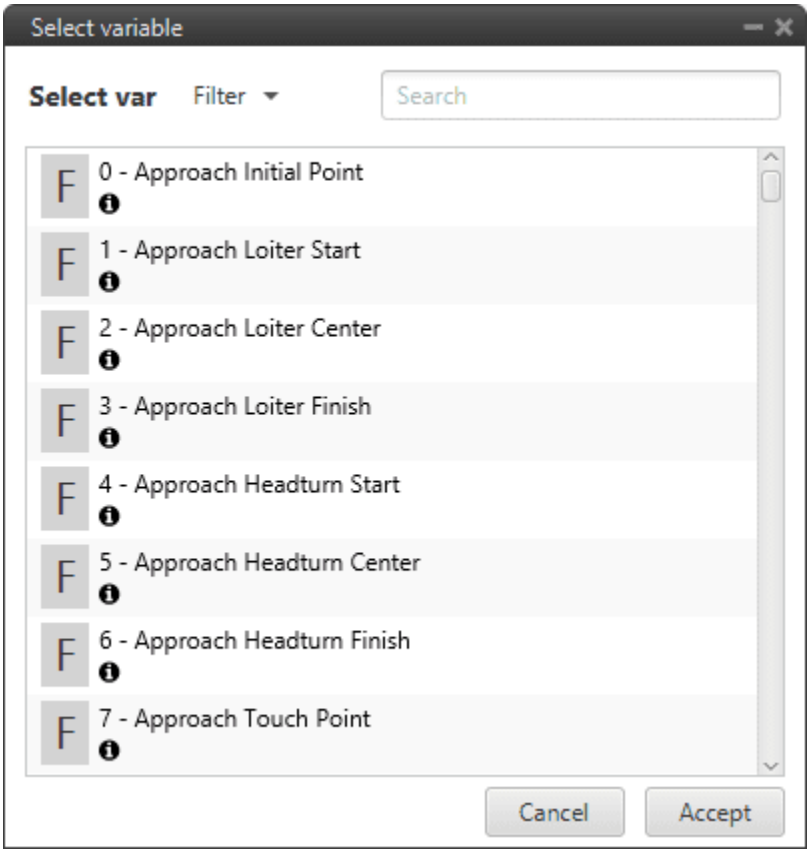


Fig. 399: **Read Feature block configuration**

Users must select the **feature variable** where the position **is stored**.

2.9.11.5 Write Feature

Write Feature block writes a position to a FID variable.

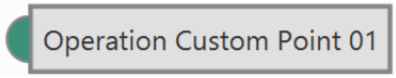


Fig. 400: **Write Feature block**

- **Input**
  - **Pin 0**: Position to write.
- **Configuration menu**:

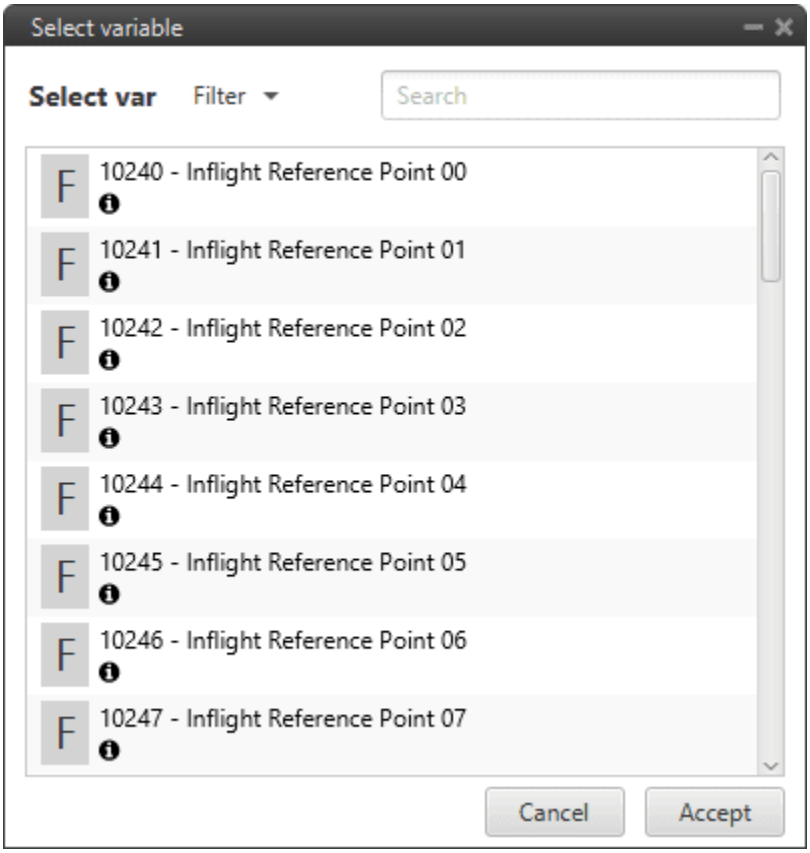


Fig. 401: **Write Feature block configuration**

Users must select the **feature variable** where the position is **to be stored**.

2.9.12 Sensors blocks

Sensors blocks allows to configure any sensor connected externally or internally to Veronte Autopilot 1x.

As the outputs of these blocks are not 'ready' to be directly implemented in the navigation block (EKF algorithm), these blocks are usually connected afterwards to *EKF Adapters blocks*.

2.9.12.1 Altimeter

Altimer sensor block configures the parameters of external altimeters.

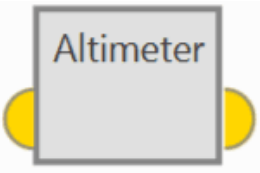


Fig. 402: **Altimeter block**

- **Input**
  - **Pin 0**: Sensor variance.
- **Output**
  - **Pin 0**: Measurement as a **3-dimensional real array** with the following components:

  - 0: Update flag

  - 1: Altitude measurement

  - 2: Variance
- **Configuration menu**:

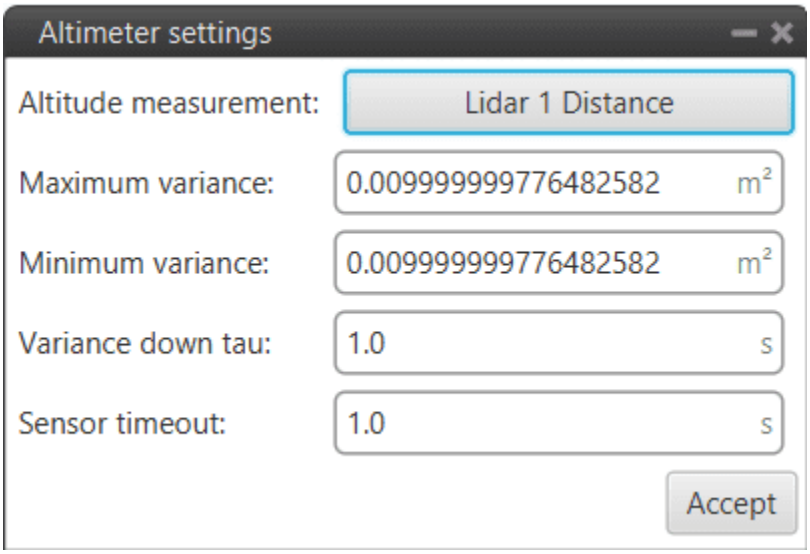


Fig. 403: **Altimeter block configuration**

The following parameters must be configured:

- **Altitude measurement**: A real variable must be selected from which the altitude measurement is read. The parameters defined here shall be applied to this input variable.

  Note: This is usually the measurement of a Lidar sensor.

- **Maximum variance**: Maximum variance applied to the measurement after measurement lost.
- **Minimum variance**: Minimum variance applied to the measurement after recovering from a measurement lost.
- **Variance down tau**: Filter constant. Smoothing parameter for the transition from maximum to minimum variance.
- **Sensor timeout**: Time before considering measurement lost.

  Note: If the measurement does not change during this time, the autopilot may consider that no measurement is entered, and therefore, the timeout is fulfilled.

The **logic** of this block is summarized below:

1. Get Altitude data (usually from a Lidar sensor)
2. Calculate the time difference between the current time and the previous time the altitude data was acquired:
   - If this time is longer than "Sensor timeout", set the variance to "Maximum variance".
   - If this time is not longer than "Sensor timeout", check if the current variance is lower than the previous variance and apply an EWMA filter with "tau", previous variance and current variance.

   Note: Note that the current variance must be lower than the previous one but never lower than the "Minimum variance"; if it is, it will get clamped to it.

5. Feed *Altitude EKF adapter block* with "Altitude measurement" and "variance".
6. Then, **EKF** will fuse SRTM and Altitude measurement together giving more weight to the one with lower variance.

2.9.12.2 GNSS sensor

GNSS sensor block configures GNSS receivers, RTK and External source.

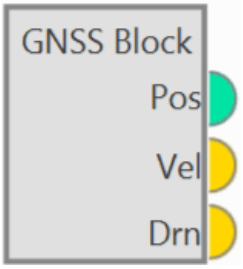


Fig. 404: **GNSS sensor block**

- **Outputs**

- **Pos**: Absolute position measurement.
- **Vel**: Velocity measurement as a **12-dimensional real array** with the following components:
  - 0: Update flag
  - 1: Fix flag
  - 2: North velocity
  - 3: East velocity
  - 4: Down velocity
  - 5: North velocity variance
  - 6: East velocity variance
  - 7: Down velocity variance
  - 8: X body antenna position
  - 9: Y body antenna position
  - 10: Z body antenna position
  - 11: Measurement delay
- **Drn**: Relative position measurement as a **10-dimensional real array** with the following components:
  - 0: Update flag
  - 1: Rover flag (one is rover, zero is base)
  - 2: Time stamp
  - 3: North relative distance
  - 4: East relative distance
  - 5: Down relative distance
  - 6: Relative distance variance
  - 7: X body antenna position
  - 8: Y body antenna position
  - 9: Z body antenna position

- **Configuration menu**:

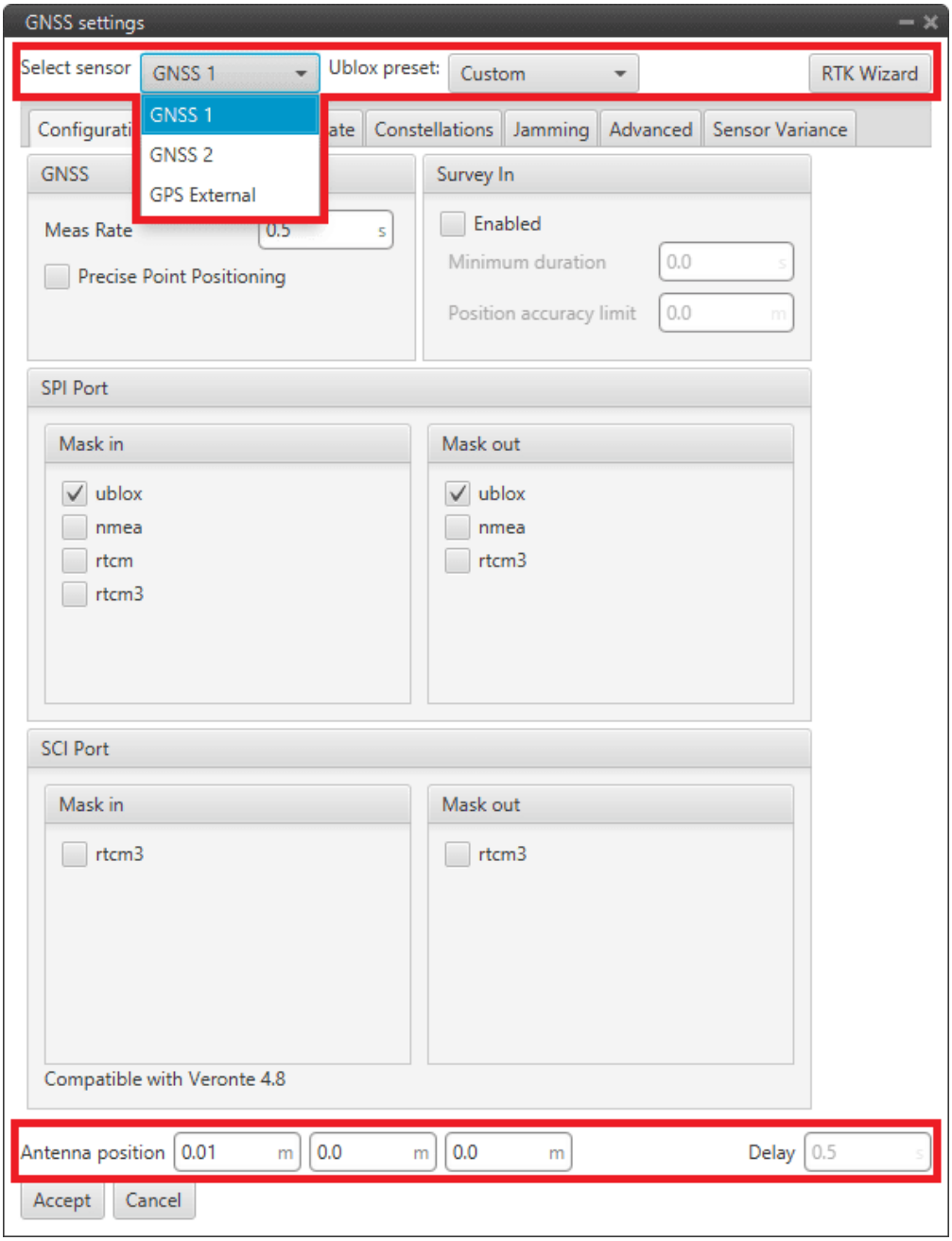


Fig. 405: **GNSS sensor block configuration**

The parameters to be configured in the first and last row of this configuration menu are presented below:

– **Select sensor**: The first parameter to select is the GNSS sensor: **GNSS 1, GNSS 2** or **GPS External**. Depending on the sensor selected, the block will change name and configuration menu.

However, GNSS 1-2 have the same configuration menu.

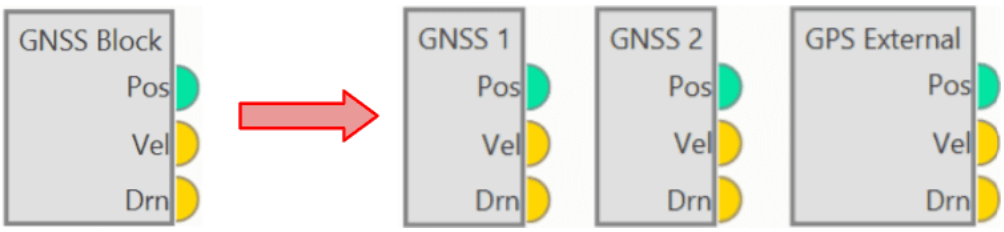


Fig. 406: **GNSS sensor blocks**

– **Ublox preset**: By default, *Custom* is selected. If an **option other than** *Custom* is selected, users will **only be able to configure the 'Sensor Variance' tab** (this tab will be described below).

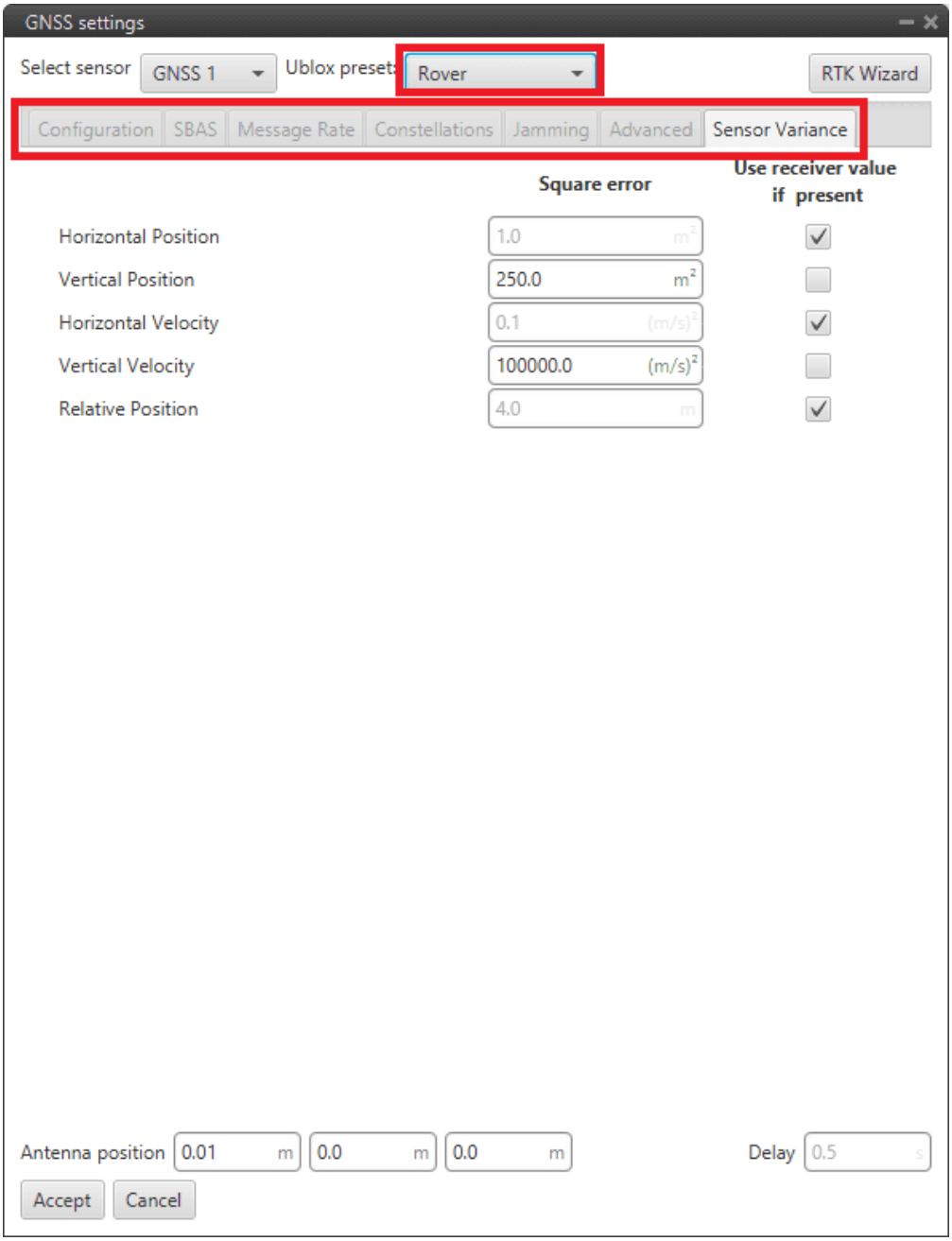


Fig. 407: **GNSS sensor block configuration - Rover/Dynamic base/Static base**

* **Custom**: This option allows the user to modify all the tabs that appear in this menu (all tabs are described below).
* **Rover**: By choosing this option, a default 'Rover' configuration will be selected for this block.

  This corresponds to 'MOVING BASE → Rover' units and 'STATIC BASE' AIR units of the RTK wizard (described below).
* **Dynamic base**: By choosing this option, a default 'Dynamic base' configuration will be selected for this block.

  This corresponds to 'MOVING BASE → Base' units of the RTK wizard (described below).

* **Static base**: By choosing this option, a default 'Static base' configuration will be selected for this block.

  This corresponds to GND units of the RTK wizard (described below).

For more information on these default configurations, click here (go to section 3.1.5).

– **RTK Wizard**: This interface helps the user configuring everything related to RTK or GNSS Compass. By clicking here, the configuration menu will be displayed:

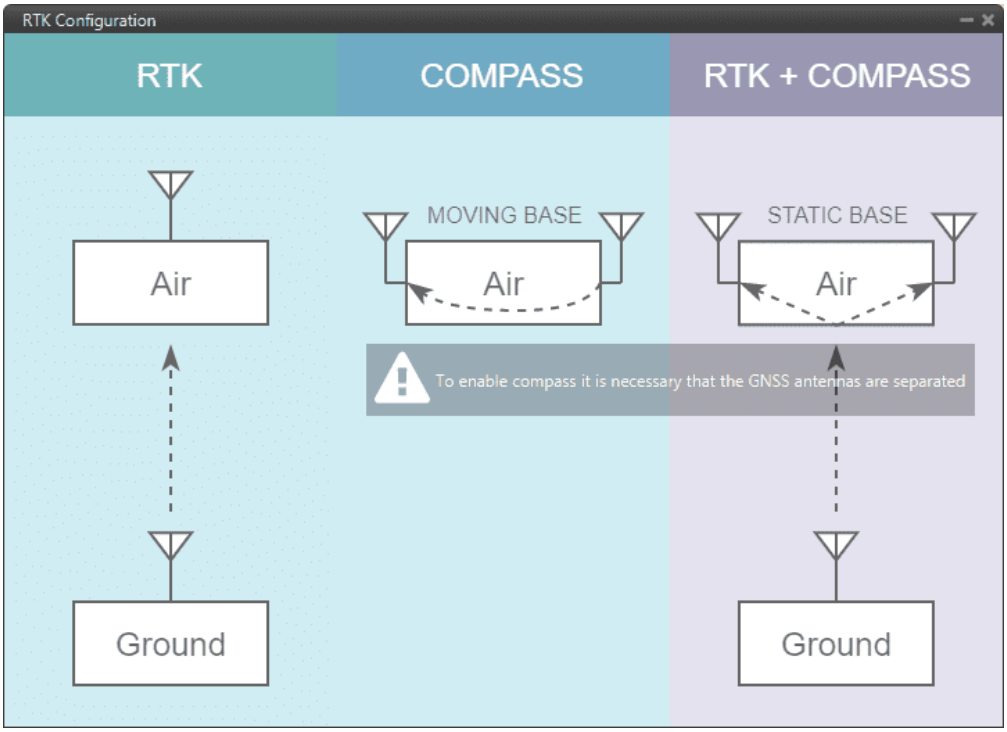


Fig. 408: **GNSS sensor block configuration - RTK Wizard**

In this menu, the user can find 3 different options:

* **RTK**: Stands for **Real Time Kinematics** (RTK) and it is a satellite navigation technique used **to enhance the precision of position data derived from satellite-based positioning systems**.

  To work, it requires 2 GNSS receivers placed in different autopilots.

  By clicking on 'Air', a default 'Custom' configuration will be loaded in this block. Likewise, by clicking on 'Ground', a default 'Static base' configuration will be selected.

* **Compass**: The GNSS compass provides accurate dual antenna GNSS-based heading that is not subject to magnetic interference.

  It requires 2 GNSS receivers placed on the same autopilot to work.

  By clicking on 'Air', a default 'Dynamic base' configuration will be loaded if this GNSS receiver has been selected as base. Otherwise, if this GNSS receiver has been selected as rover, a default 'Rover' configuration will be loaded.

* **RTK + Compass**: Hybrid combination where both tools are employed at the same time in a system where the **AIR unit must have 2 GNSS receivers** and the **GND must have**, at least, **1 GNSS reseiver**.

By clicking on 'Air', a default 'Rover' configuration will be loaded in this block. Likewise, by clicking on 'Ground', a default 'Static base' configuration will be selected.

- **Antenna position**: It is used to set the relative distance to the center of mass from the GNSS antenna in aircraft body axis. This parameter has to be set correctly in order to get a correct value when using GNSS Compass.
- **Delay**: Delay with which the GPS information is 'picked up', as the GPS may have a small delay while it reads and processes the information. In case the user has selected **GNSS 1 or GNSS 2**, the internal GPS of the Veronte Autopilot 1x has a default **delay of 0.5 seconds**.

- **GNSS 1-2 configuration menu**:
  - **Configuration**: This menu contains some of the parameters needed to configure the GNSS 1-2 receiver located in Veronte Autopilot 1x.

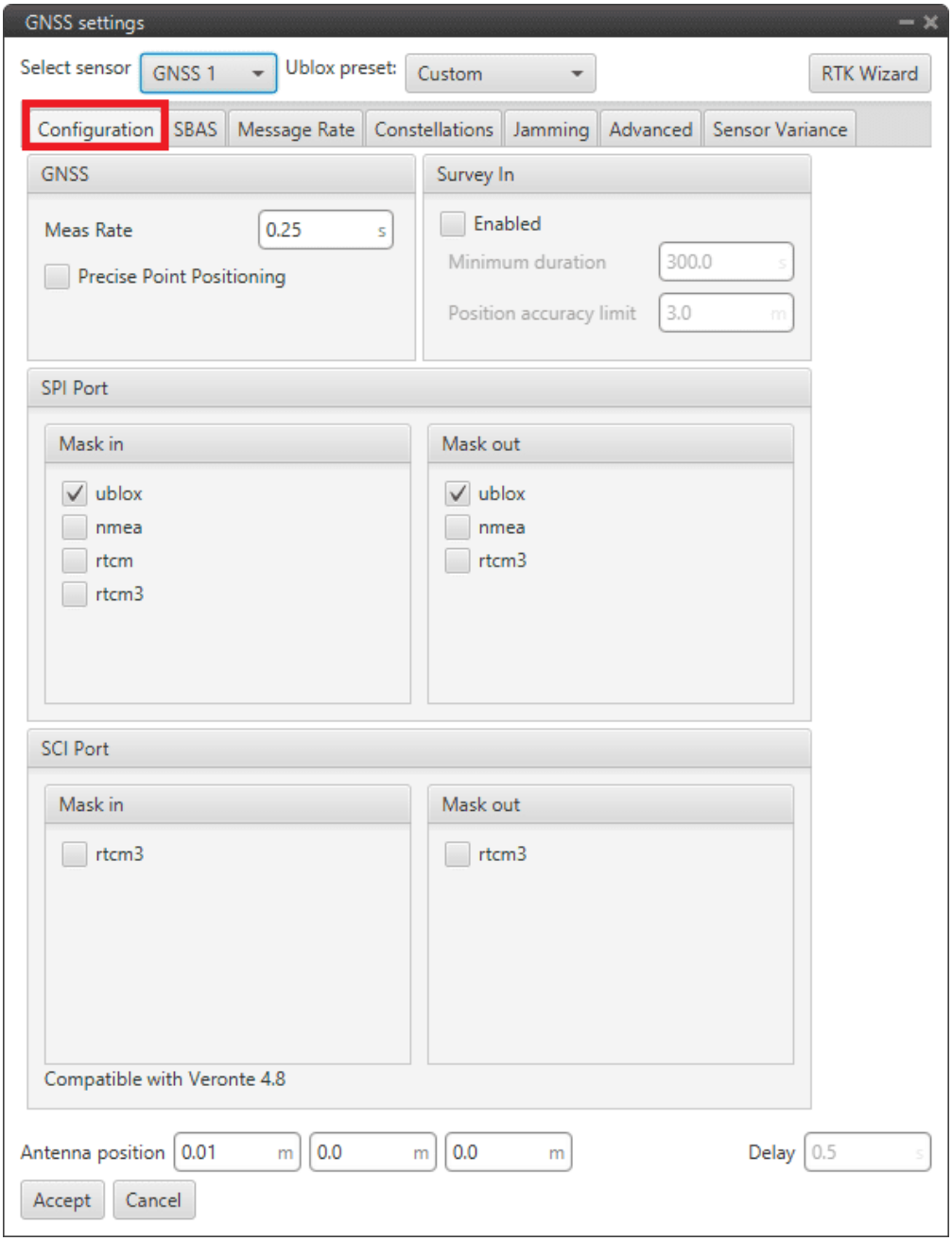


Fig. 409: **GNSS 1/2 sensor block configuration - Configuration tab**

The following parameters are configurable:

* **GNSS**: Data values that can be configured.
  - **Meas Rate**: Defines the minimum time between data acquisition.
  - **Precise Point Positioning (PPP)**: This option is a precise global positioning service. PPP is able to **provide centimetre to decimetre level positioning solutions** after a few minutes with an unobstructed view of the sky.
* **Survey In**: Determines the position of a stationary receiver by building a weighted average of all valid

3D position solutions.

This mode should be activated on a Ground unit to enable GNSS Differential mode and send corrections to the Air unit. **Two requirements must be specified to stop the procedure.** Survey in procedure shall **end when both requirements are met**:

- **Minimum duration**: Defines a minimum amount of observation time independent of the actual number of valid fixes that were used for the position calculation.

  Reasonable values range from **one day for high accuracy** requirements to a **few minutes for approximate position determination**.

- **Position accuracy limit**: Defines a limit on the dispersion of positions contributing to the calculated mean.

* **SPI Port**: Allows the user to select the different comunication protocols as input or output. One port can handle several protocols at the same time (e.g. NMEA and UBX).

  - **Mask in**: Defines the inputs, i.e., receives the data (usually the air unit). The available protocols are UBLOX, NMEA, RTCM and RTCM3.
  - **Mask out**: Defines the outputs, i.e. sends the data (usually the ground unit). The available protocols are UBLOX, NMEA and RTCM3.

  More information on protocols and configuration can be found in the U-blox documentation.

* **SCI Port**: In this case, RTK messages that are sent through the RTCM3 protocol are connected directly through an SCI port, so they do not occupy the bandwidth of the SPI port.

Note: Only for Veronte Autopilot 1x `hardware version 4.8 and higher`.

– **SBAS**: SBAS stands for **Satellite Based Augmentation System**. It is a set of geostationary satellites that are used to check the status of the signals sent by GPS Satellites and to improve tracking by correctiong for atmospheric disturbations, orbit deviations, clock errors, etc.

In **1x PDI Builder**, it is possible to select the satellites to be used for this purpose by **selecting the numbers listed in the table** in the figure below or have the **software choose them automatically** according to the location of the platform.

The automatic option is recommended.

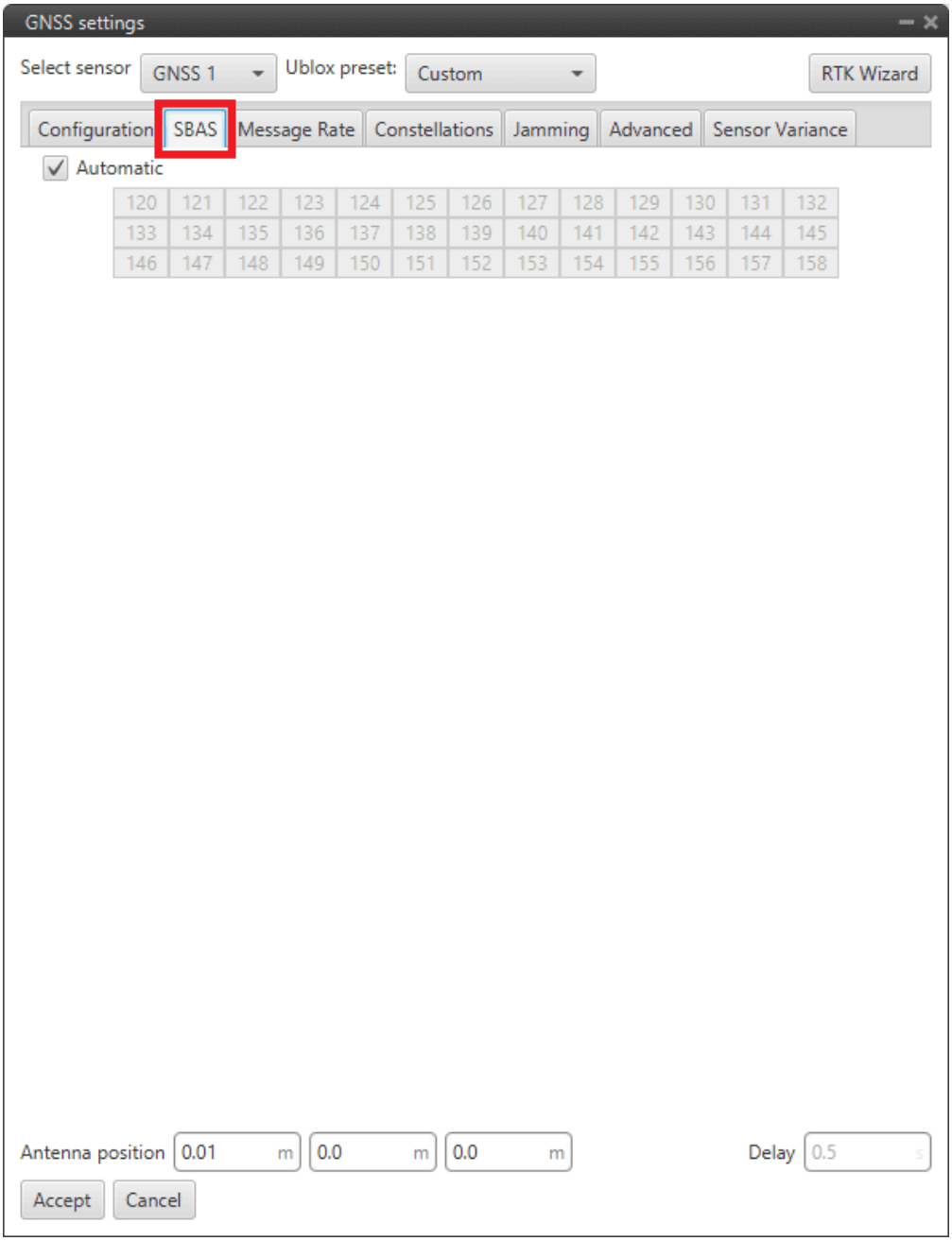


Fig. 410: **GNSS 1/2 sensor block configuration - SBAS tab**

– **Message Rate**: The Message rate options are used to set the **time between the messages received on the autopilot**. Each of the different messages can be configured separately: ECEF (Earth Centred Fixed Reference Frame), LLH (Latitude, Longitude and Height), Speed, GPS Time, SV Status (status of the GPS satellite), etc.

Fig. 411: **GNSS 1/2 sensor block configuration - Message Rate tab**

More information on the list of RTCM 3 messages can be found here.

– **Constellations**: In this tab, the user can select which GNSS constellations are being used from the supported constellations listed in the figure below:

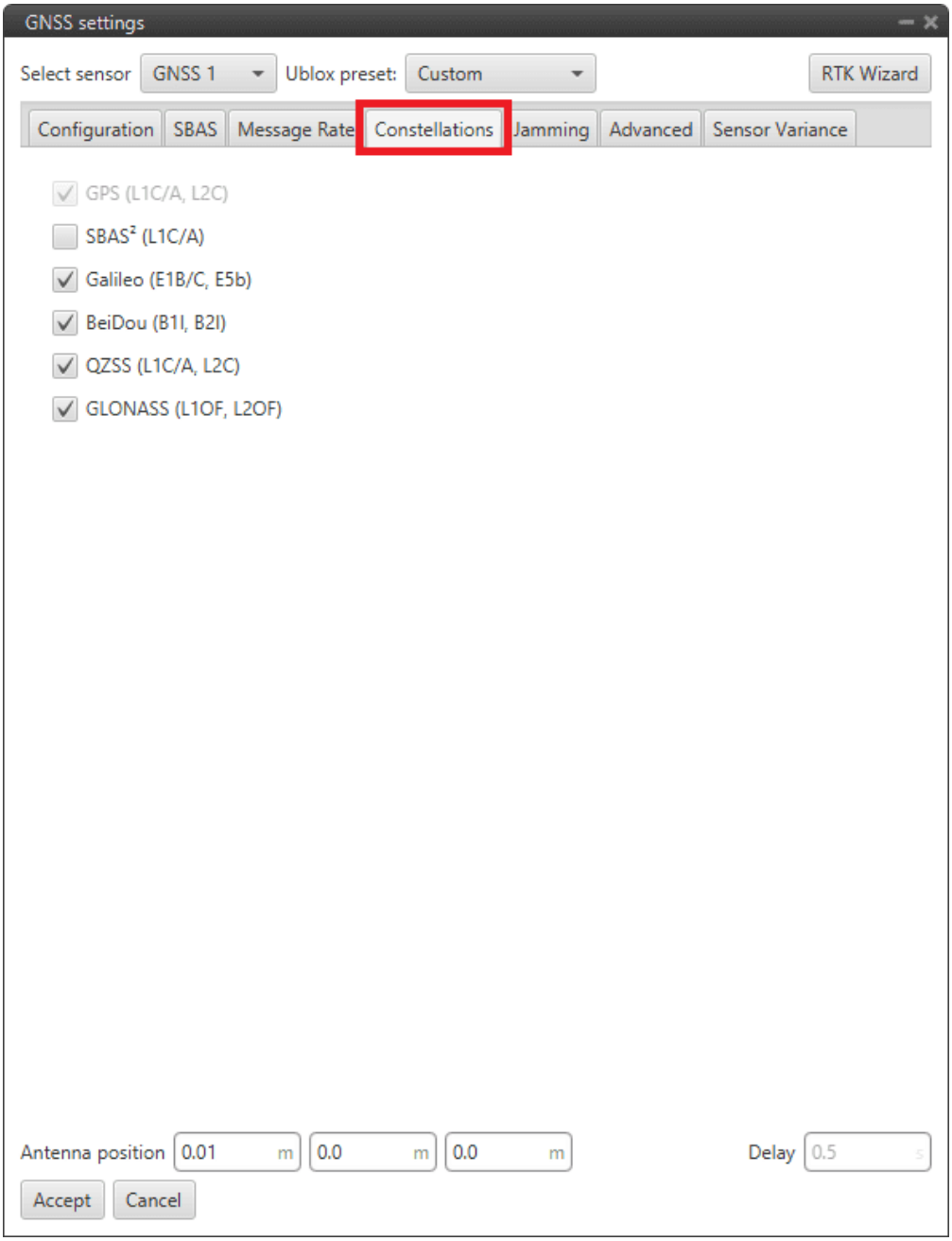


Fig. 412: **GNSS 1/2 sensor block configuration - Constellations tab**

* **SBAS**
* **Galileo**
* **BeiDou**
* **QZSS**
* **GLONASS**

– **Jamming**: This menu allows the user to configure an indicator for both broadband and continuous wave

(CW) jammers/interference. The receiver monitors the background noise and looks for significant changes.

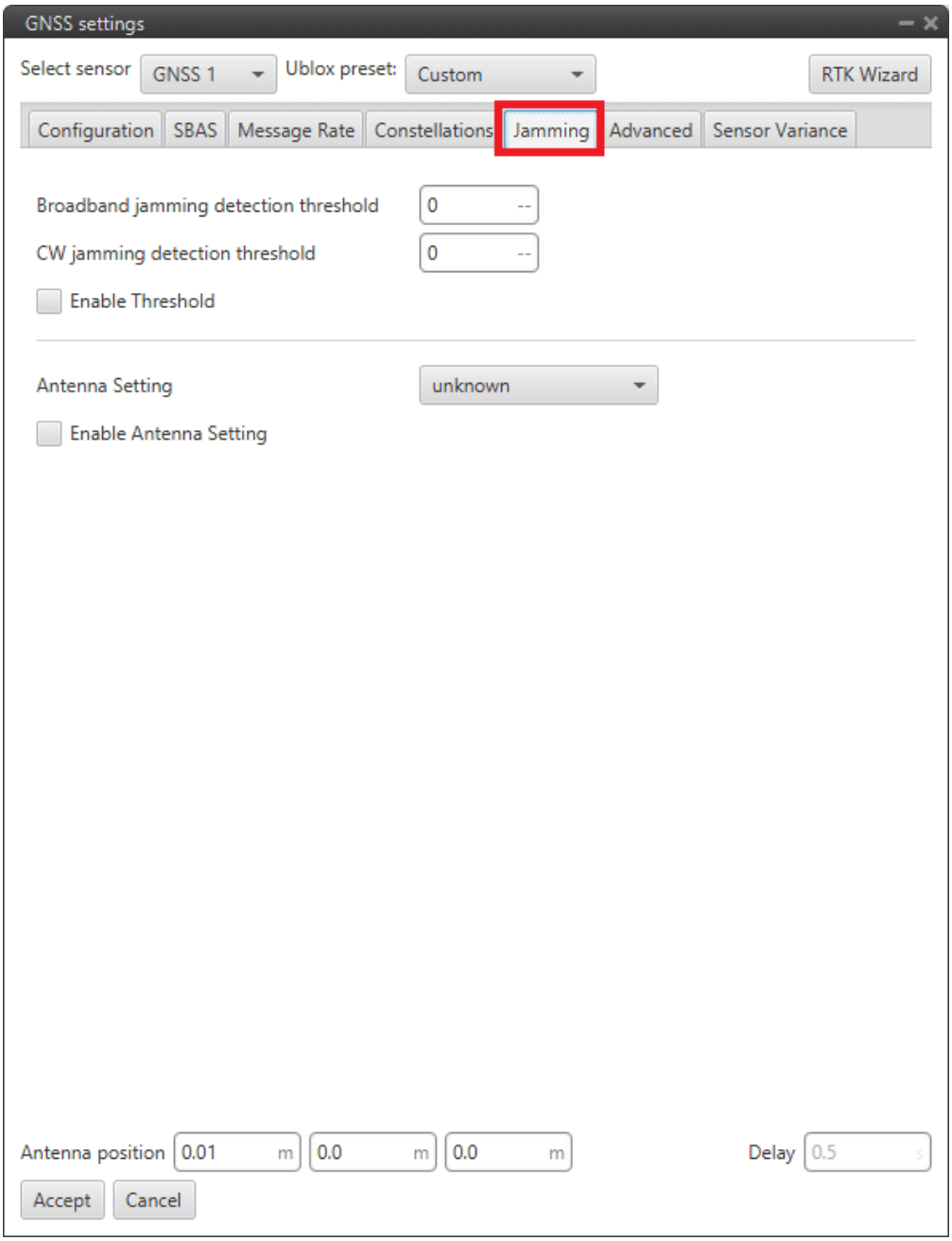


Fig. 413: **GNSS 1/2 sensor block configuration - Jamming tab**

* **Enable Threshold**: Enables the interference detection. Therefore, if broadcast or CW jamming is reported, Veronte Autipilot will disregard the GPS information, position not fixed.
* **Broadband jamming detection threshold**: If the value rises significantly above this threshold, this indicates that a broadband jammer is present.
* **CW jamming detection threshold**: If the value rises significantly above this threshold, this indicates that a continuous wave (CW) jammer is present.
* **Enable Antenna Setting**.

* **Antenna Setting**: It is also possible to specify whether the receiver expects an *active* or a *passive* antenna; *unknown* option if the user does not know the behavior of the antenna.

– **Advanced**: **The values shown here should only be modified by advanced users**. For this reason, the following message appears when entering this tab:

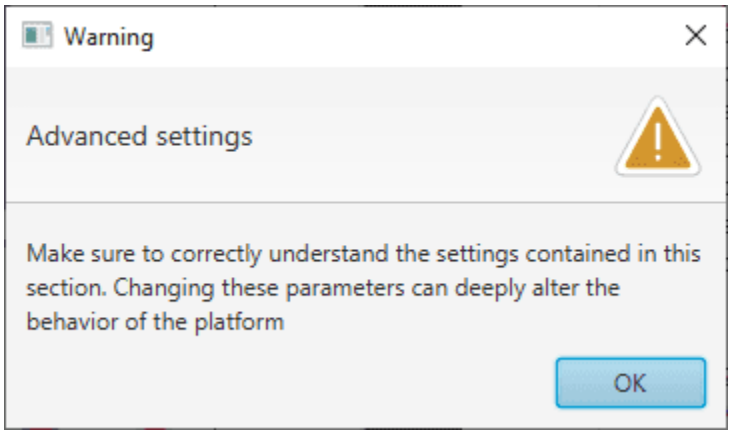


Fig. 414: **GNSS 1/2 sensor block configuration - Warning advanced tab**

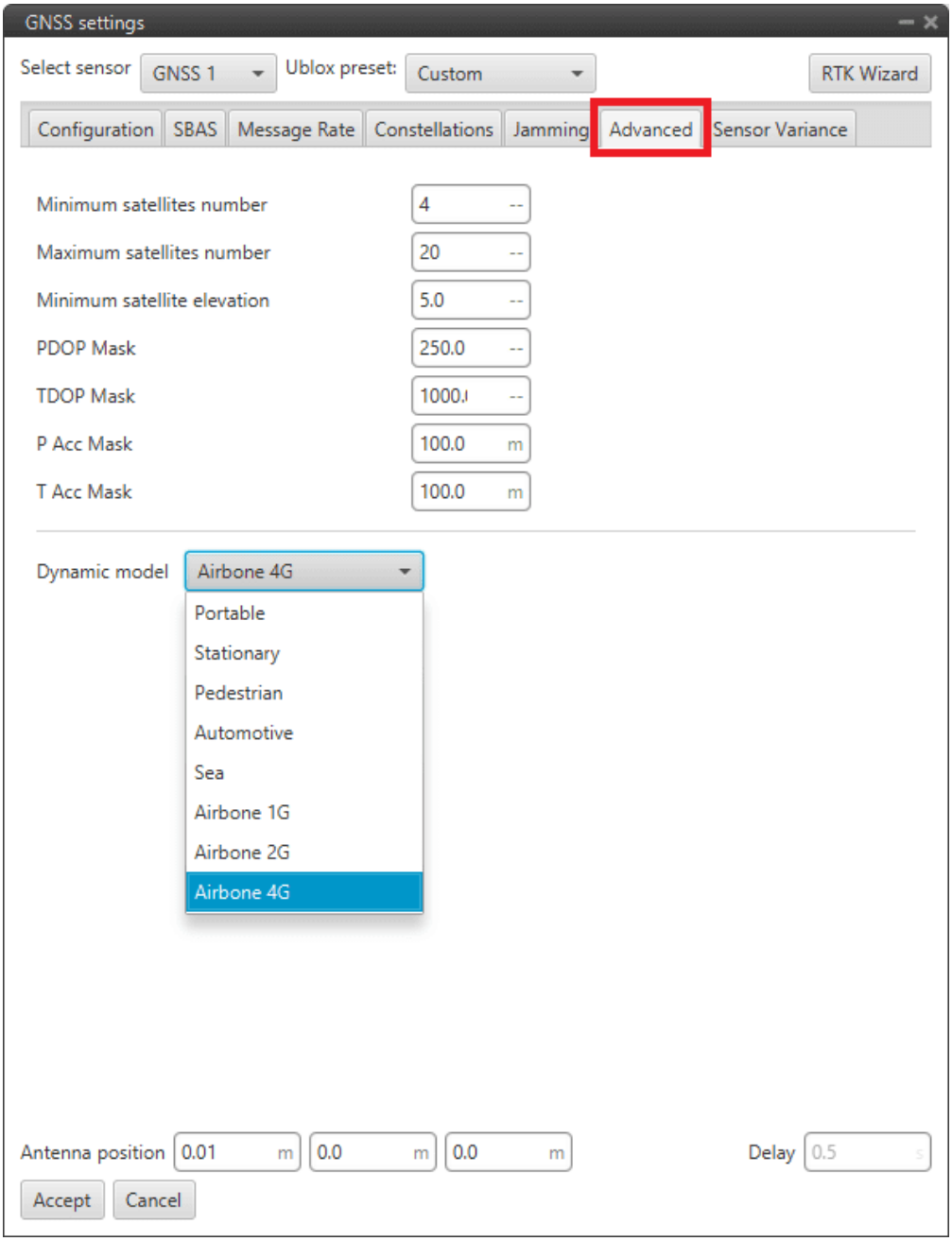


Fig. 415: **GNSS 1/2 sensor block configuration - Advanced tab**

Warning: Modifying these parameters can cause problems during the acquisition of GNSS positioning.

- **Minimum satellites number**: Minimum number of satellites needed to have position fixed.
- **Maximum satellites number**: Maximum number of satellites needed to have position fixed
- **Minimum satellite elevation**: Minimum elevation of a satellite to be considered. **Value in degrees**.
- **PDOP mask**: Maximum *Position Dilution of Precision* to consider the solution.

* **TDOP mask**: Maximum *Time Dilution of Precision* to consider the solution.
* **P Acc mask**: Maximum *Position Accuracy* to consider the solution.
* **T Acc mask**: Maximum *Time Accuracy* to consider the solution.
* **Dynamic model**

  The embedded receiver supports different dynamic platform models to adjust the GNSS navigation engine to the expected application environment. The settings improve the receiver's interpretation of the measurements and thus provide a more accurate position output. **Setting the receiver to an unsuitable platform model for the given application environment is likely to result in a loss of receiver performance and position accuracy**.

Platform	Description
Portable	Applications with low acceleration.
Stationary	Stationary applications. Velocity restricted to 0 m/s. Zero dynamics assumed.
Pedestrian	Applications with low acceleration and speed. Low acceleration assumed.
Automotive	Used for applications with equivalent dynamics to those of a car. Low vertical acceleration assumed.
Sea	Recommended for applications at sea, with zero vertical velocity. Zero vertical velocity assumed. Sea level assumed.
Airborne 1G	Used for applications with a higher dynamic range and greater vertical acceleration than a car.
Airborne 2G	Recommended for typical airborne environments.
Airborne 4G	Recommended for extremely dynamic environments.

– **Sensor Variance**: The variances considered in the EKF for the GNSS solution are by default the values provided by the GNSS receiver but can be modified for more complex scenarios.

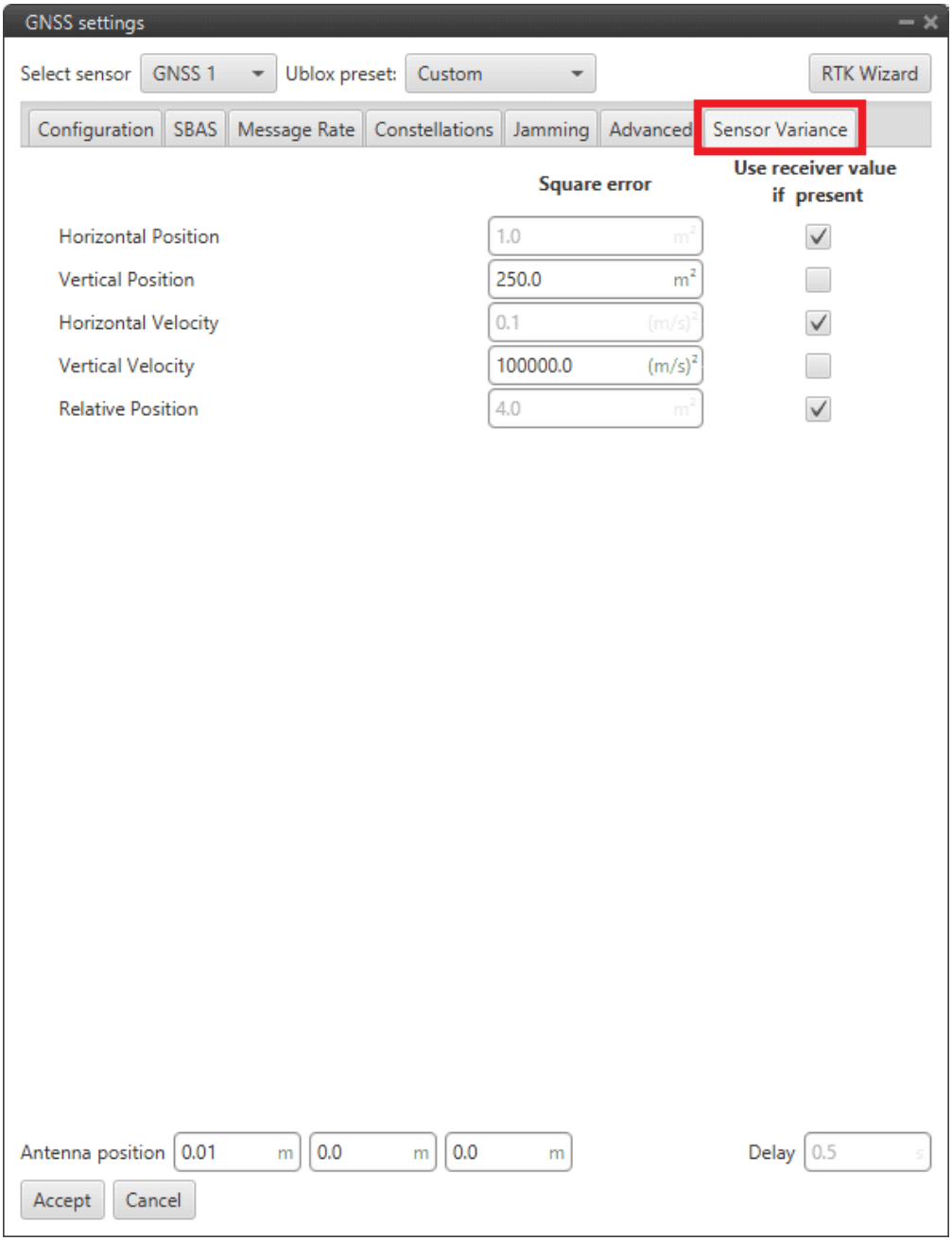


Fig. 416: **GNSS sensor block configuration - Sensor variance tab**

- **Horizontal Position**: Variance for the North and East components of the position solution.
- **Vertical Position**: Variance for the Down component of the position solution.
- **Horizontal Velocity**: Variance for the North and East components of the velocity solution.
- **Vertical Velocity**: Variance for the Down component of the velocity solution.
- **Relative Position**: This is the variance of the relative position from one GNSS receiver to another.

- **GPS External configuration menu**: If the GNSS information is received via an external system, the user must

configure it in this menu, so that this system can be included in the navigation filters.

After correctly configuring the communication protocol in the corresponding channel (RS232, RS485, CAN, ...) the GPS External variables of interest must be filled in this interface:

– **Configuration**:

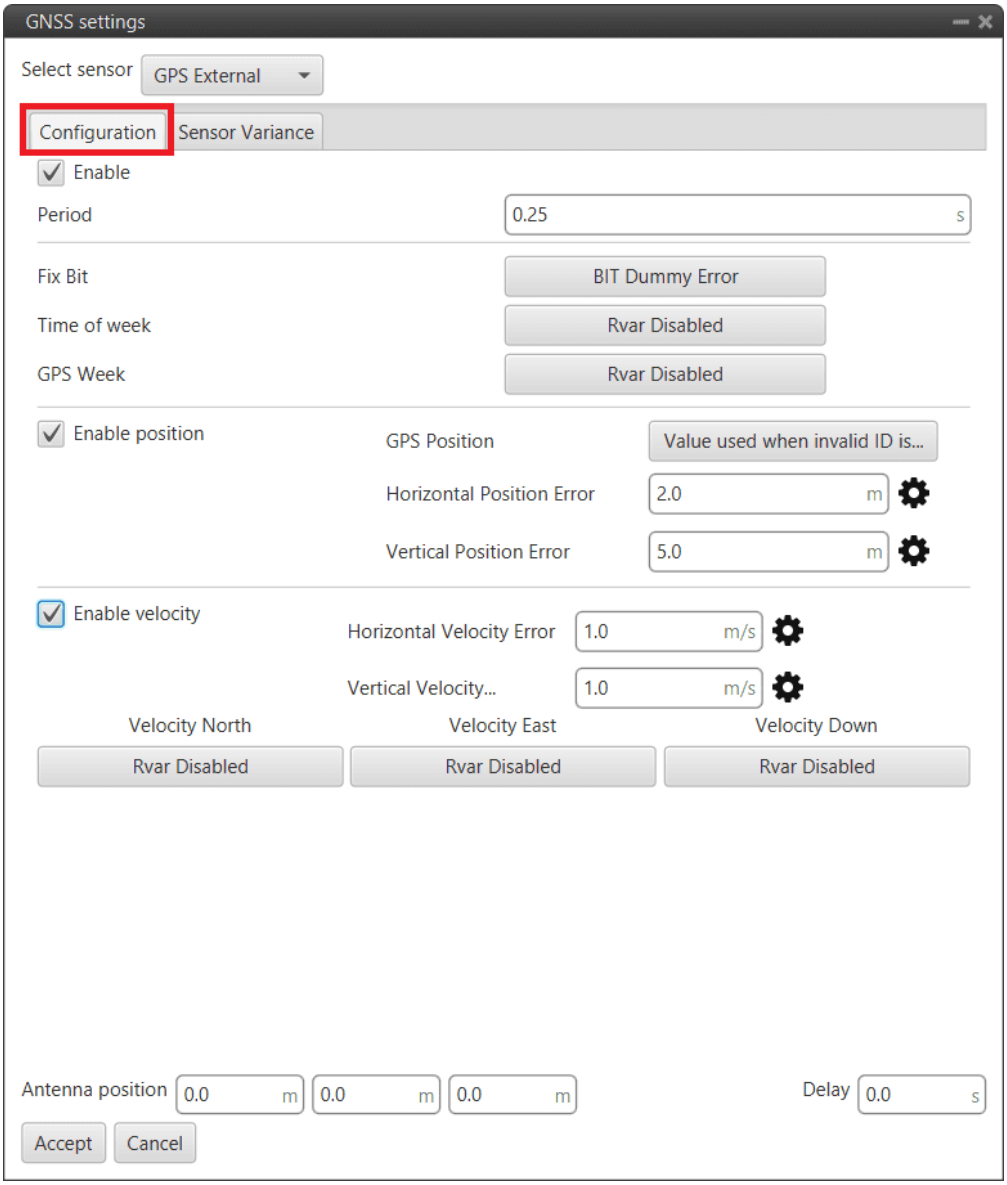


Fig. 417: **GPS External sensor block configuration - Configuration tab**

Caution: Check GNSS External device communication protocol before filling this menu.

The user must create a **Custom Message** according to the communication protocol used by the external sensor, so that its readings are stored in system variables. Then, the user can select these variables to configure the following parameters:

    - **Enable**.
    - **Period**: Defines the period of incoming information from the external system.
    - **Fix Bit**: Data provided by the external device which is important to know the status of the positioning.
    - **Time of week**: Variable extracted from the communication protocol defining the time of the week.
    - **GPS Week**: Variable extracted from the communication protocol defining the week.
    - **Enable position**:
        - **GPS Position**: Variable defining latitude, longitude and height from GNSS. Usually Moving Object variables are used in **1x PDI Builder**.
        - **Horizontal Position Error**: Defined by the GNSS External device provider.
        - **Vertical Position Error**: Defined by the GNSS External device provider.
    - **Enable Velocity**:
        - **Horizontal Velocity Error**: Defined by the GNSS External device provider.
        - **Vertical Velocity Error**: Defined by the GNSS External device provider.
        - **Velocity North/East/Down**: Variables extracted from the communication protocol defining GNSS velocity measured.

- **Sensor Variance**: This tab is configured in the same way as described above.

2.9.12.3 Magnetic Field

Magnetic Field sensor block returns the configured magnetic field in the current location.

Note: The magnetic field configuration is global, shared by all blocks of this type.

Fig. 418: **Magnetic Field block**

- **Output**

  Pin 0: Magnetic field in **NED frame** as a **5-dimensional real array** with the following components:

  - 0: Update flag (always 1)
  - 1: North component of magnetic field
  - 2: East component of magnetic field
  - 3: Down component of magnetic field
  - 4: Variance (always zero)

2.9.12.4 Magnetometer

Magnetometer sensor block returns the magnetic field being read by the selected sensor in the body frame.

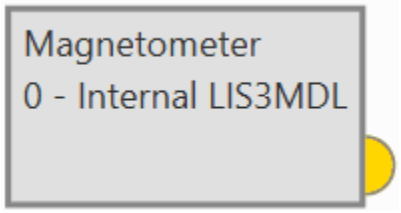


Fig. 419: **Magnetometer block**

- **Output**

  Pin 0: Magnetic field in **body frame** as a **5-dimensional real array** with the following components:

  - 0: Update flag (always 1)
  - 1: X body component of magnetic field
  - 2: Y body component of magnetic field
  - 3: Z body component of magnetic field
  - 4: Variance

- **Configuration menu**:

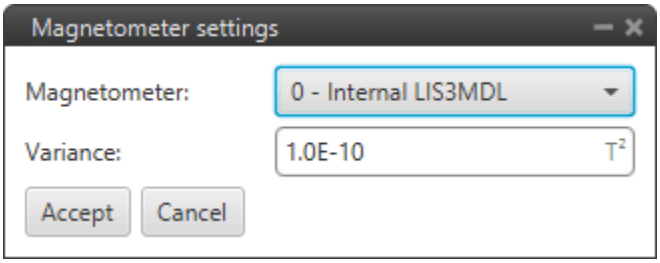


Fig. 420: **Magnetometer block configuration**

  – **Magnetometer**: Users must select the desired internal or external sensor magnetometer to be used.

  – **Variance**: Means the influence of the parameter on the Navigation filters. The higher the variance, the lower the effect. This is the only parameter configured indepently from the Magnetometer selected.

2.9.12.5 Relative position

Relative position sensor block works in conjunction with the *Internest configuration panel* (of the **Sensors menu**) to configure the **Internest system** as an ultrasound sensor that calculates the position of Veronte Autopilot 1x.

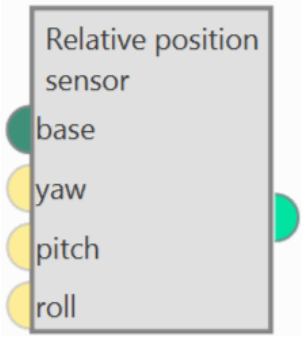


Fig. 421: **Relative position block**

- **Inputs**: The 3 navigation angles (*yaw*, *pitch* and *roll*) of the **base** platform can be entered.
  - **base**: Base position to which position measurements are relative. Usually a 'Moving object' is linked.
  - (Optional) **yaw**: Yaw of system of reference in which position measurements are received (0 if not connected).
  - (Optional) **pitch**: Pitch of system of reference in which position measurements are received (0 if not connected).
  - (Optional) **roll**: Roll of system of reference in which position measurements are received (0 if not connected).
- **Output**
  - **Pin 0**: Absolute position measurement.
- **Configuration menu**:

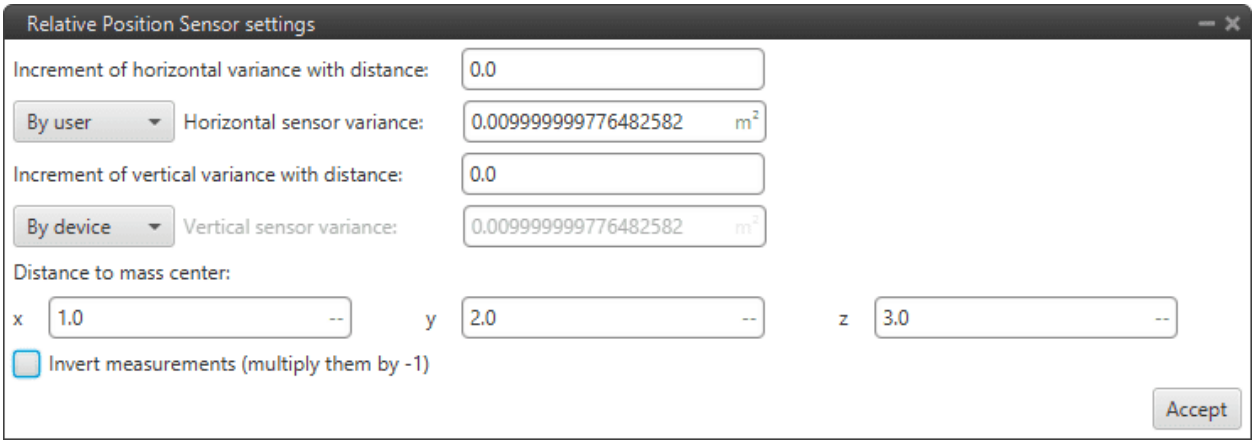


Fig. 422: **Relative position block configuration**

  - **Increment of horizontal/vertical variance with distance**: With this increment, the further away the Veronte Autopilot 1x is from the base, the more variance it is given in a linear fashion.
  - **Horizontal/Vertical sensor variance**: Square error of the internest position in xy/z planes.

> **Note:** If the option *By device* is selected, these parameters are automatically set by the autopilot.

  - **Distance to mass center: x/y/z**: Defines the distance between the Internest system and the center of mass from the *base*.

– **Invert measurements (multiply them by -1)**: If enabled, the measurements shall be multiplied by -1.

More information

These parameters are used in the calculation of the variance for the EKF algorithm by means of the following equation:

$$square\ error = position\ error + (increment\ error \cdot distance)$$

– $position\ error$: Horizontal/Vertical sensor variance position

– $increment\ error$: Increment of hotizontal/vertical variance with distance

– $distance$: Distance to the *base*

2.9.12.6 SRTM height

SRTM height sensor block gets the terrain altitude at the current UAV position according to the configured mesh.

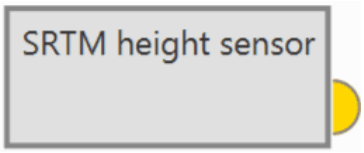


Fig. 423: **SRTM height block**

- **Output**

  Pin 0: Terrain altitude as a **4-dimensional real array** with the following components:

  - 0: Update flag (always 1)

  - 1: Valid flag (inside mesh)

  - 2: Terrain height

  - 3: Variance

- **Configuration menu**:

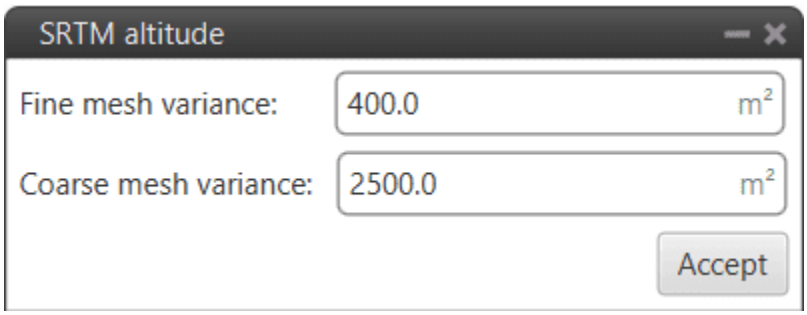


Fig. 424: **SRTM height block configuration**

– **Fine mesh variance**: Variance of the fine mesh. This is the smallest mesh, which contains detailed information on the altitude of the terrain.

– **Coarse mesh variance**: Variance of the coarse mesh. This is the medium mesh with the least detail.

Important:

– **The values to be entered** in the configuration of this block must be > 0.

– If the configured error values are very large, the EKF will converge more slowly to them or give more importance to other sensors, such as Lidar, to know the height of the terrain.

– If these values are small, much more importance will be given to the terrain grid and it will converge faster.

2.9.12.7 Static Pressure

Static Pressure sensor block returns the static pressure measured by the selected sensor.

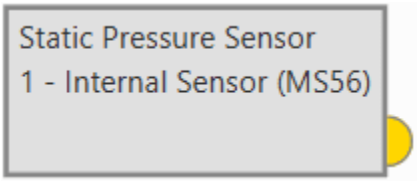


Fig. 425: **Static Pressure block**

- **Output**

  Pin 0: Static pressure measurement as a 3-dimensional real array with the following components:

  - 0: Update flag

  - 1: Pressure

  - 2: Variance

- **Configuration menu**:

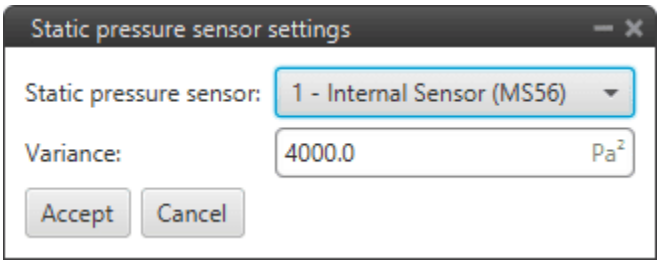


Fig. 426: **Static Pressure block configuration**

– **Static pressure sensor**: Users must select the desired static pressure sensor to be used.

– **Variance**: Means the influence of the parameter on the Navigation filters. The higher the variance, the lower the effect. This is a parameter configured independently from the Static Pressure sensor selected.

2.9.13 Servos blocks

2.9.13.1 Actuator

Actuator block controls the transformation of the action to the servo value.

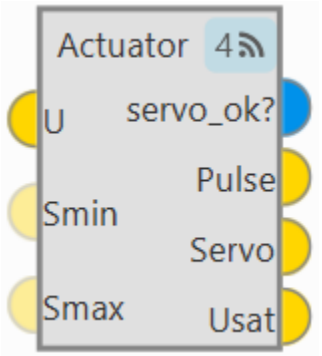


Fig. 427: **Actuator block**

- **Inputs**
  - **U**: Control actions (U) before servo saturation.
  - (Optional) **Smin**: Vector of minimum values allowed for the servos.
  - (Optional) **Smax**: Vector of maximum values allowed for the servos.
- **Outputs**
  - **servo_ok?**: Output BIT vector that indicates the servos that had to be trimmed to prevent saturation.
  - **Pulse**: PWM pulse for servos.
  - **Servo**: Servo value.
  - **Usat**: Control actions (U) after servo saturation.
- **Configuration menu**:

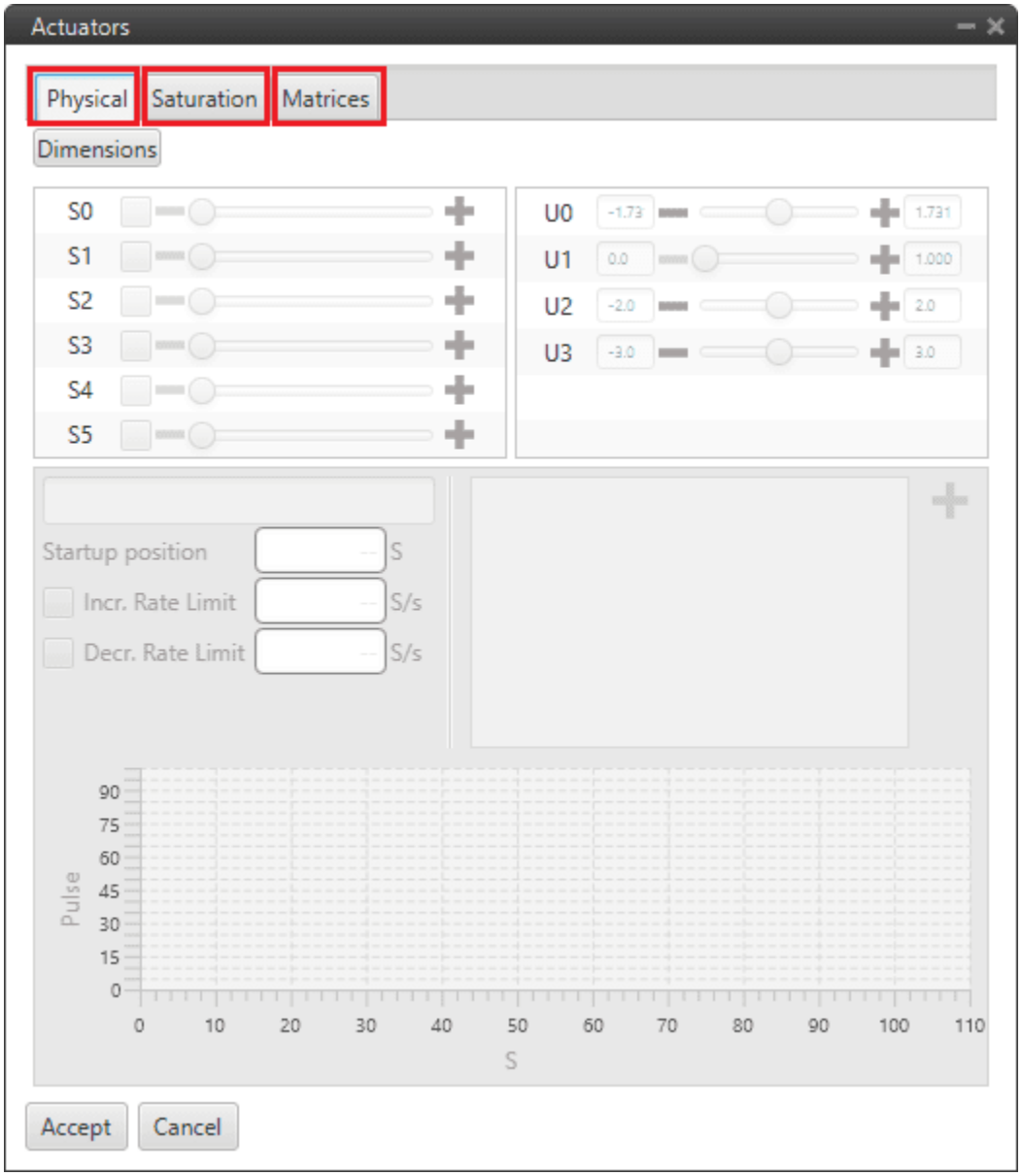


Fig. 428: **Actuator block configuration**

The servo configuration is divided into 3 different menus: **Physical**, **Saturation** and **Matrices**.

– **Physical**: This tab allows the actuators physical configuration.

> **Warning:** The calibration of all connected actuators is performed in the 1x PDI Calibration software.

Fig. 429: **Actuator block configuration - Physical**

1. **Dimensions**: Set the number of servos and control outputs.

Fig. 430: **Actuator block configuration - Physical dimensions**

Note: Veronte Autopilot 1x allows up to 32 actuators to be configured at the same time.

2. **Servos (actuators)**: This menu contains the servos of the platform.

   Warning: The modification of this menu is not available from this block, as mentioned above, this calibration is carried out in the 1x PDI Calibration software.

3. **Control Signals**: This menu contains the variables representing the control signals/outputs U generated by the system. The mapping of the controls to servo positions is indicated within the **SU matrix**, which is set in the **Matrices tab**.

   Warning: The modification of this menu is not available from this block, as mentioned above, this calibration is carried out in the 1x PDI Calibration software.

4. **Servo parameters**:
   - **Actuator Output variable**: If the *Actuator output* variable has been renamed, it will be renamed here as well.
   - **Startup position** *(S)*: Sets the initial values of the actuators.
   - **Increasing/Decreasing Rate Limit** *(S/s)*: Sets a rate limit for increasing/decreasing motions of the servo.
5. **Servo Position - PWM**: This option is used to set the mapping of the S servo position to the PWM signal. In this example, *1* S position corresponds to a 100 % pulse to be sent to the corresponding servo (*Motor 1*).

   The mapping is expressed through the **graph**, where the user can enter as many points as desired.

– **Saturation**: In this menu, the user can configure the behavior of the platform when one or more of its actuators is/are in saturation state.

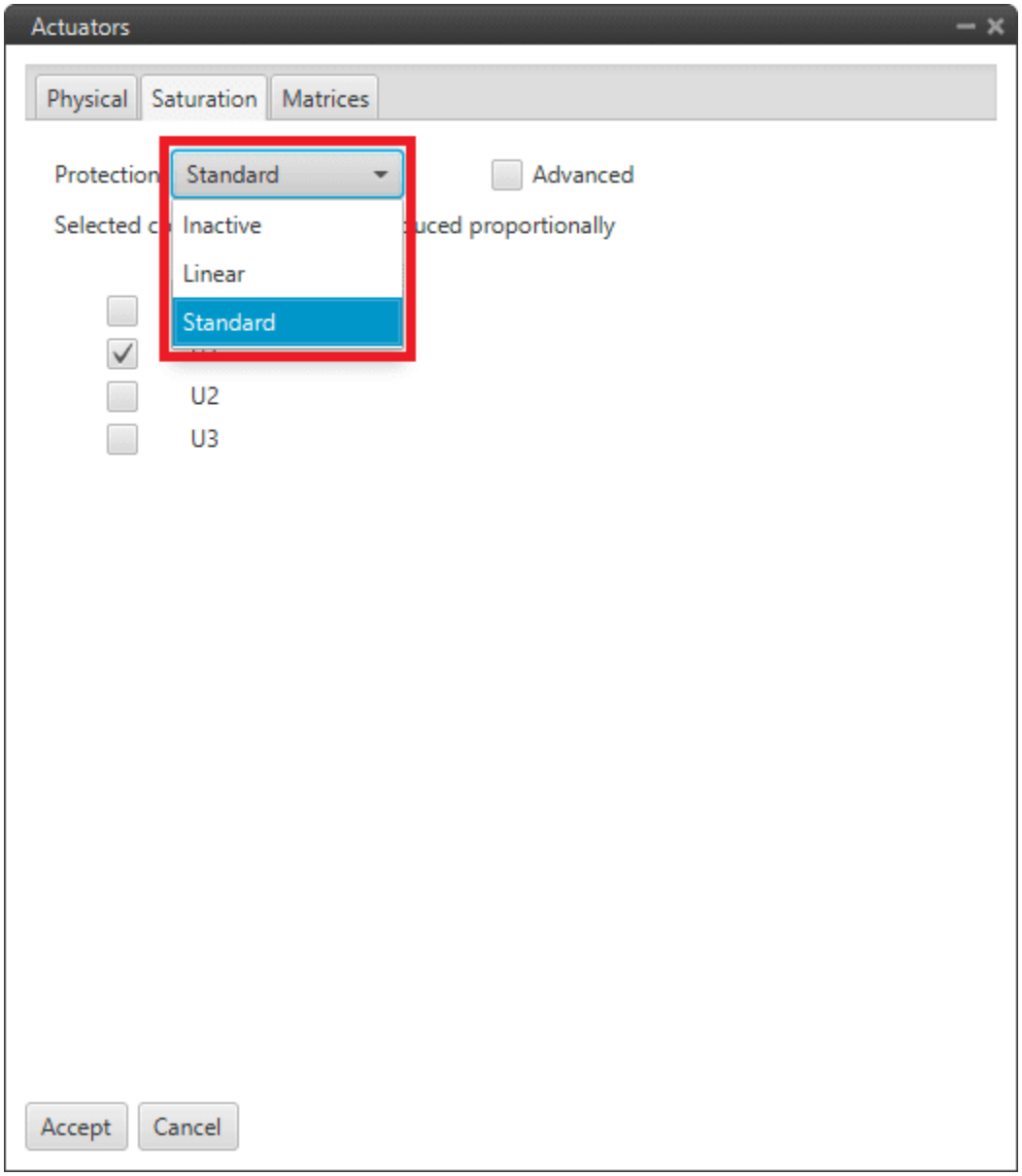


Fig. 431: **Actuator block configuration - Saturation**

The three available options are:

* **Inactive**: The system does not respond to saturation.
* **Linear**: The system affects all the actuators in the same way if saturation is reached.
* **Standard**: The system affects only the selected actuators if saturation is reached at any actuator. It can be chosen from 1 to all of them (which will be linear action).

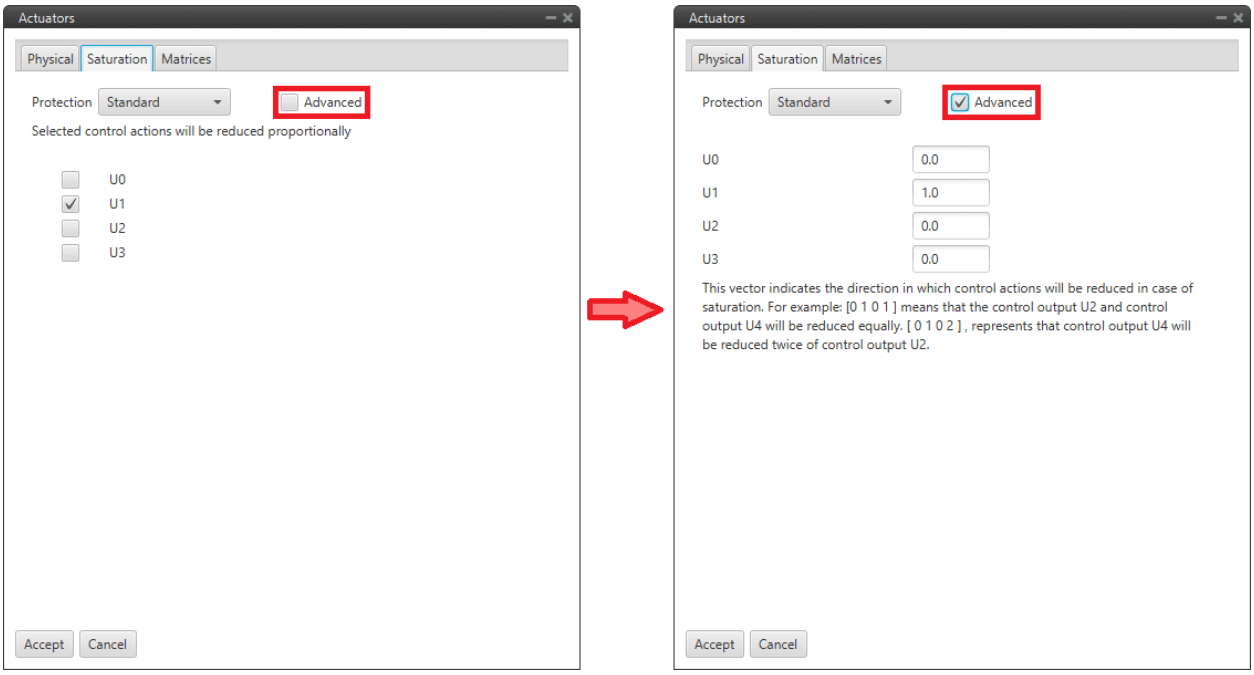


Fig. 432: **Actuator block configuration - Saturation Standard**

Clicking the **Advanced** checkbox generates a vector that includes all control outputs, allowing proportional control over the system when saturation occurs.

This tool is configured to allow the user to have more extensive control over this feature if required.

– **Matrices**:

SU and US are 2 matrices (inverse of one another, respectively) which contain the relationship between actuator outputs S and control outputs U, i.e. the influence of each control channel on each actuator output. The option of having a configurable SU matrix allows Veronte Autopilots 1x to control any type of vehicle, independently of how its control surfaces/devices are set and adjusted.

U is a vector which contains the control outputs of the platform, e.g. **pitch**, **roll**, **yaw**, **throttle**, etc. The values of U do not represent a physical variable. They are instead fictitious variables which are used in the control algorithm. What is actually applied to the system are the actuators movements, i.e. the PWM signals sent to the servos, which are mapped in the S vector.

The relation between S and U is essential for the right attitude control of the platform.

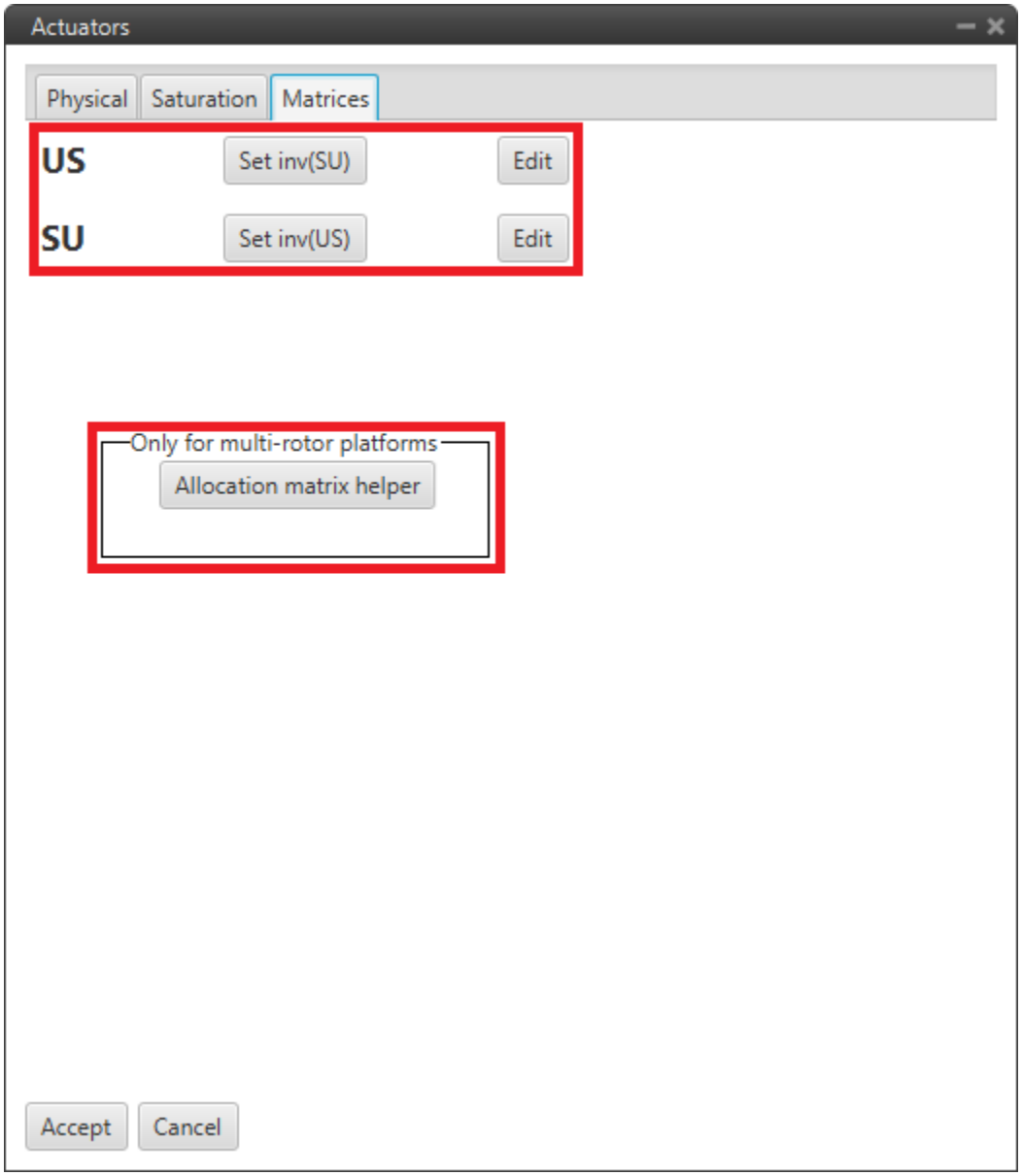


Fig. 433: **Actuator block configuration - Matrices**

* Normally, the **SU matrix is defined** instead of the US matrix because it is **more intuitive**, the US is calculated automatically. To define it, click *Edit* and the following pop-up window will open with the matrix. Control outputs U are placed on the columns and actutator outputs S on the rows.

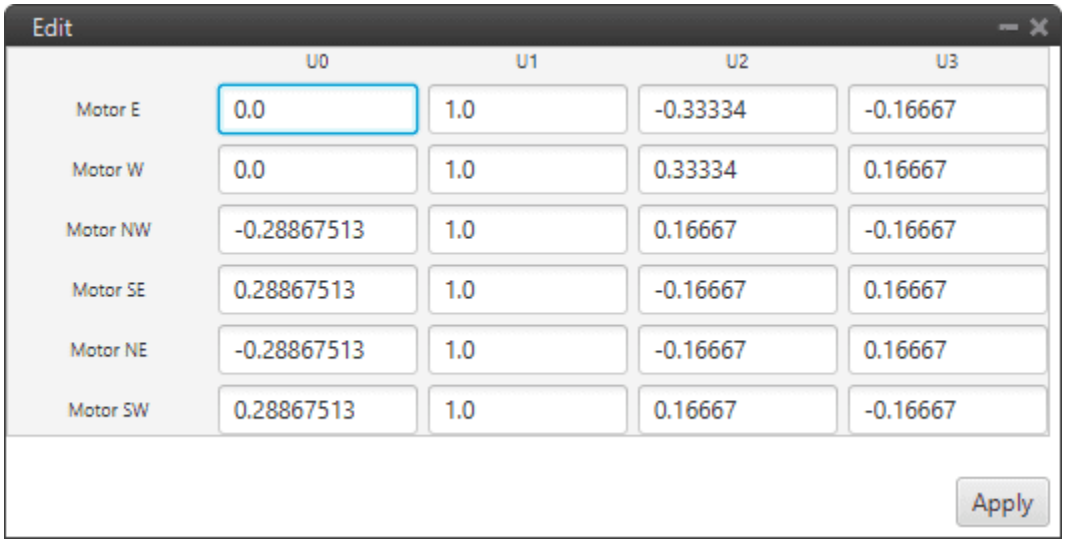


Fig. 434: **Actuator block configuration - SU matrix**

* In addition, an allocation matrix is available to help the user configure these matrices for a **multi-rotor**.

Warning: Regarding the selection of the parameters of SU matrix, the order of magnitude of the parameters should be respected at least for every row, i.e. every control channel, as long as there are no coupled control channels U.

Good practice recommendations

* Unitary values are recommended. Doing so, U will be equal to S. And if S has been defined according to a physical value – e.g. deflected angle, then control outputs can be easier to understand.
* The order of magnitude and the value of the SU parameters will not influence control algorithm calculations. But it will affect the control parameters, i.e. the control gains.
* It is recommended to keep the same order of magnitude for the whole matrix. That will allow an easier set up of a scaled version of the platform. Keeping the same SU and knowing the scaling factor, then the new control gains should be the old ones multiplied by that scaling factor. This practice can also be useful for transition to similar platforms.
* The SU matrix and S vector should be defined accordingly in order to follow the sign convention for aerial navigation, a positive roll lowers the right wing, a positive pitch moves the nose up and a positive yaw moves the nose the right.

An example of the use of this block is given below:

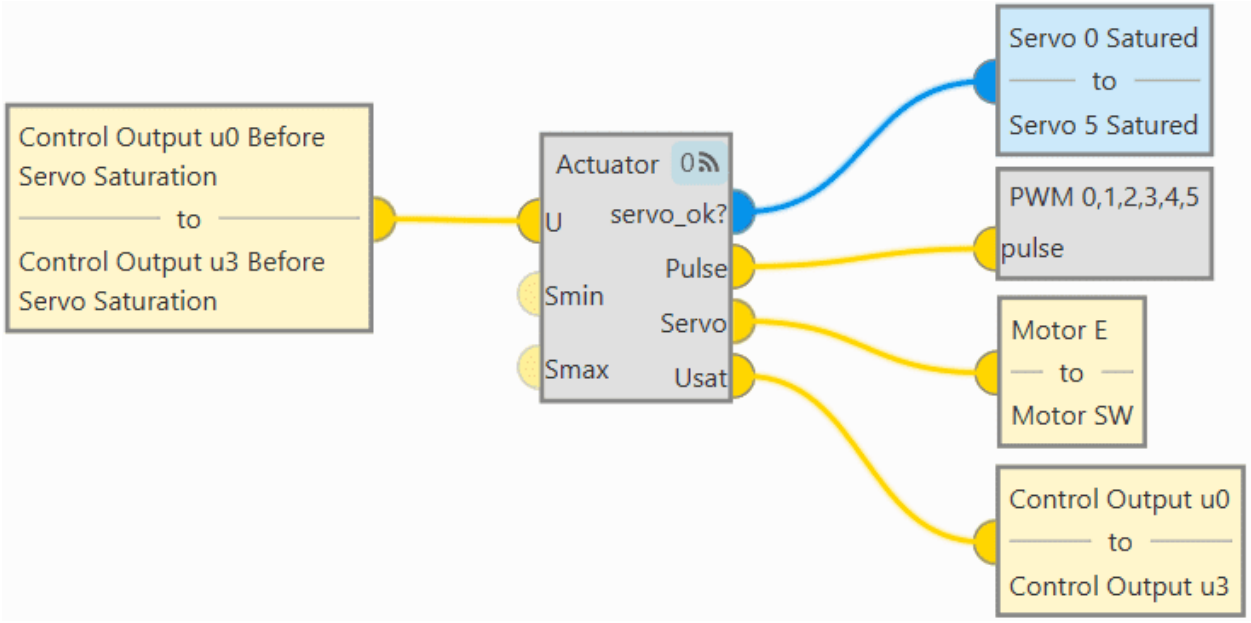


Fig. 435: **Actuator block - Example of use**

2.9.13.2 Arc Trim

Arcade stick trim block is used to set the zero-stick position for the *Arcade Mode*.

Fig. 436: **Arc Trim block**

The *Stick input* variable that enters the navigation algorithm is called '*Stick input d*', which is the one obtained from the **Arc Trim block**.

It is calculated as $D = U - U_0$, where U_0 is the arcade trim.

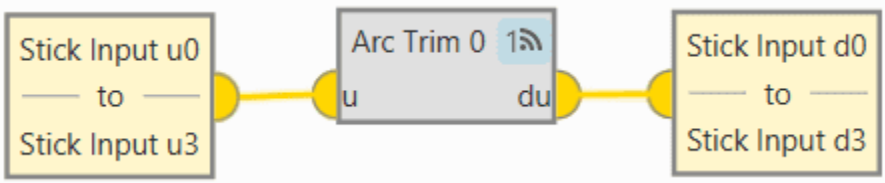


Fig. 437: **Arc Trim block - Example of use**

- **Input**
  - **u**: Input stick vector to trim.
- **Output**

- **du**: Trimmed stick vector.

- **Configuration menu**:

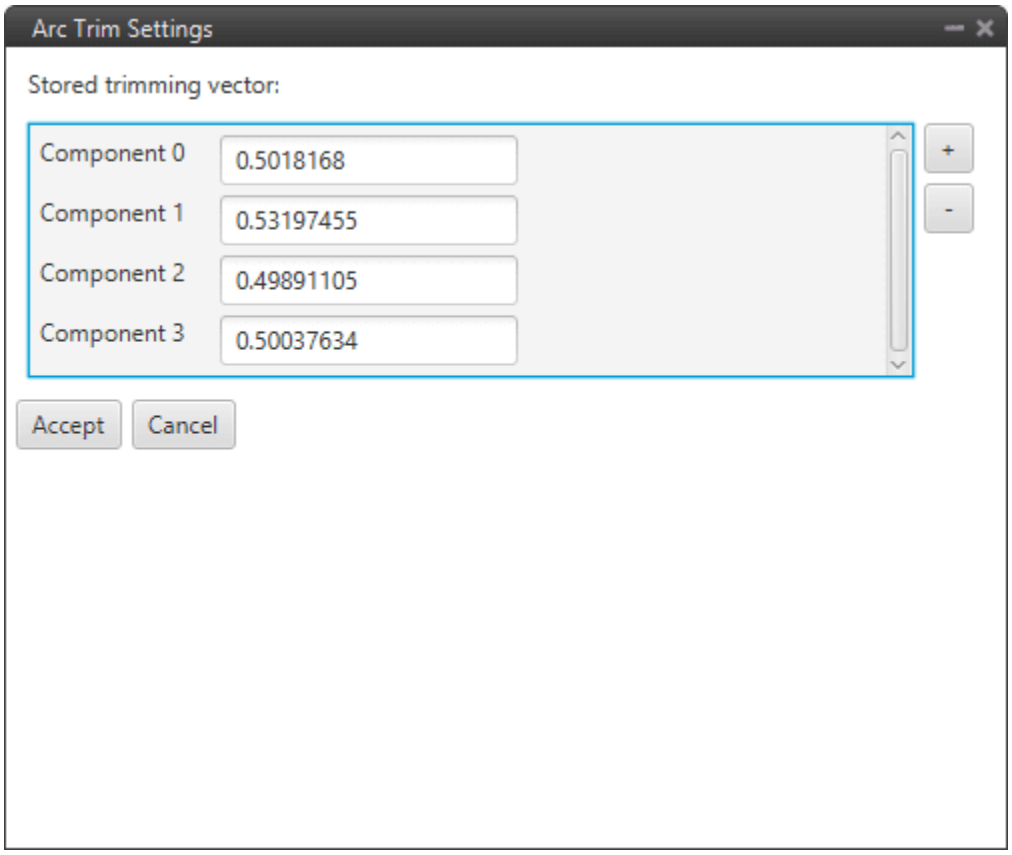


Fig. 438: **Arc Trim block configuration**

The values of the trim vector U_0 can be **entered manually** in the configuration menu (as shown in the figure above) or by **creating an automation** that autocompletes these values with the stick position. For the latter case, the configuration menu should be with all trim values to **0**.

For more information on this automation, see the *ArcTrim (Command block) action - Automations* section of this manual.

Warning: The Arcade mode has to be trimmed before flight. If not trimmed, the zero level will be different from the desired one.

2.9.13.3 PWM

PWM block applies the input vector to the configured PWM outputs.

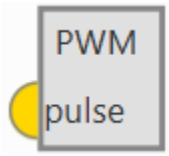


Fig. 439: **PWM block**

- **Input**

  pulse: Input vector of pulses to apply.

- **Configuration menu**:

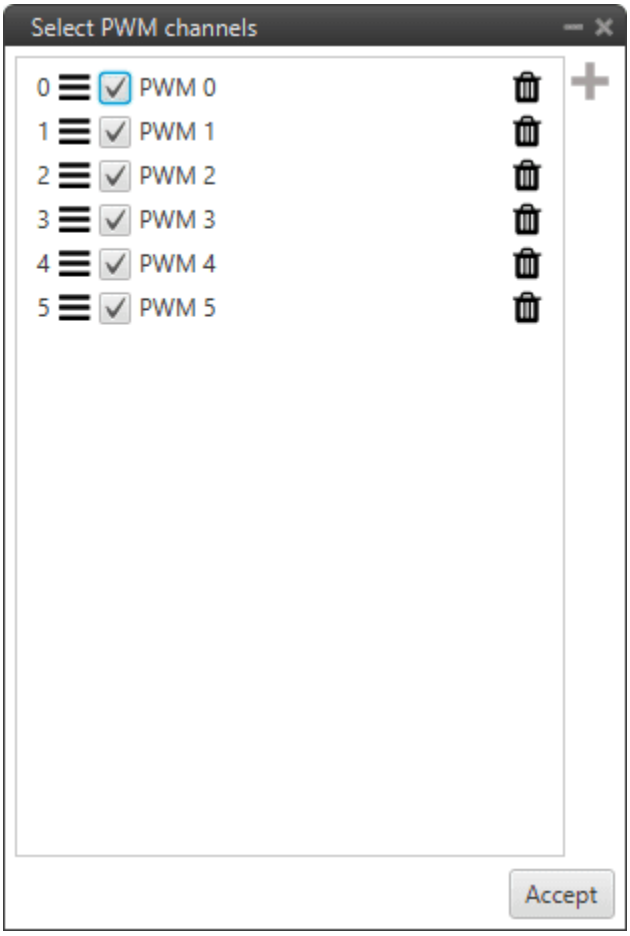


Fig. 440: **PWM block configuration**

  - : Users must enter the PWM variables to be configured.
  - : PWMs variables can be sorted as desired by simply dragging and dropping them.
  - Check the variable to enable commands to this PWM.

- : Deletes the PWM.

A PWM servo configuration can be found in the *PWM (Servos) - Integration examples* section of this manual.

2.9.14 Signals blocks

Signal blocks include functions for processing and filter signals, control inputs and outputs, etc.

2.9.14.1 3D Table Interpolation

3D Table Interpolation block returns the value obtained interpolating the configured table with the input variables.

Fig. 441: **3D Table Interpolation block**

- **Inputs**
  - **Pin 0**: X component, **columns**.
  - **Pin 1**: Y component, **rows**.
- **Output**
  - **Pin 0**: Value interpolated from table for the input X and Y components.
- **Configuration menu**:

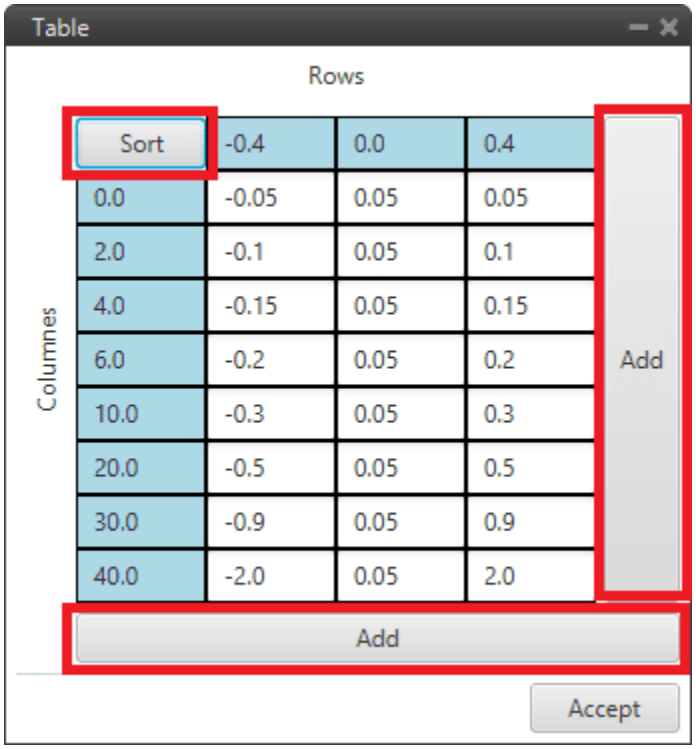


Fig. 442: **3D Table Interpolation block configuration**

- **Sort**: By pressing this button, rows and columns are sorted from lowest to highest.
- **Add**: A row/column is added

Note: If **out of range**, the value for the closest limit shall be taken.

2.9.14.2 Acceleration limiter

Acceleration limiter block is a signal first and second derivative limiter.

It is like the *Rate limiter block* but "improved", as a less aggressive response is obtained, approaching the desired input in a "softer" way.

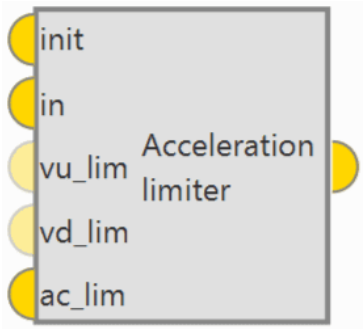


Fig. 443: **Acceleration limiter block**

- **Inputs**
  - **init**: Initialization value, this is the output of the block in the first step after on_focus.
  - **in**: Input signal.
  - (Optional) **vu_lim**: **First derivative limit** in the **up direction**. The value is read as **absolute value**, this means that **the sign** of this input is **neglected**.
  - (Optional) **vd_lim**: **First derivative limit** in the **down direction**. The value is read as **absolute value**, this means that **the sign** of this input is **neglected**.
  - **ac_lim**: **Second derivative limit**. The value is read as **absolute value**, this means that **the sign** of this input is **neglected**.
- **Output**
  - **Pin 0**: Limited signal.

2.9.14.3 Bound

Bound block limits the input signal and produces a bit to indicate if it was within the allowed range.

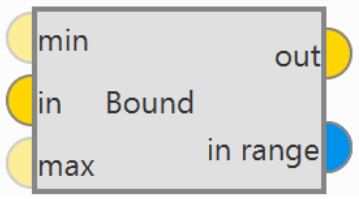


Fig. 444: **Bound block**

- **Inputs**
  - (Optional) **min**: Minimum value allowed for input signal. If **not defined**, it is assumed to be **infinity**.
  - **in**: Input signal.
  - (Optional) **max**: Maximum allowed value for input signal. If **not defined**, it is assumed to be **infinity**.
- **Outputs**
  - **out**: Limited signal.
  - **in range**: Bit that is true when the input signal is within the allowed range and false otherwise.

User can use the **Bound** block to monitor critical system parameters that are within operating limits, e.g. airspeed. If **not OK**, **in range** can be used to trigger an alarm.

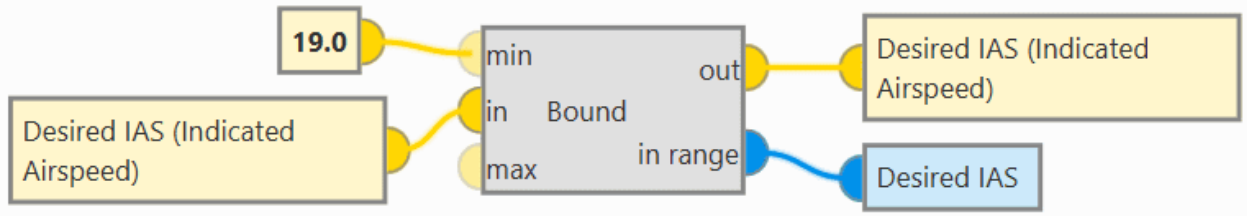


Fig. 445: **Bound block - Example of use**

2.9.14.4 Derivative

Derivative block is a numerical derivative of an input signal.

Note: As the result of a numerical derivative can be noisy, this block includes a simple first order filter with configurable time constant tau to smooth the output.

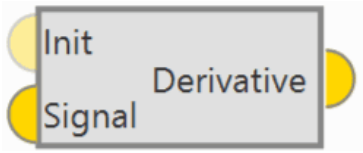


Fig. 446: **Derivative block**

- **Inputs**
  - (Optional) **Init**: Optional **initial value of the derivative**, if not connected a value of zero is assumed.
  - **Signal**: Signal to compute the numerical derivative.
- **Output**
  - **Pin 0**: Derived signal.
- **Configuration menu**:

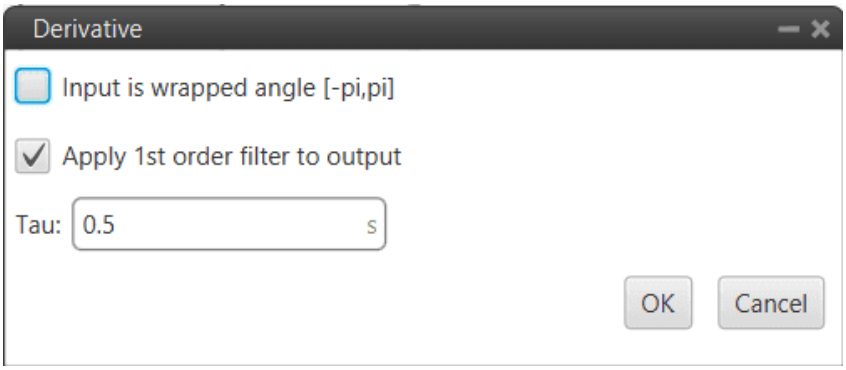


Fig. 447: **Derivative block configuration**

  - **Input is wrapped angle [-pi,pi]**: Perform a [-pi, pi] wrap. It should be enabled when using angles.
  - **Apply 1st order filter to output**: If enabled, a simple first order filter is applied to smooth the output. By default it is enabled and is also recommended.
  - **Tau**: The time constant *Tau* must be entered.

2.9.14.5 EWMA Tau filter

EWMA (Exponentially Weighted Moving Average) Tau filter block is a simple first order filter with configurable time constant *Tau*.

This filter follows the following equation:

$$y = \alpha \cdot u + (1 - \alpha) \cdot y_{-1}$$

Where:

- $\alpha = \frac{dt}{dt+\tau}$
  - dt: GNC Timestamp
  - τ: Time constant
- u: Input value to be filtered
- y_{-1}: Initialization value at the first execution of the block (t=0)

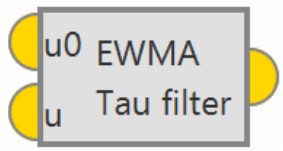


Fig. 448: **EWMA Tau filter block**

- **Inputs**
  - **u0**: Initialization value (set in on_focus).
  - **u**: Current value to filter.
- **Output**
  - **Pin 0**: Filtered value.
- **Configuration menu**:

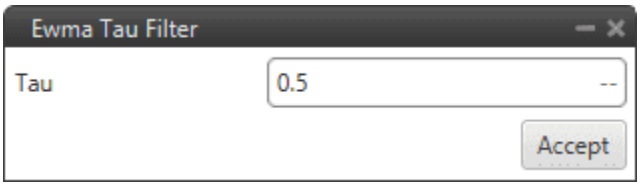


Fig. 449: **EWMA Tau filter block configuration**

  - **Tau**: The time constant *Tau* must be entered.

2.9.14.6 FFT

Error: The FFT block is temporarily disabled in this version.

FFT (Fast Fourier Transform) block outputs the Fast Fourier Transform of the input signal.

Fig. 450: **FFT block**

- **Input**
  - **in**: Input signal.
- **Outputs**
  - **Amax**: 3D vector containing the magnitude of the three dominant frequencies (sorted from higher to lower magnitude).
  - **Fmax**: 3D vector containing the frequency of the three dominant frequencies (sorted from higher to lower magnitude).
- **Configuration menu**:

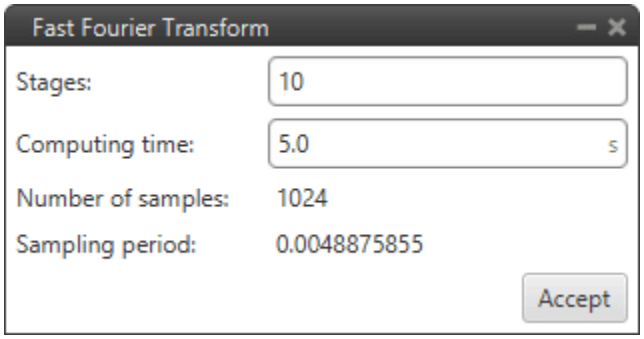


Fig. 451: **FFT block configuration**

  - **Stages**.
  - **Computing time**.

2.9.14.7 Hysteresis

Hysteresis block applies hysteresis to input signal to prevent changes in output signal when the input is close to zero.

Fig. 452: **Hysteresis block**

The behavior is as shown in the following diagram:

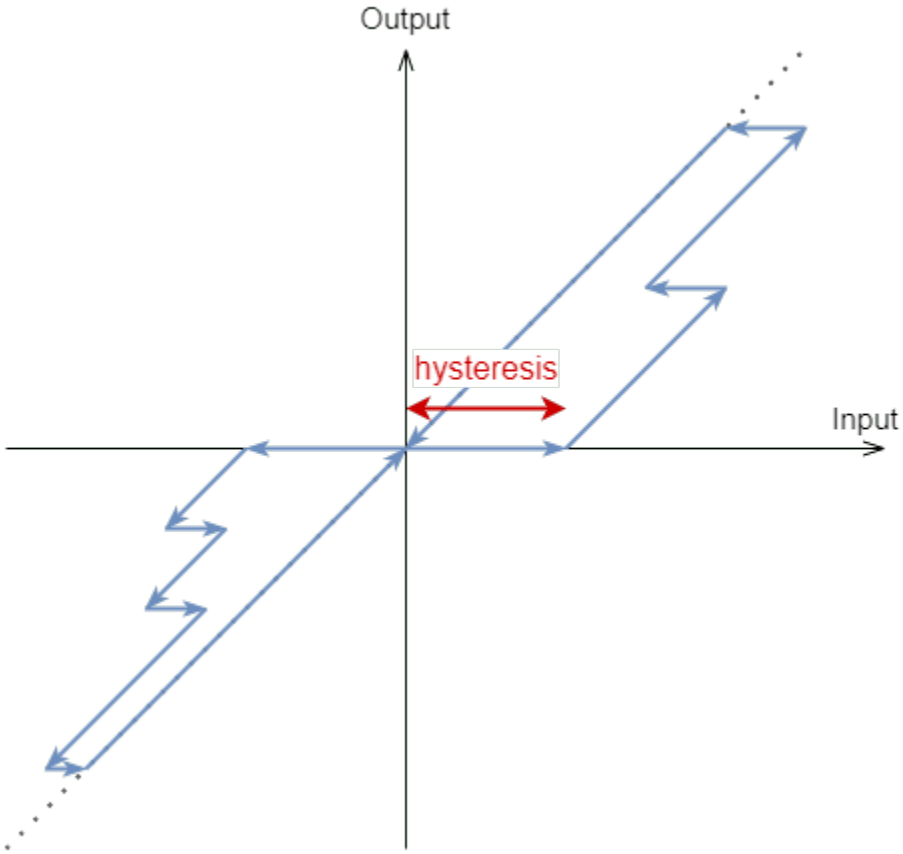


Fig. 453: **Hysteresis diagram**

- **Input**
  - **Pin 0**: Input signal.
- **Output**
  - **Pin 0**: Output signal.
- **Configuration menu**:

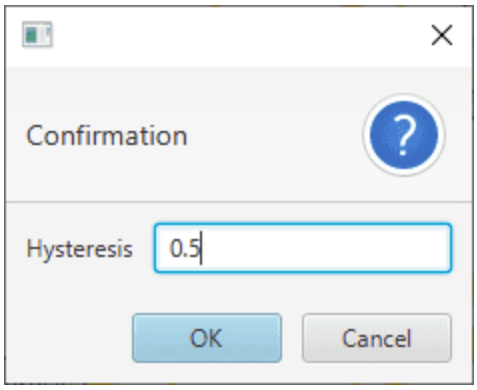


Fig. 454: **Hysteresis block configuration**

- **Hysteresis**: Users must enter the magnitud of the hysteresis.

2.9.14.8 IIR Filter

IIR Filter block allows the user to define an Infinite Input Response filter, it applies a **Z-transform**.

Fig. 455: **IIR Filter block**

- **Input**
  - **Pin 0**: Input signal.
- **Output**
  - **Pin 0**: Output signal.
- **Configuration menu**:

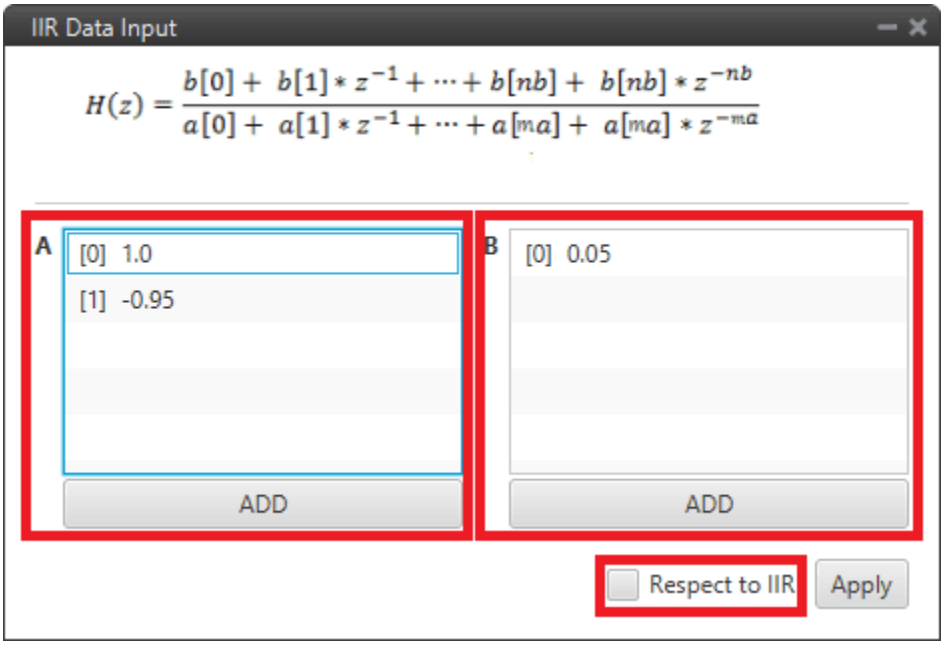


Fig. 456: **IIR Filter block configuration**

- **A**: Filter *a* coefficients. The user can add as many *coefficients a* as desired.
- **B**: Filter *b* coefficients. The user can add as many *coefficients b* as desired.
- **Respect to IIR**: If enabled, the first time the block is executed, it takes the value of input as the initial offset.

This block can be used as a derivative if configured as shown in the following figure:

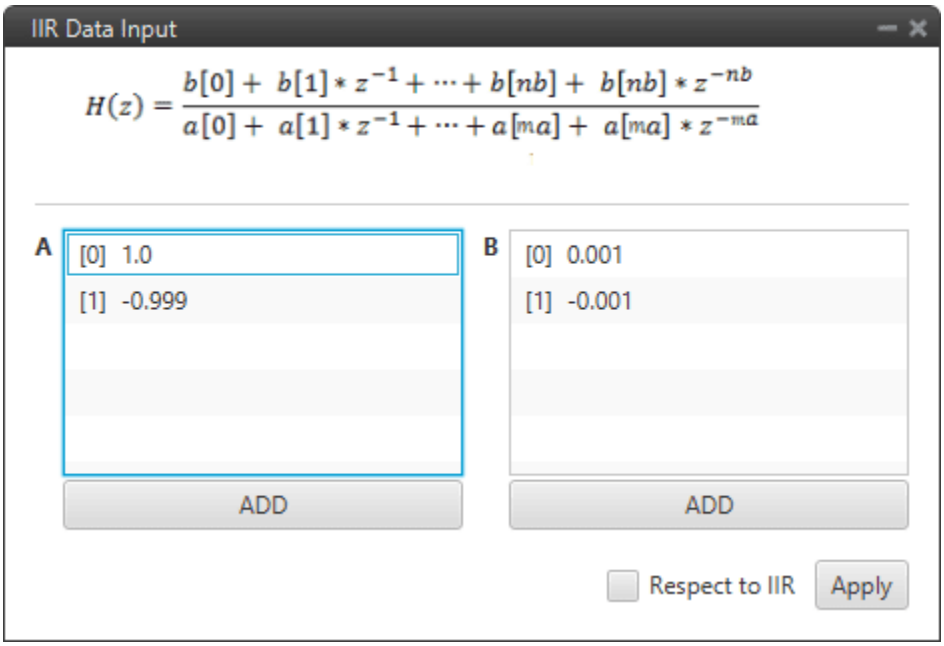


Fig. 457: **IIR Filter block - Derivative example**

2.9.14.9 Integrator

Integrator block is a numerical integrator. It calculates the numerical integral of the input signal using the **trapezoidal formula**.

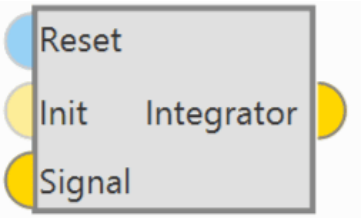


Fig. 458: **Integrator block**

- **Inputs**
  - (Optional) **Reset**: Optional reset, assumed false if not connected.
  - (Optional) **Init**: Optional **initial value of the integral**, if not connected a value of zero is assumed.
  - **Signal**: Signal to integrate.
- **Output**
  - **Pin 0**: Integrated signal.

2.9.14.10 Interpolation Vector

Interpolation Vector block applies the configured table interpolation over each of the components of the input vector.

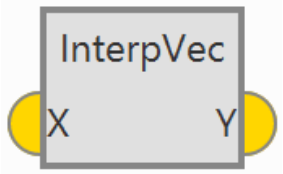


Fig. 459: **Interp Vec block**

- **Input**
  - **X**: Input vector.
- **Output**
  - **Y**: Interpolated output vector.
- **Configuration menu**:

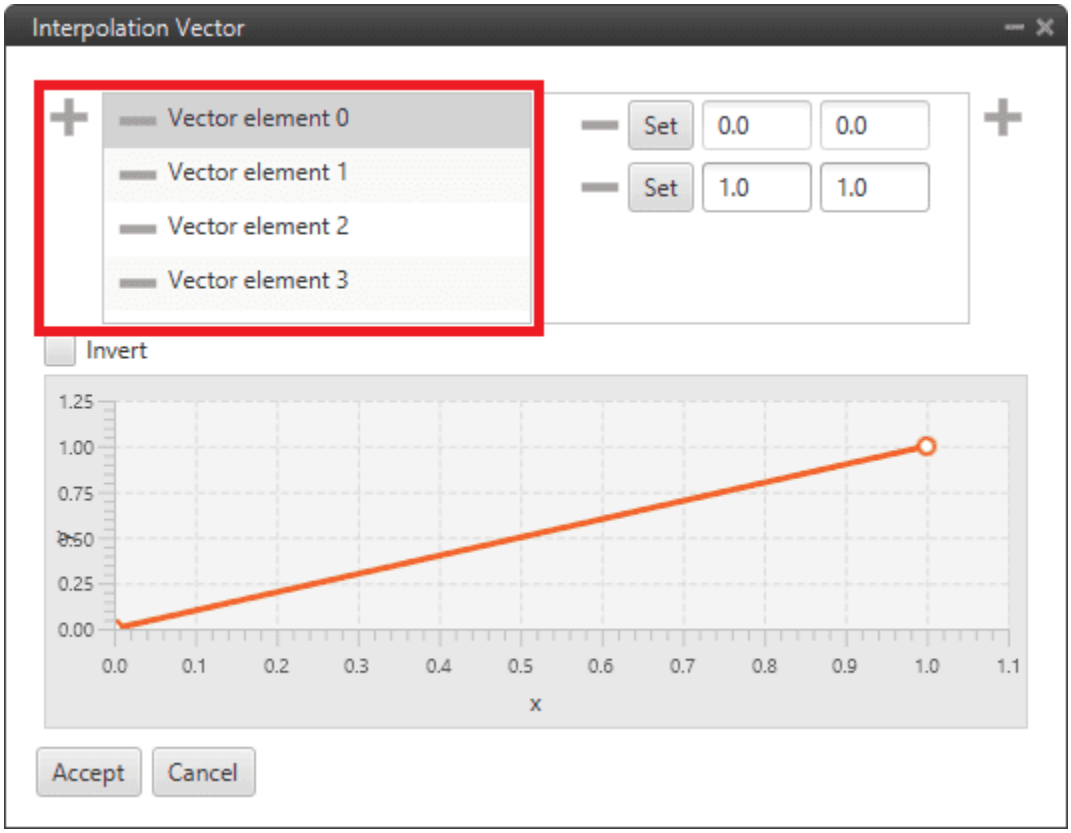


Fig. 460: **Interp Vec block configuration**

- : Users should add as many *Vector elements* as there are components in the input vector.
- **Points**: The interpolation function of each component must be configured by the user. It is represented by the graph below.
- **Invert**: If enabled, the *y axis* of the function will correspond to the input vector and the *x axis* to the interpolated output vector.

2.9.14.11 Ramp

Ramp block will **ramp up to** the **final value** defined as input, starting **from the initial value** defined as input, and respecting the parameters **Ramp Delay** and **Ramp Time**.

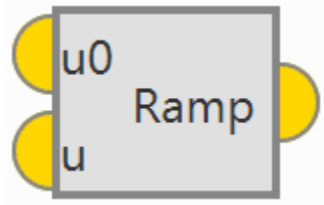


Fig. 461: **Ramp block**

- **Inputs**
  - **u0**: Initial value of the ramp, read only in on_focus.

- **u**: Final value of the ramp, updated in each step.

- **Output**
  - **Pin 0**: Output ramp value.
- **Configuration menu**:

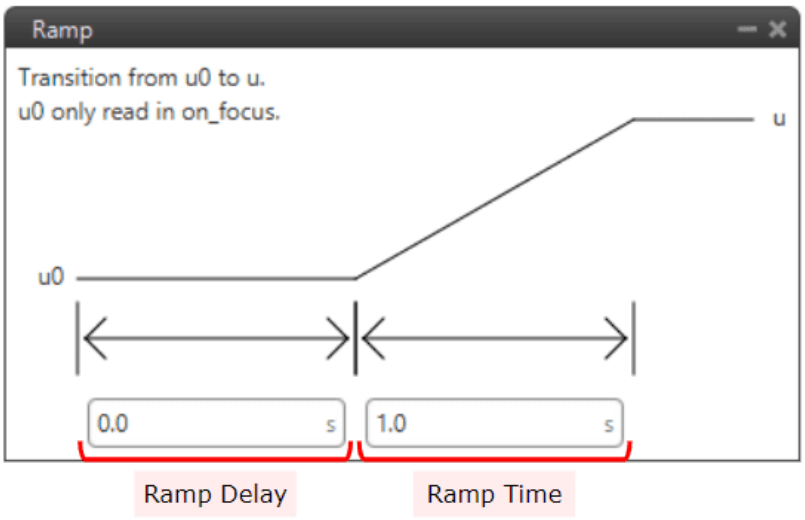


Fig. 462: **Ramp block configuration**

  - **Ramp Delay**: Time before starting the ramp.
  - **Ramp Time**: Time in which the variable must change from the initial value to the final value.

2.9.14.12 Rate limiter

Rate limiter block limits the rate of change of the input signal. It returns the signal input, but limiting its maximum rate of change.

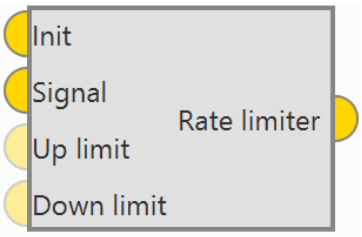


Fig. 463: **Rate limiter block**

If the rate of change of the input is higher than the maximum, the output will try to converge to the input, but respecting the imposed maximum rate of change.

The **first time** the block is executed the **output will be equal to Init**.

- **Inputs**
  - **Init**: Initialization value, this is the output of the block in the first step after on_focus.

- **Signal**: Input signal.
- (Optional) **Up limit**: Rate limit in the up direction. The value is read as **absolute value**, this means that the sign of this input is neglected.
- (Optional) **Down limit**: Rate limit in the down direction. The value is read as **absolute value**, this means that the sign of this input is neglected.

- **Output**
  - **Pin 0**: Rate-limited signal.
- **Configuration menu**:

Fig. 464: **Rate limiter block configuration**

  - **Angle wrap**: Perform a [-pi, pi] wrap. It should be enabled when using angles.

This block can be used to avoid instantaneous spikes and spikes in control signals, effectively reducing control noise and smoothing flight:

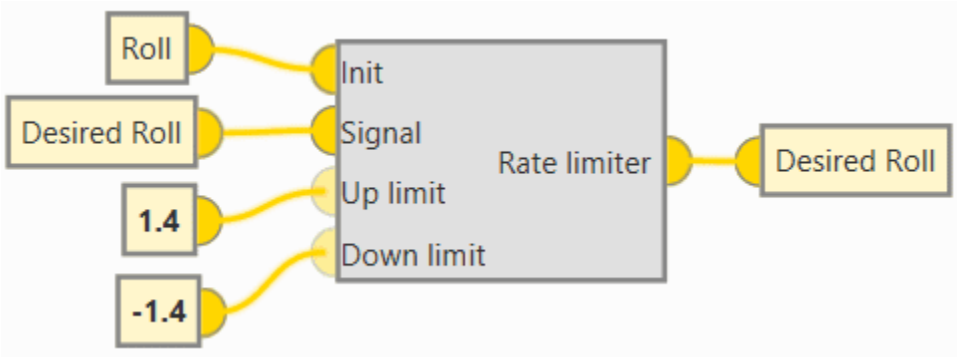


Fig. 465: **Rate limiter block - Example of use**

2.9.14.13 Signal generator

Signal generator block is a wave signal generator.

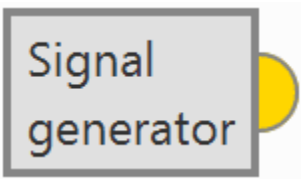


Fig. 466: **Signal generator block**

- **Output**

- **Pin 0**: Signal generated according to the configured type.

- **Configuration menu**:

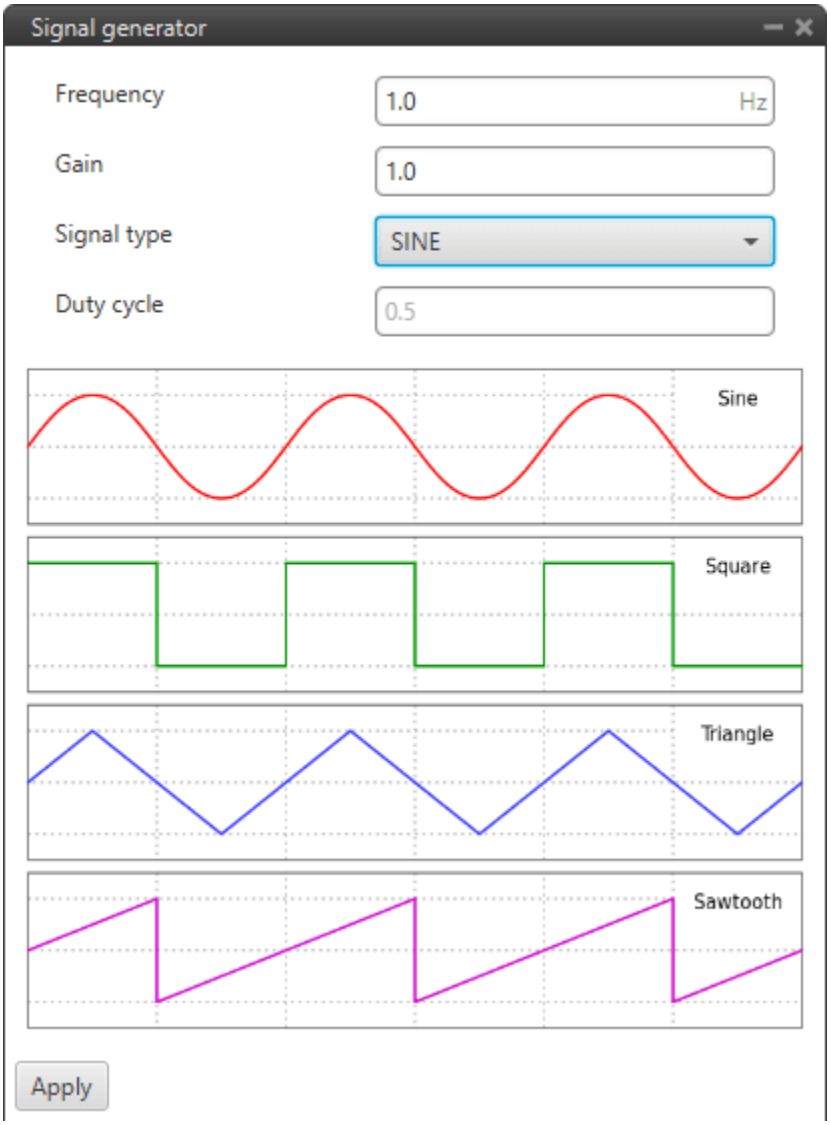


Fig. 467: **Signal generator block configuration**

  - **Frequency**: Frequency of the signal.
  - **Gain**: Gain of the signal
  - **Signal type**: Users must select the type of signal to be generated. The available options are: **Sine**, **Square**, **Triangular** and **Sawtooth**.

    Note: An example of the square signal type is shown below.

  - **Duty cycle**: Duty cycle of the signal. Can **only** be modified when **Square signal** is selected.

An example is given below:

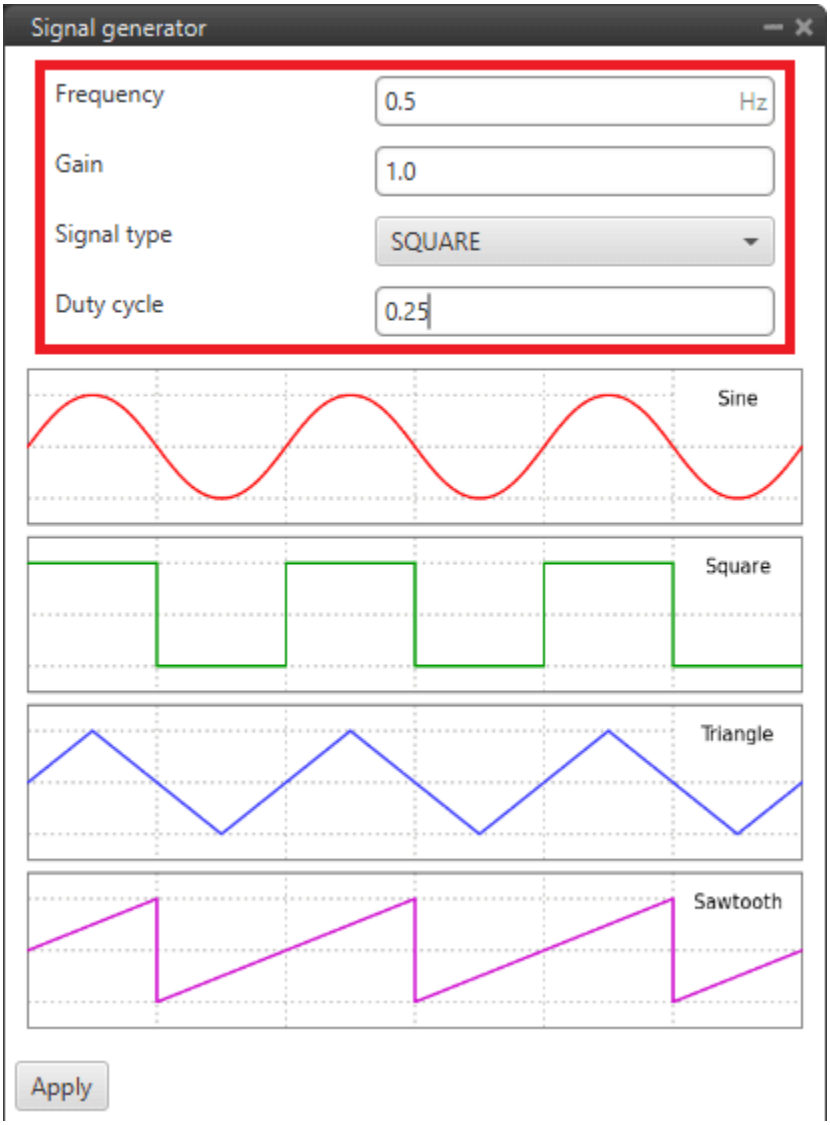


Fig. 468: **Signal generator block - Configuration example**

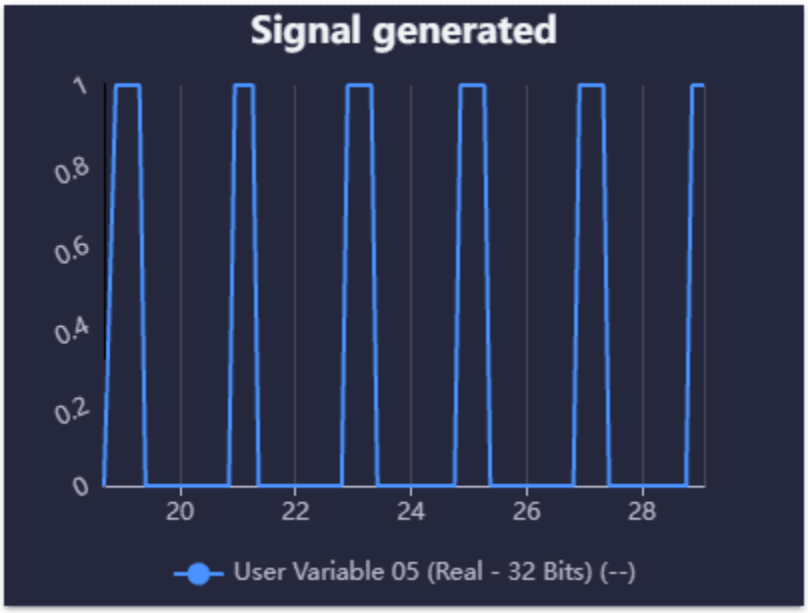


Fig. 469: **Signal generator block - Example of signal generated**

2.9.15 Type Casting blocks

These blocks allow to change from one data type to another. There are four different blocks available:

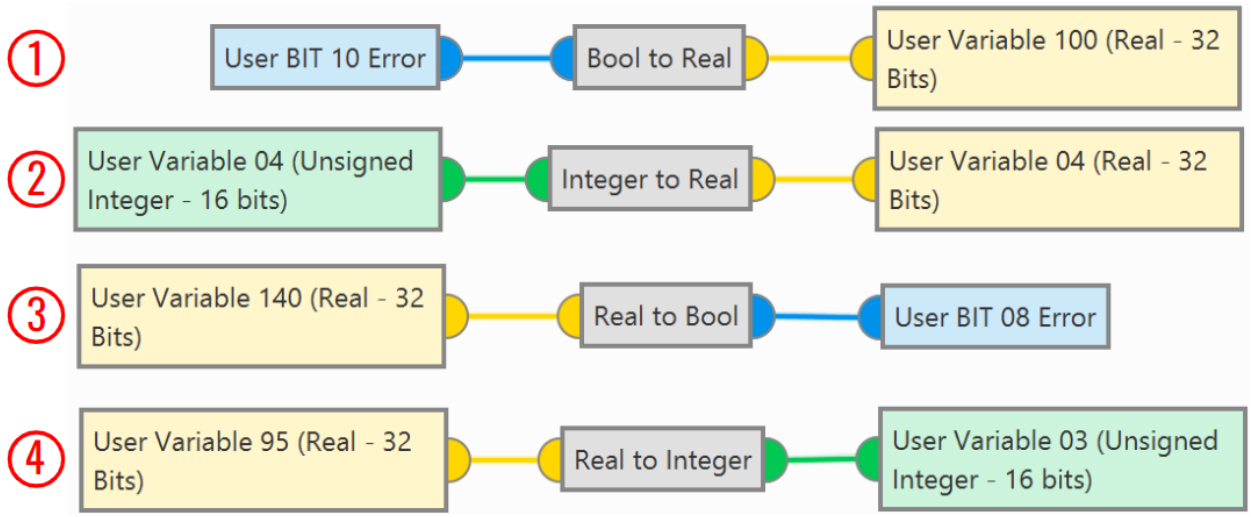


Fig. 470: **Type Casting blocks**

1. **Bool to Real**: It transforms a boolean variable to a real variable.
2. **Integer to Real**: It converts an integer variable to a real variable.
3. **Real to Bool**: It transforms a real variable to a boolean variable.

   Any number (**negative numbers included**), except 0, will be transformed to TRUE; **0** will be **FALSE**.
4. **Real to Integer**: It converts a real variable to an integer variable.

2.10 Devices

This menu displays the possible payloads/devices that can be configured with Veronte Autopilot 1x. Each panel will allow the user to configure different parameters from the available variety of payloads.

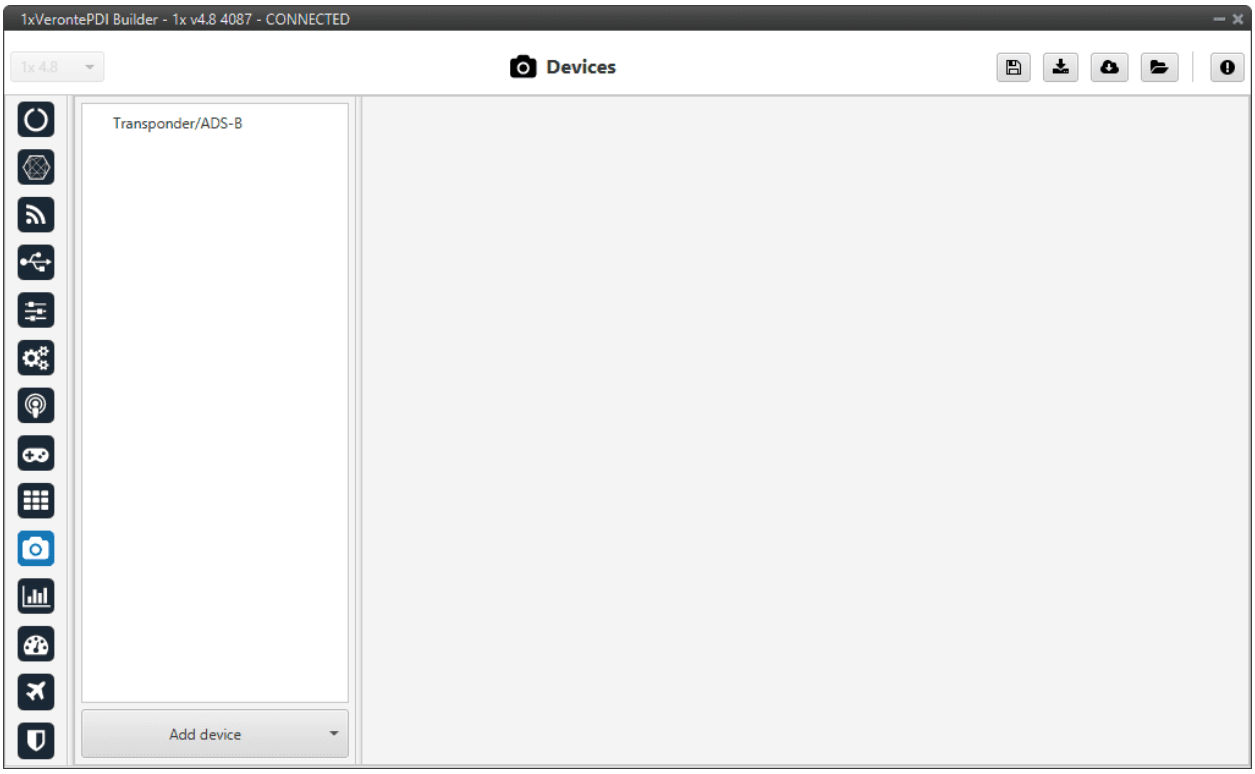


Fig. 471: **Devices menu**

By default, only the *Transponder/ADS-B* device is added to this menu (as shown in the figure above). However, users can added other devices supported by Veronte Autopilot 1x simply by clicking **Add device**:

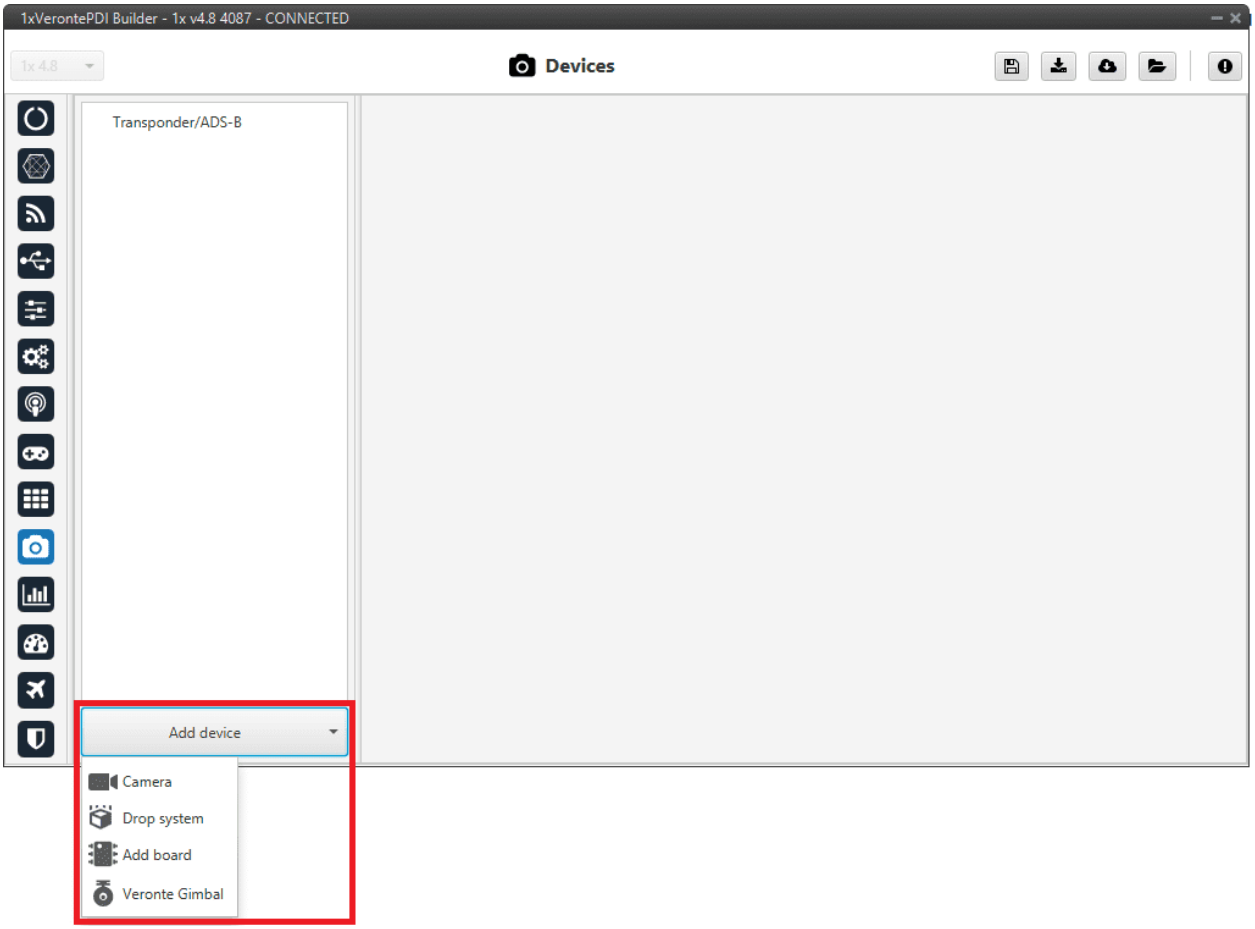


Fig. 472: **Devices menu - Add device**

2.10.1 Transponder/ADS-B

A transponder is a device that generates or parses **ADS-B messages**. This menu allows the user to use/configure a transponder through **Custom Messages**.

The interface includes different transponder models, depending on the one selected, some configurable parameters are enabled or disabled.

Important: To configure the **Squawk Code**, enable/disable **Ident Mode** and select the "dynamic" ADS-B Out mode, users must use the **ADS-B widget** of **Veronte Ops**. For more information on this, please refer to ADS-B - Flight instruments widgets of the **Veronte Ops** user manual.

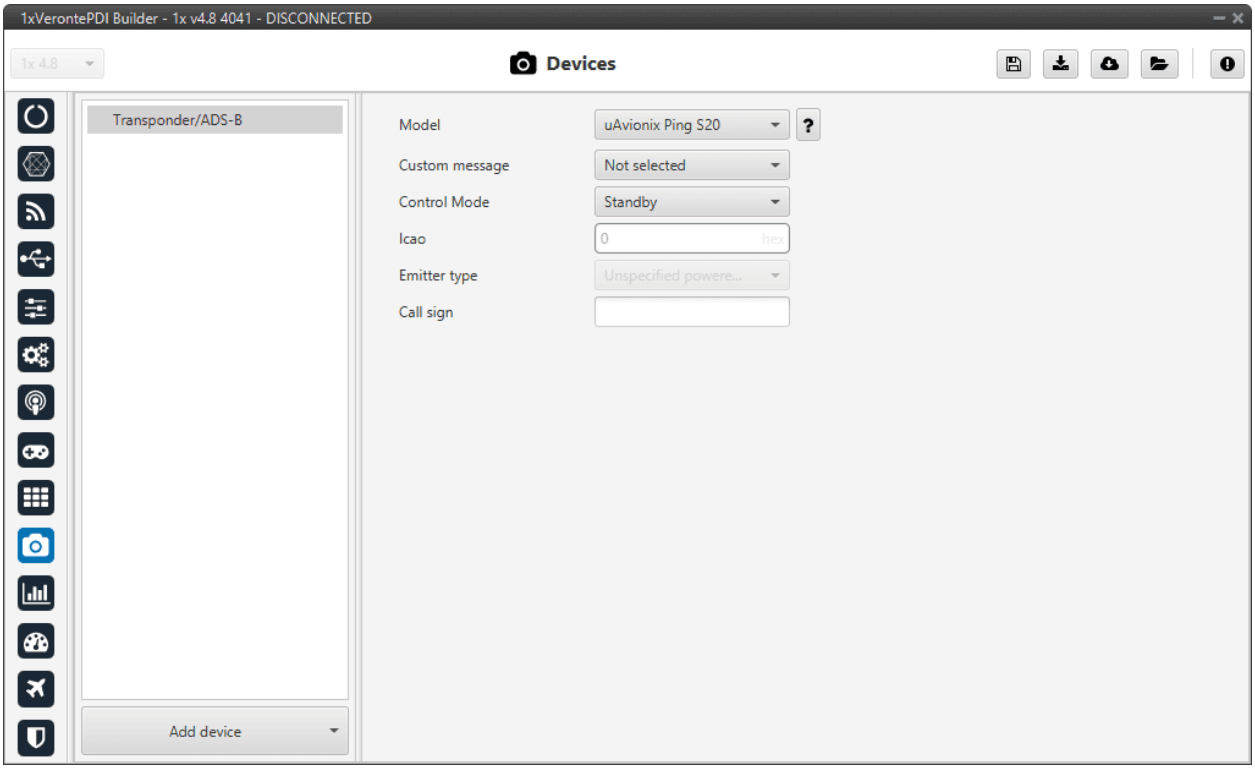


Fig. 473: **Transponder/ADS-B panel**

The parameters that appear in this menu and that must be configured by the user depending on the transponder model are explained below:

- **Model**: The options available in Veronte Autopilot 1x are as follows:
  - Internal Transponder
  - uAvionix Ping 20s
  - uAvionix Ping 1090
  - Sagetech MX Series
  - Sagetech XPS-TR
  - Sagetech XPG-TR
  - Sagetech XPS-TRB
  - Daedalean

  By clicking on the ? button next to the drop-down menu, an "information" dialog window about the selected model will appear. For example:

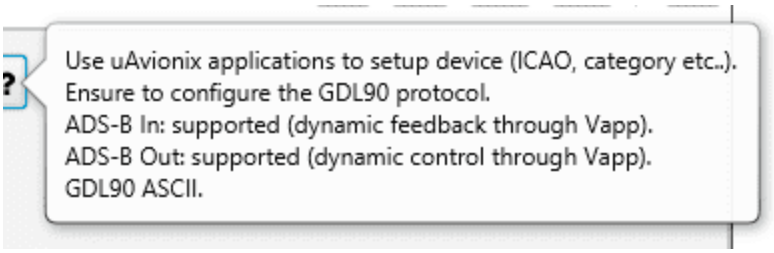


Fig. 474: **Transponder/ADS-B panel - Information dialog**

- **Custom message**: Select the *Custom message 0-2* to be used for the information exchange, as it will be automatically filled with the information required by the transponder/ADS-B.
- **Control Mode**: The user must configure the transponder type:
  - **Standby**: Silence mode.
  - **ADS-B In**: To detect external aircraft.
  - **ADS-B Out**: To communicate with external aircraft. There are 2 different types of information that can be shared:
    - **Static information**: This information should not change during operation (flight). Depending on the model, it can be configured from this app (1x PDI Builder) or from the app of each transponder.
    - **Dynamic information**: Information that can change during operation (flight). It is configured from the ADS-B widget of the **Veronte Ops** software.
  - **ADS-B In/Out**: To detect and communicate with external aircraft.
- **Icao**: Unique ICAO 24-bit address **permanent** for the aircraft, which becomes a part of the aircraft's Certificate of Registration.
- **Emitter type**: The user should specify the type/category of the vehicle in which the transponder is located.
- **Call sign**: Communication call sign assigned as unique identifier to the aircraft.

The following table summarizes the options that can be **selected** for each transponder model in the **1x PDI Builder** software:

Model	Custom message	Control Mode					Icao	Emitter type	Call sign
		Standby	ADS-B In	ADS-B Out		ADS-B In/Out			
				Static	Dynamic (through Veronte Ops)				
Internal Transponder	Not available	Available	Available	Available	Available	Available	Available	Available	Available
uAvionix Ping s20	Available	Available	Not available	Not available	Available	Not available	Not available	Not available	Not available
uAvionix Ping 1090	Available	Available	Available	Available	Available	Available	Available	Available	Available
Sagetech MX Series	Available	Available	Available	Not available	Available	Not available	Not available	Not available	Not available
Sagetech XPS-TR	Available	Available	Not available	Not available	Available	Not available	Not available	Not available	Not available
Sagetech XPG-TR	Available	Available	Not available	Not available	Available	Not available	Not available	Not available	Not available
Sagetech XPS-TRB	Available	Available	Available	Not available	Available	Not available	Not available	Not available	Not available
Daedalean	Available	Not available	Available	Not available	Not available	Not available	Not available	Not available	Not available

Caution: The control mode can be modified from the ADS-B widget of the **Veronte Ops** software.

2.10.2 Camera

Adding a camera will create a default Camera configuration menu.

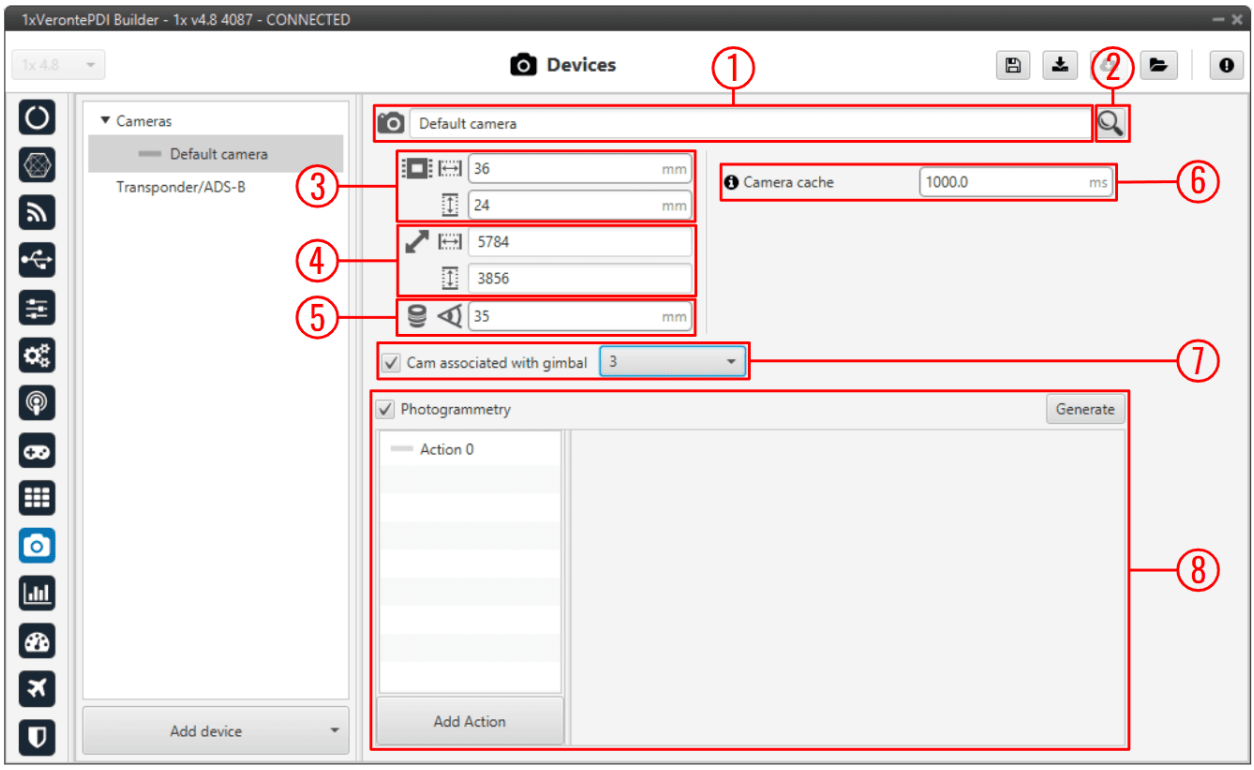


Fig. 475: **Cameras panel**

1. **Name**: Give a name to the customized camera.
2. **Search**: Users can also choose a camera from the predefined list of cameras, which will automatically establish the values of the **Sensor**, **Resolution** and **Lens** parameters.
3. **Sensor**: Defines the camera sensor width and height in mm.
4. **Resolution**: Defines the camera resolution width and height.
5. **Lens**: Defines the focal length from the camera in mm.
6. **Camera cache**: Defines the **cache time used to play the selected camera on the gimbal widget**.
   - A **higher** cache might increase the **video delay**.
   - A **lower cache** might cause **video artifacts or disconnections**.

   Tip: 333 miliseconds should be enough for a 1080p video.

   ---

7. **Cam associated with gimbal**: If the camera is **from a Gimbal device**, it is important to configure this field and select **the Gimbal block number** that is related to this camera.
8. **Photogrammetry**: This allows the creation of Photogrammetry actions.

   The actions performed in a Photogrammetry mission can be defined here, following the same possibilities as in *Actions - Automations menu*.

   - **Add Action**: Will add a new action.

> **Warning:** A **maximum of 4 Actions** can be defined (*Actions 0-3*).

- **Generate**: Clicking on 'Generate' will create the automation 'Photogrammetry' with a Button as event and with the actions defined here.

An example is given below:

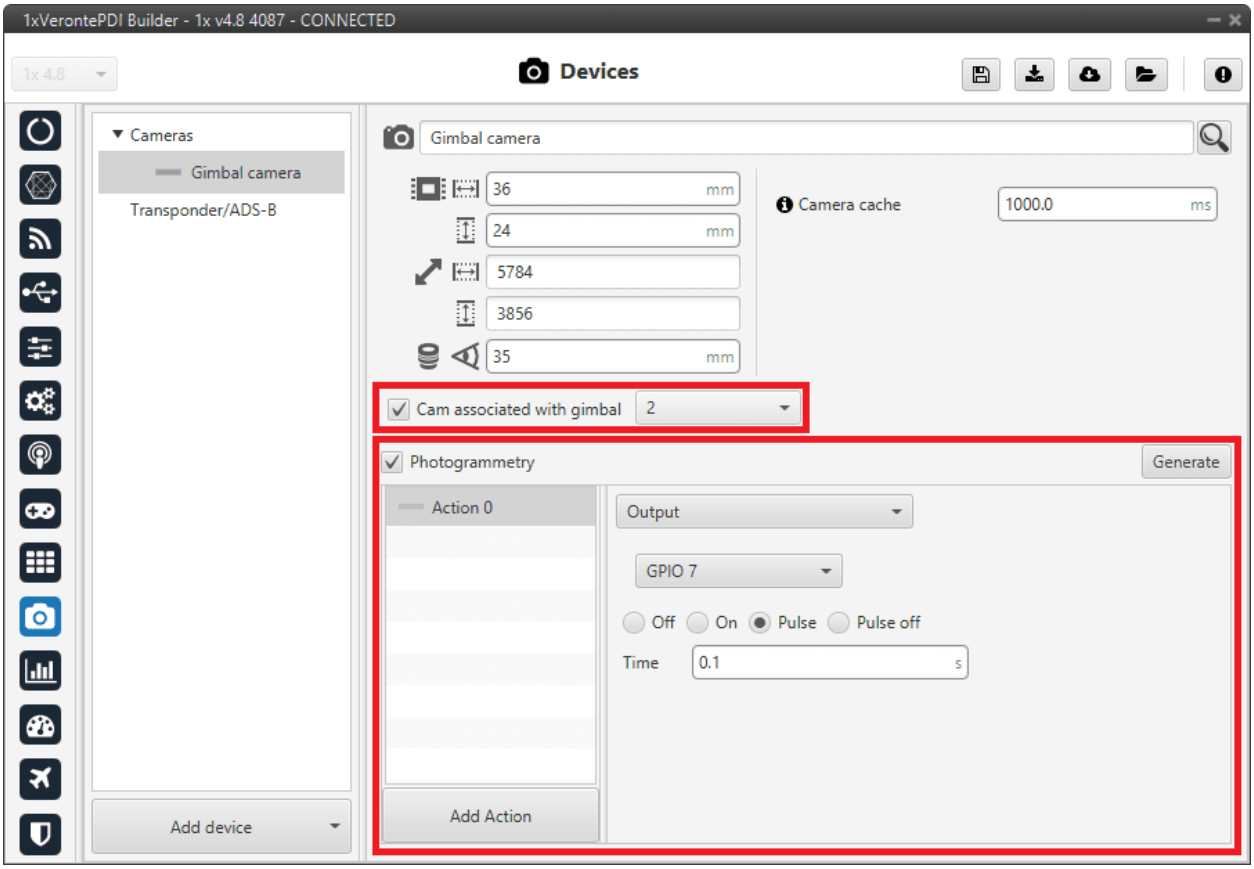
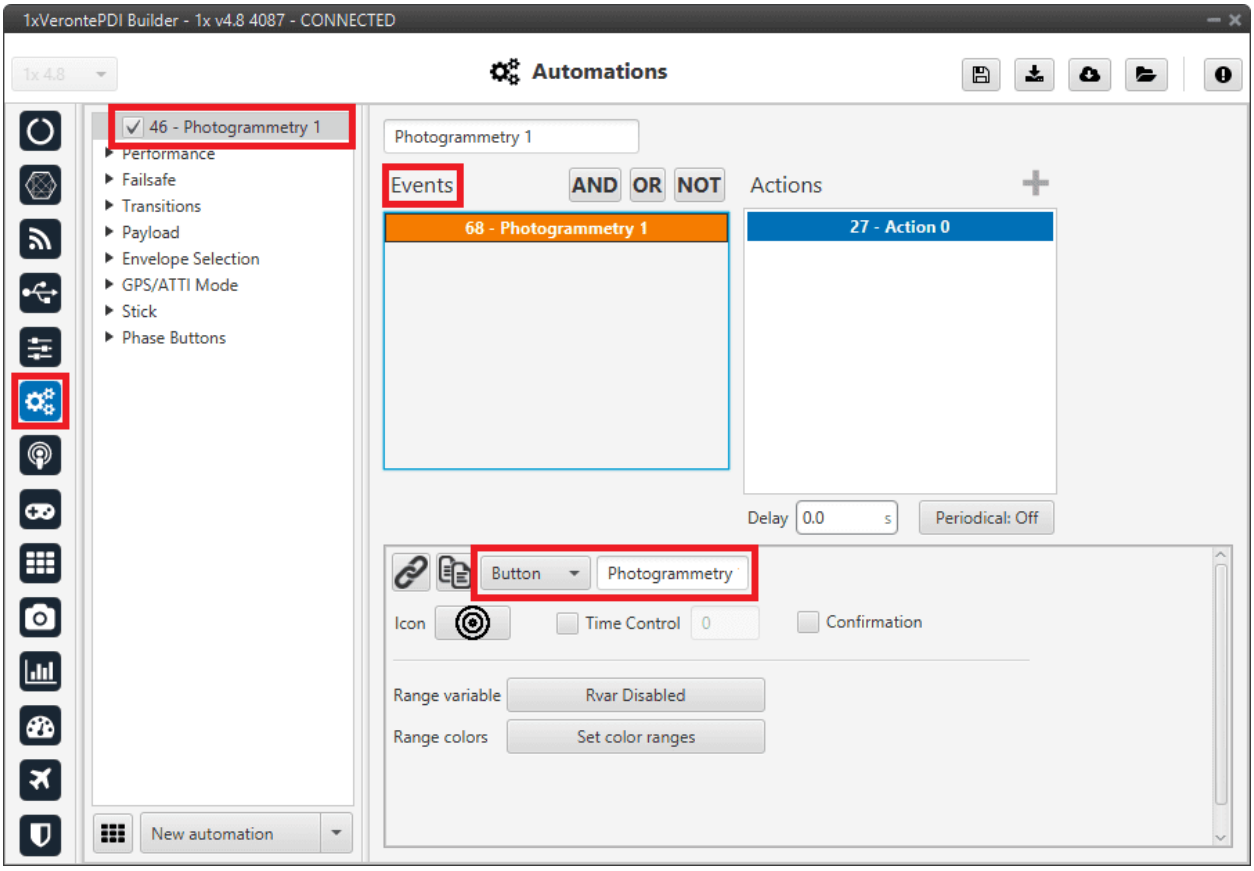


Fig. 476: **Cameras panel - Example**

The automation created for Photogrammetry is shown below:


1xVerontePDI Builder - 1x v4.8 4087 - CONNECTED
Automations
46 - Photogrammetry 1
Performance
Failsafe
Transitions
Payload
Envelope Selection
GPS/ATTI Mode
Stick
Phase Buttons
New automation
Photogrammetry 1
Events
AND
OR
NOT
Actions
68 - Photogrammetry 1
27 - Action 0
Delay
0.0
s
Periodical: Off
Button
Photogrammetry
Icon
Time Control
0
Confirmation
Range variable
Rvar Disabled
Range colors
Set color ranges

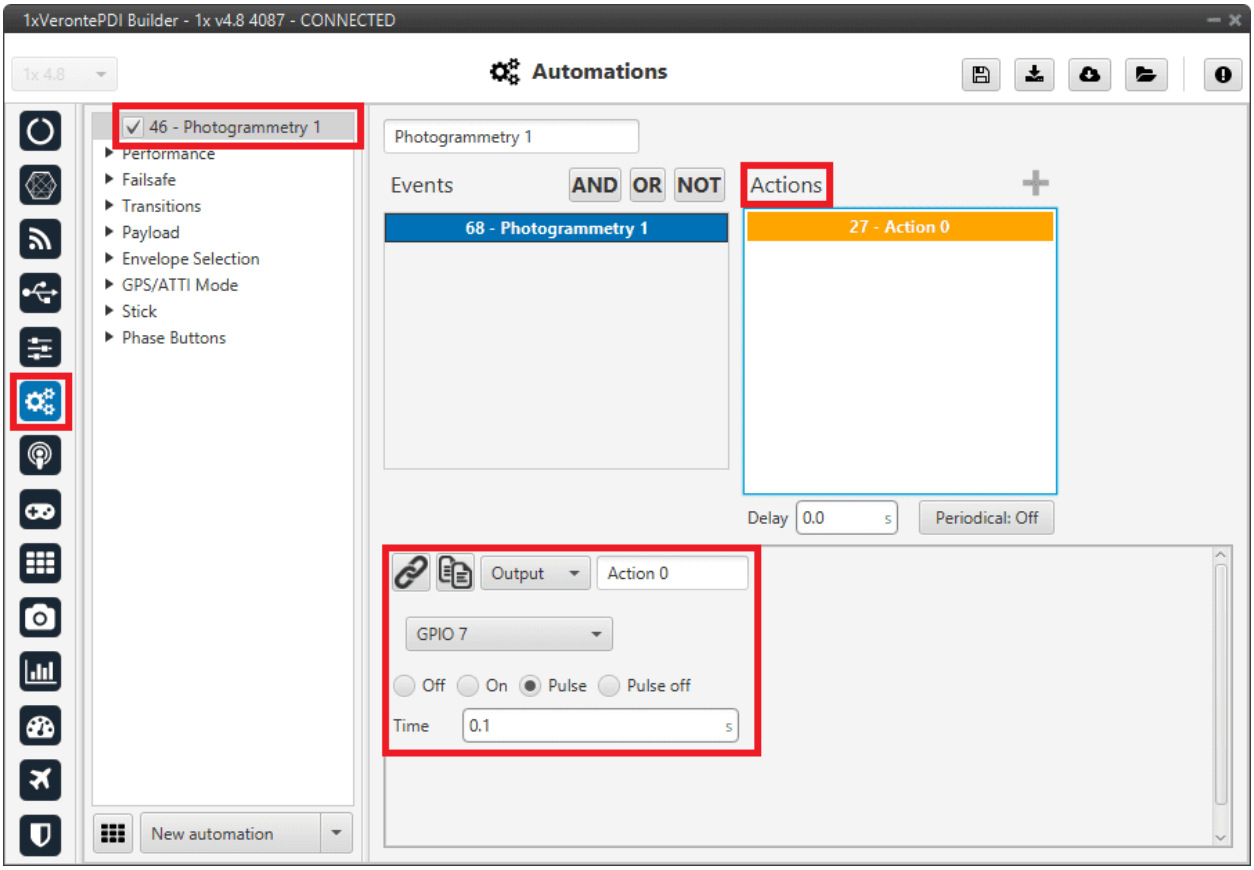


Fig. 477: **Cameras panel - Automation example**

2.10.3 Board

This board menu allows the user to configure **communicate via CAN**, in **1x PDI Builder**, with another device such as *CEX*, *MEX*, *MC01*, etc., in **only 1 step**, so in only 1 interface window. Instead of doing it in several steps, as is explained in *CEX/MEX - Integration examples* section or *MC01 - Integration examples* section.

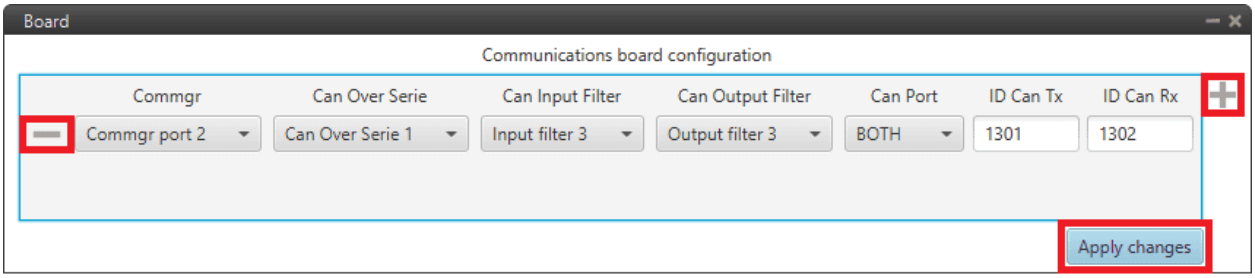


Fig. 478: **Board panel**

- Click on the + icon to add a new field. The following parameters must be defined:
  - **Commgr port**: Select the desired COM Manager port: *Commgr port 0-5*.
  - **Can Over Serie**: Select the desired **Serial to CAN / CAN to serial**: *Serial to CAN 0-1/CAN to Serial 0-1*.
  - **Can Input Filter**: Select the desired input filter to use: *Input Filter 0-3*.

- **Can Output Filter**: Select the output filter the user wishes to use: *Output Filter 0-3*.
- **Can Port**: *CAN A*, *CAN B* or *BOTH* can be selected.
- **ID Can Tx**: Enter the ID of the CAN message to be sent.
- **ID Can Rx**: Enter the ID of the CAN message to be received.

> **Warning:** Be careful not to select a producer/consumer that is being used for another purpose, as the configuration defined here has "priority" and will be changed to this.

For more information on these parameters, see *Input/Output* section of this manual.

- Clicking the icon will remove the field and display a confirmation warning message.
- By clicking on '**Apply changes**', all these CAN communication settings are applied to the Autopilot 1x configuration.

Below is an example of before/after when changes are applied:

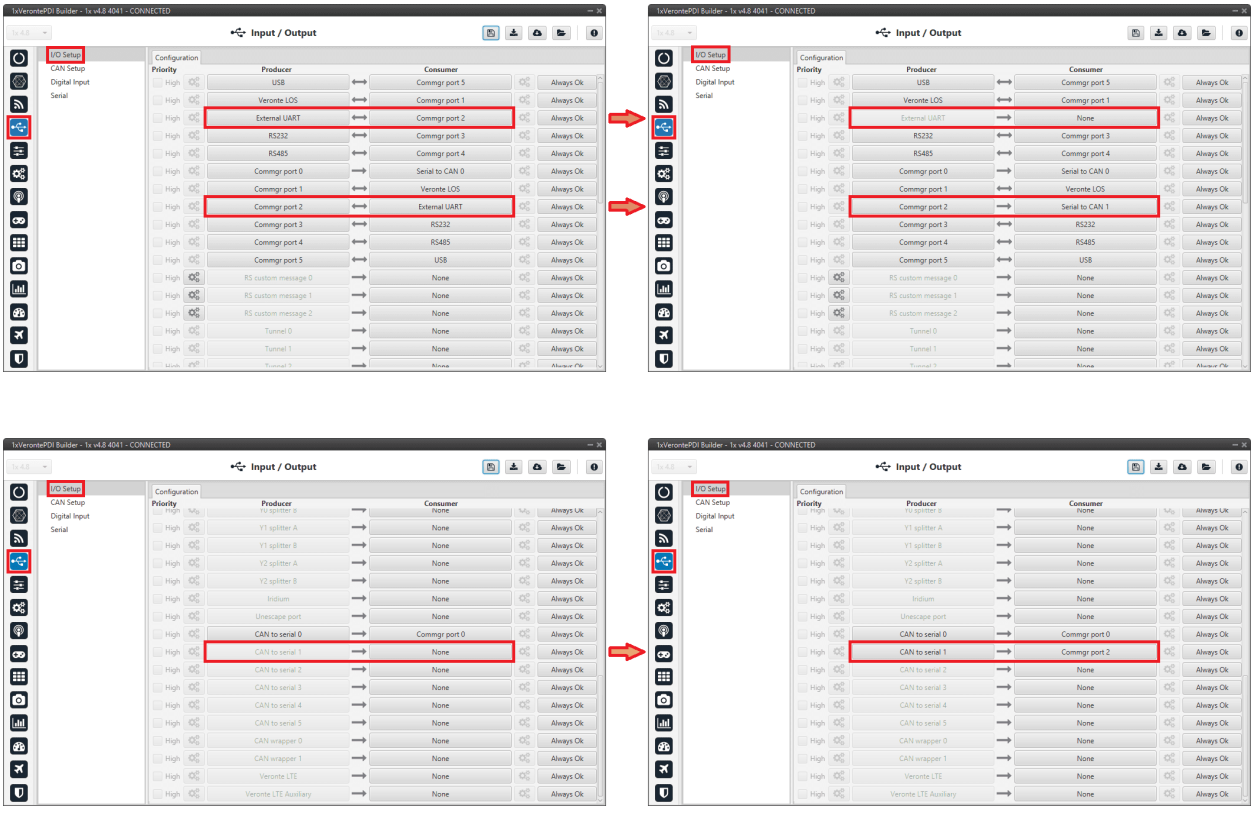


Fig. 479: **Board panel - I/O Setup configuration**

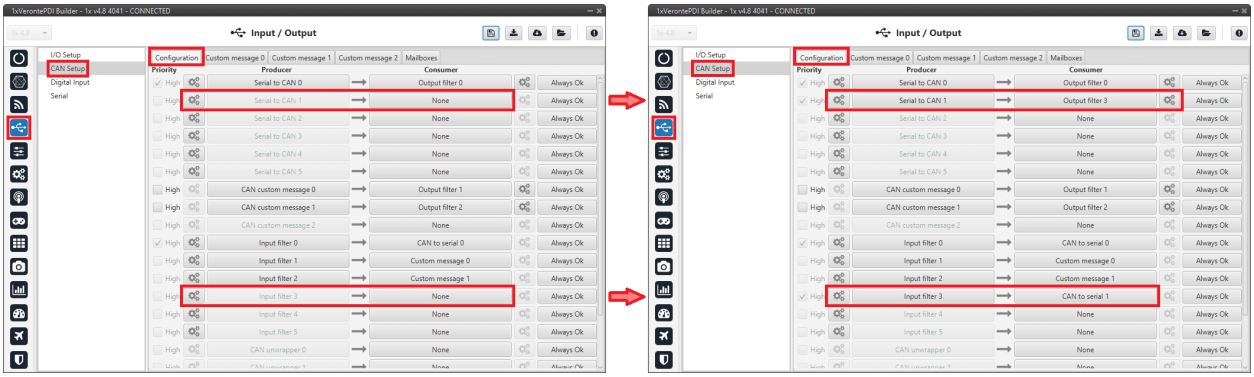


Fig. 480: **Board panel - CAN Setup configuration**

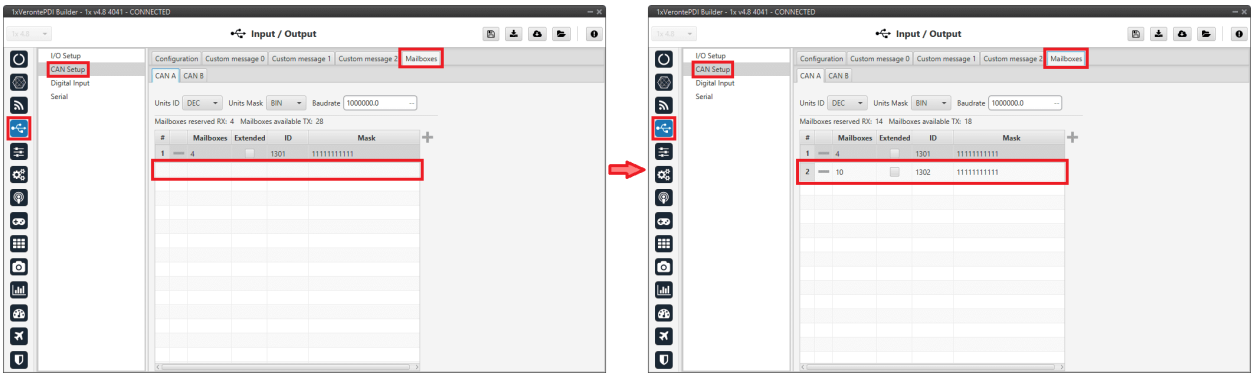


Fig. 481: **Board panel - Mailboxes configuration**

2.11 Telemetry

In this menu there are 2 options available: **Telemetry** and **Sniffer**.

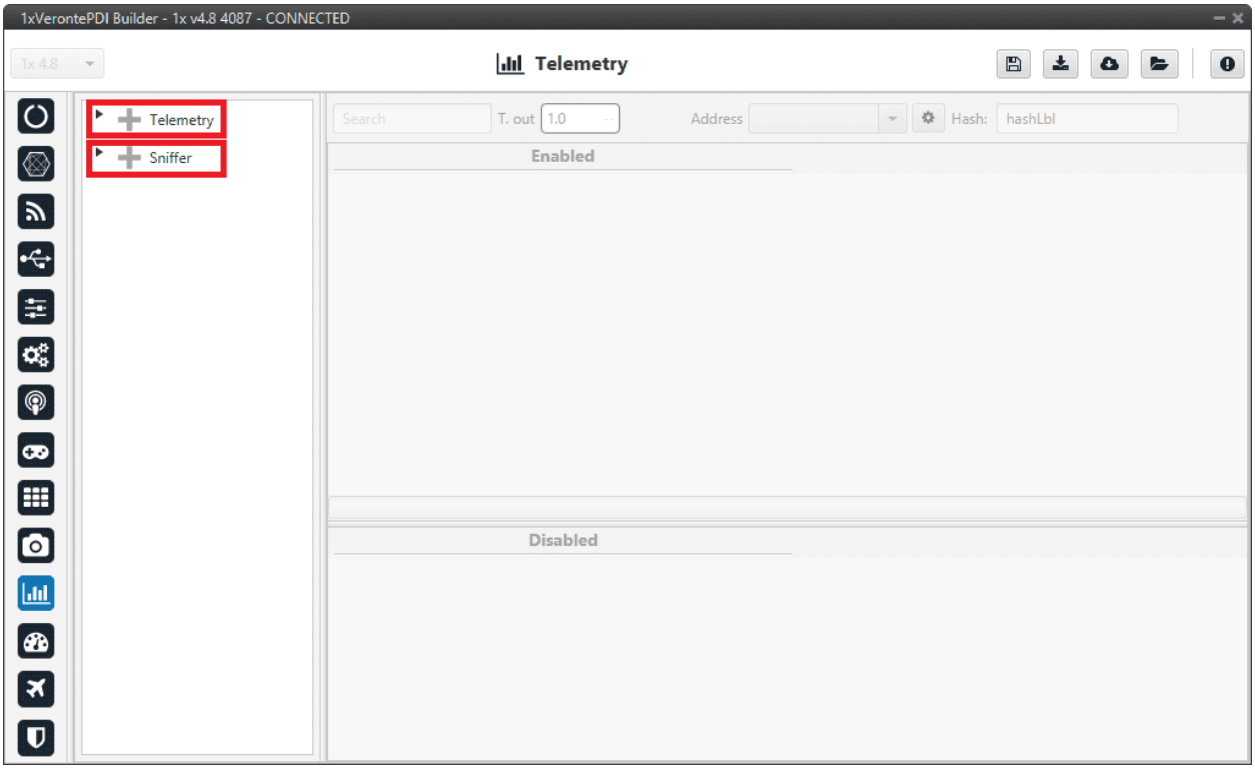


Fig. 482: **Telemetry menu**

2.11.1 Telemetry

Telemetry controls permit to configure data to be stored or transmitted on the system. There are 4 main items that can be configured within this panel:

Type	Description
Data vectors	Configures the variables to send throughout the data link channel.
Onboard Log	Sets the variables to be stored on system Log (on 1x SD Card).
User Log	User Log for custom applications.
Fast Log	Saves data at the maximum frequency available on the system. Recording time depends on the selected variables.

Configuration display permits to enable the desired variables for each telemetry file and to set the maximum and minimum values together with precision for each one.

2.11.1.1 Data vectors

This panel contains the variables that **Veronte Autopilots 1x** send to the indicated address.

By default, **1x PDI Builder** generates a Data vector to send telemetry from the configured Autopilot 1x to the **Address 2** (Veronte applications). The variables indicated in **red** in a Data vector are **required for correct operation**.

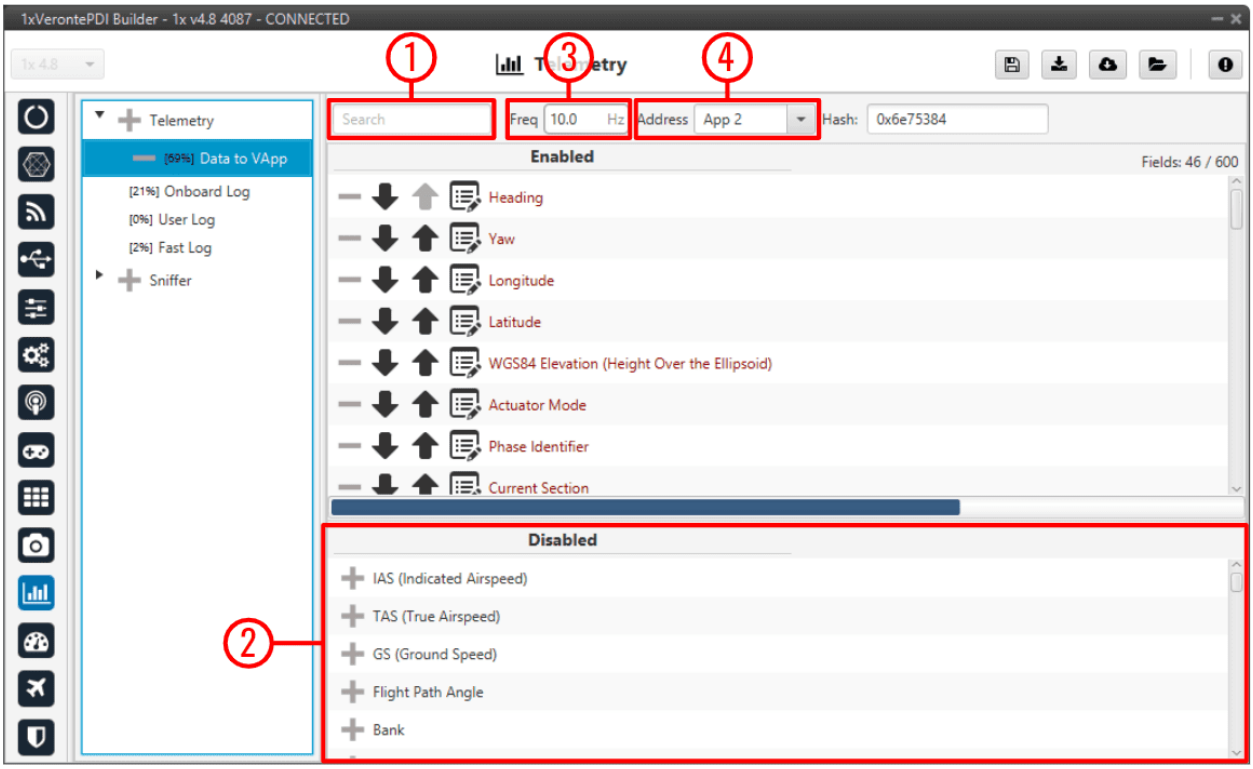


Fig. 483: **Data vectors panel**

In order to configure the variables sent, users have to:

1. **Search**: Search the desired variables into the **Disabled** panel.
2. **Disabled**: When the desired variables are found, add them to the **Enabled** panel by dragging and dropping them into it or simply by clicking on the + button.
3. **Freq**: Specify the sending rate. 10 Hz usually works well, this frequency depends on the bandwidth of the radio.
4. **Address**: Enter the corresponding address, the following options are the most common: *App 2* (Veronte applications address), *Broadcast* (all units on the network) and *Veronte device address* (to a specific unit). For more information on the available addresses, see List of addresses section of the **1x Software Manual**.

Note: **Hash** parameter is not configurable, it is automatically calculated by the system based on the telemetry vector configured by the user.

It is a hexadecimal representation of the CRC of the fieldset.

1x PDI Builder allows the creation of more Data links by simply pressing in the + icon next to 'Telemetry'.

As an example, another possible data link could be set between the **air** and **ground** autopilots directly (without **Veronte applications**) and used to send the position of the UAV to the ground autopilot for the configuration of a tracker. This data link example is presented in the following figure.

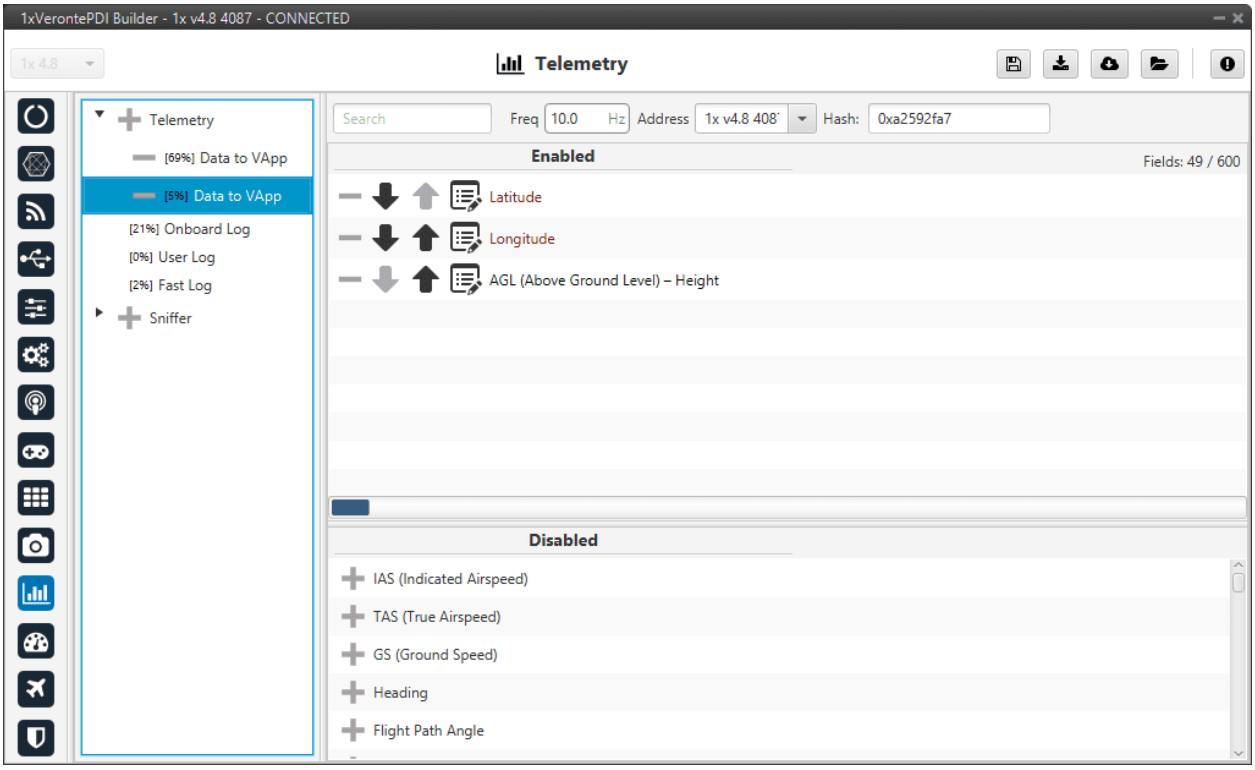


Fig. 484: **Data vectors panel - Ground/Air units communication**

If we consider this configuration as that of the air unit, this Autopilot 1x will send the Latitude, Longitude and AGL to the autopilot with address 1x v4.8 4087. The unit that receives the telemetry has to configure its *sniffer* (more information about this in the following section) in order to store the data.

Warning: If the number of variables enabled for telemetry communication are higher than the maximum supported by the system, the latest variables will not be sent, so they will display a zero value if shown in the workspace.

Note: It is possible to create more than one data link associated to the same receiver address, and they can also have different sending rates. It could be useful in case one of the data links is almost full.

2.11.1.2 Onboard Log

The Onboard Log determines the variables that are being **stored** on the **autopilot SD Card**. In this case, there are not sending/receiving units, so the only thing to configure here is the list of variables that will be saved on the autopilot internal memory for a further download and processing, as well as the writing frequency.

The log starts writing once the autopilot is turned on and **does not stop logging until the autopilot is turned off**.

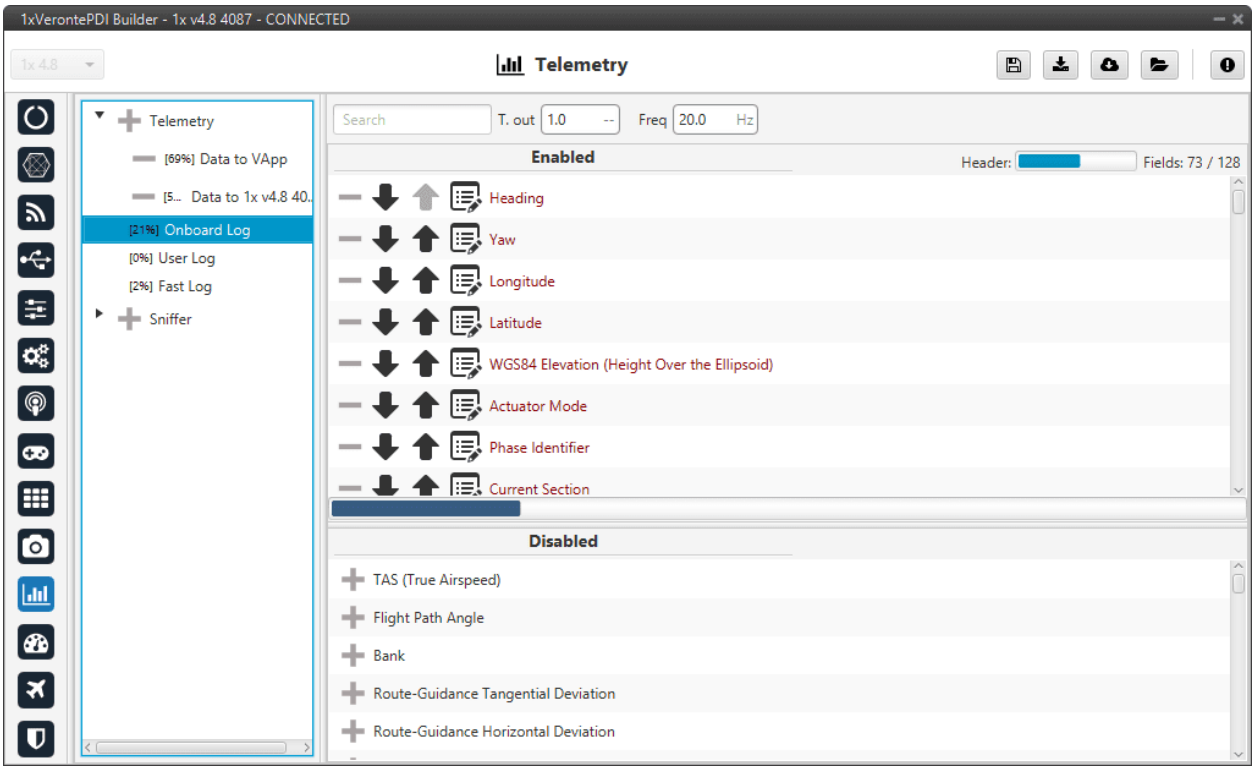


Fig. 485: **Onboard Log panel**

Warning: This is a circular log, which means that if the SD card memory is full, Veronte Autopilot 1x will delete the oldest logs automatically so it can continue logging.

Autopilot 1x has **2.5 GB** of memory reserved for these logs. Hence the registered time can be calculated, for example:

- If 10 variables are stored with 4 bytes each one, then each log will occupy 40 bytes
- With a frequency of 10 Hz, the writing speed will be 400 bytes/s
- $\frac{2.5\ GB}{400\ bytes/s} = 6710886\ s = 1864\ h = 77.7\ days$

2.11.1.3 User Log

The user log contains the variables that are stored according to an automation created by the user.

Considering an example, in a photogrammetry mission it is important to record the aircraft location when the photo is taken, so a user log could be used to record a certain set of variables (position, speed, direction, ...) each time a photo is taken.

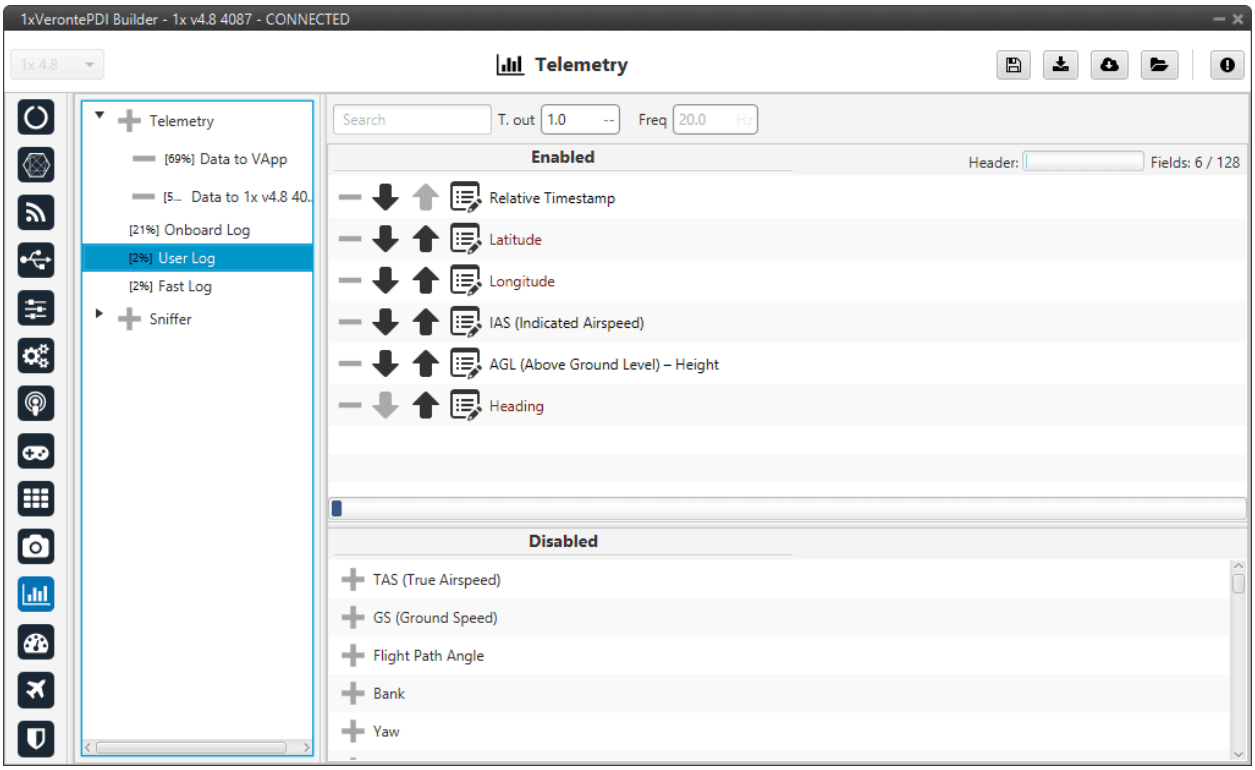


Fig. 486: **User Log panel**

In order to create a **User Log action** where an entry is added to the log when a certain set of events are accomplished check the *User Log action - Automations* section of this manual.

2.11.1.4 Fast Log

The fast log store the specified variables at the maximum rate available on the system. This tool could be used to save information in an operation that happens extremely fast, such as missile launching. The time that this logging process lasts depends on the number of variables being saved.

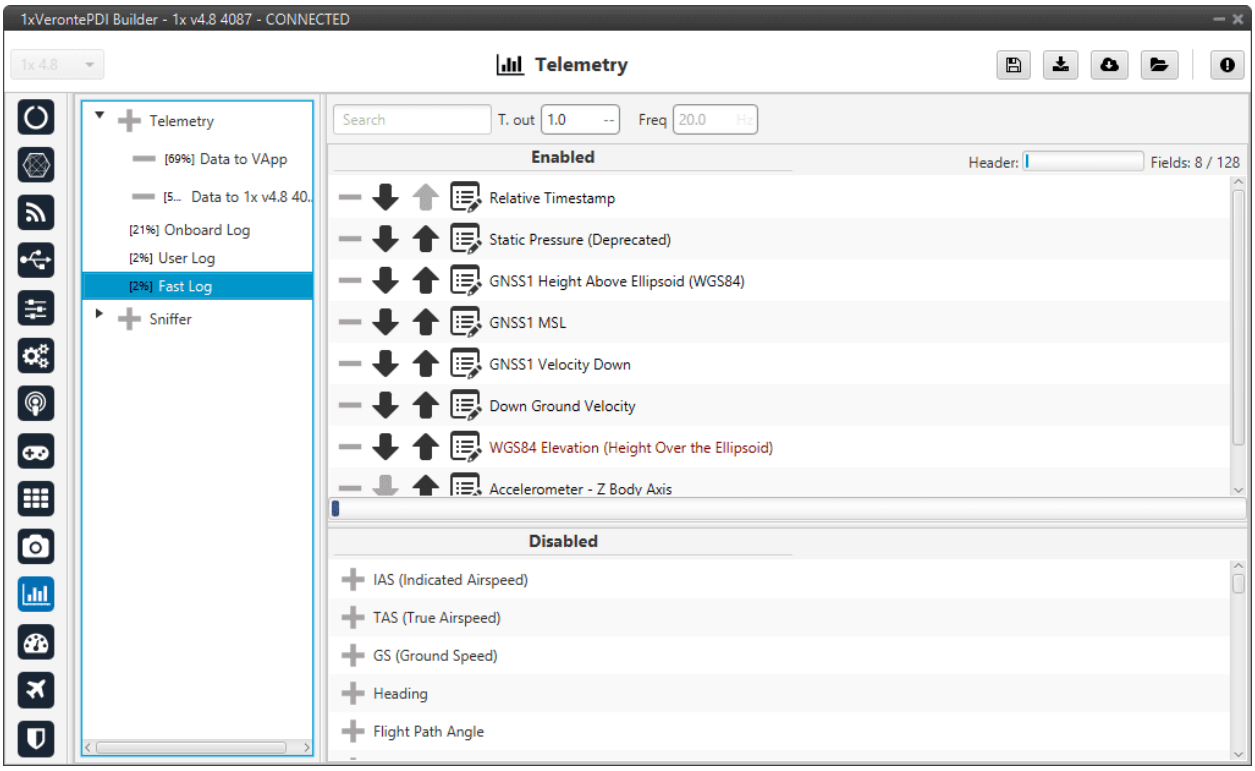


Fig. 487: **Fast Log panel**

The Fast Log can be activated when the user wants, but must be done in **Veronte Ops**. For more information about Fast Log, check the Veronte Ops manual.

Important: The downloading of the information of an operation depends on how it has been stored, i.e depends on the type of log (data link, onboard, user or fast log).

- For more information related to **Onboard Log**, **User Log** and **Fast log** downloading, visit Veronte FDR user manual.
- Visit Veronte Link user manual for information about **Data vectors** downloading.

Besides, Autopilot 1x includes some compression tools that may be useful for increasing the amount of information transmitted in a certain bandwith or stored in a log. Each variable can be compressed separately in each log.

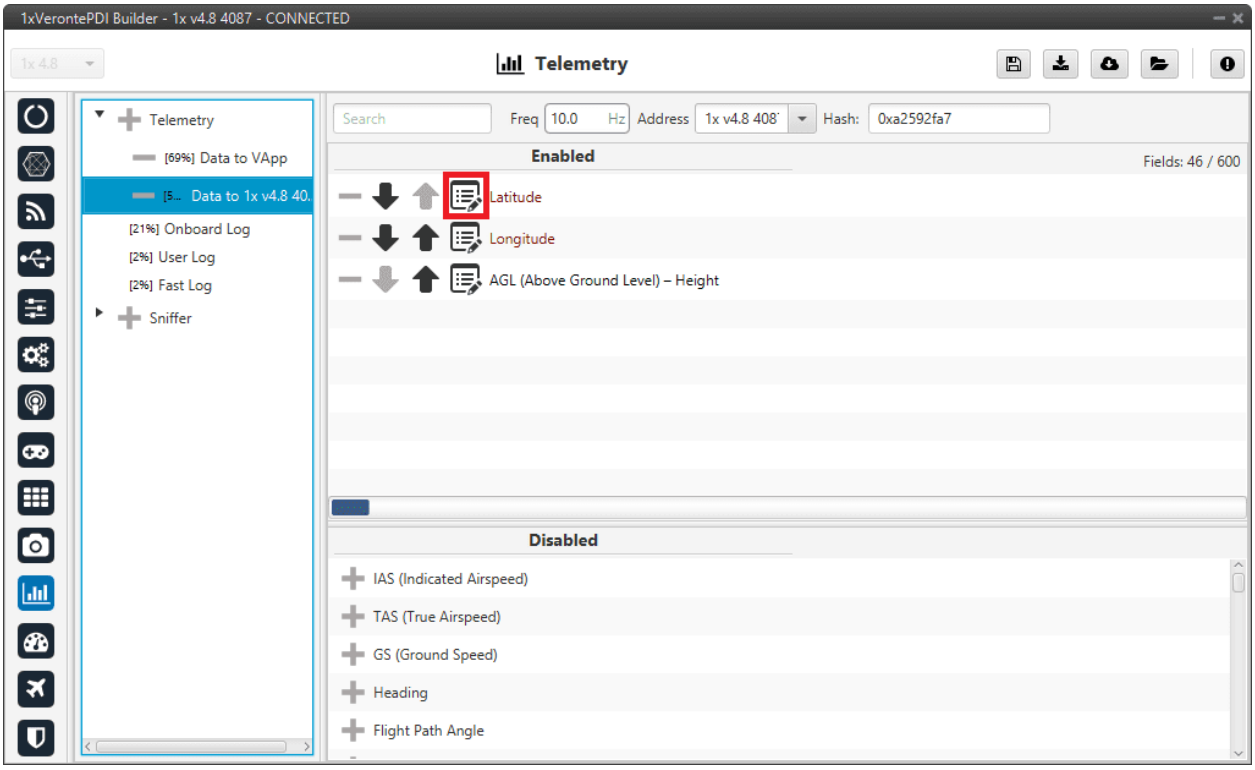


Fig. 488: **Compression options**

There are different types of compression available:

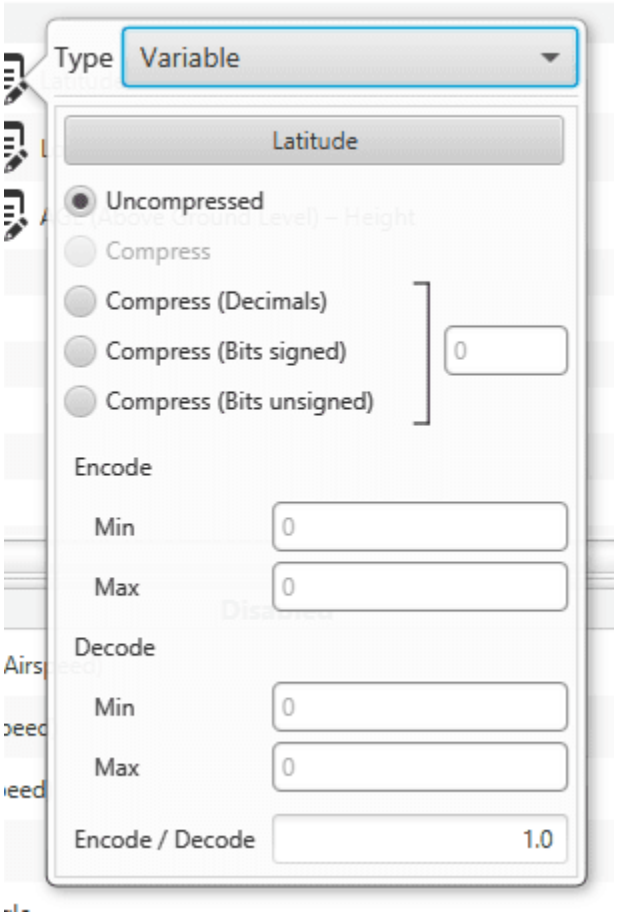


Fig. 489: **Compression options panel**

- **Uncompressed**: The variable is taken in its full length, with no value modification.
- **Compress (Bits signed)**: Specify the number of bits to be compressed to (**negative values accepted**). It is necessary that the user configures **Encode/Decode** options.
- **Compress (Bits unsigned)**: Specify the number of bits to be compressed to (**no negative values accepted**). It is necessary that the user configures **Encode/Decode** options.
- **Compress (Decimals)**: The variable is compressed according to the number of decimals specified and the range specified (max and min values). The resultant compression (number of bits) follows the relation $(max - min) \cdot 10^{decimals}$, which yields the encoding of the maximum value of the range (and the number of bits necessary for that). The range needs to be specified on the **Encode - Min/Max** field.
- **Encode/Decode**: These values are used to apply a scaling factor after the transformation from binary to decimal value, or before the transformation from decimal to binary value.

In the example shown below, the Heading variable with 3 decimals will be compressed, so instead of using 32 bits, it will only require 19 bits.

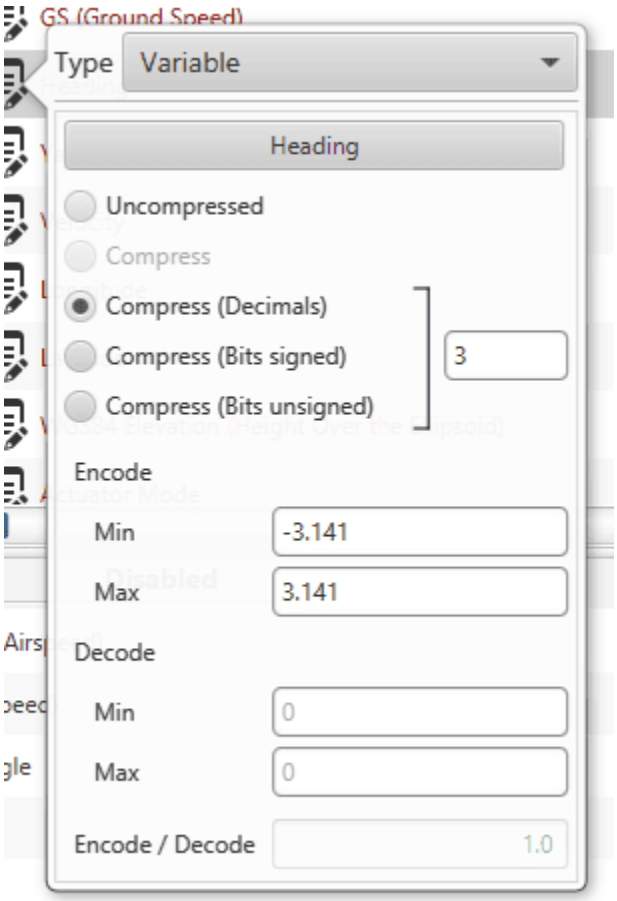


Fig. 490: **Compression example**

2.11.2 Sniffer

This panel is used to establish telemetry communication between two Autopilots 1x or between a BCS unit and an Autopilot 1x. The autopilot being configured will "listen" the variables indicated in the window **Enabled**, from another autopilot whose address is indicated in **Address**. The sniffer is commonly used to make the aircraft listen the position of the ground station and the link quality.

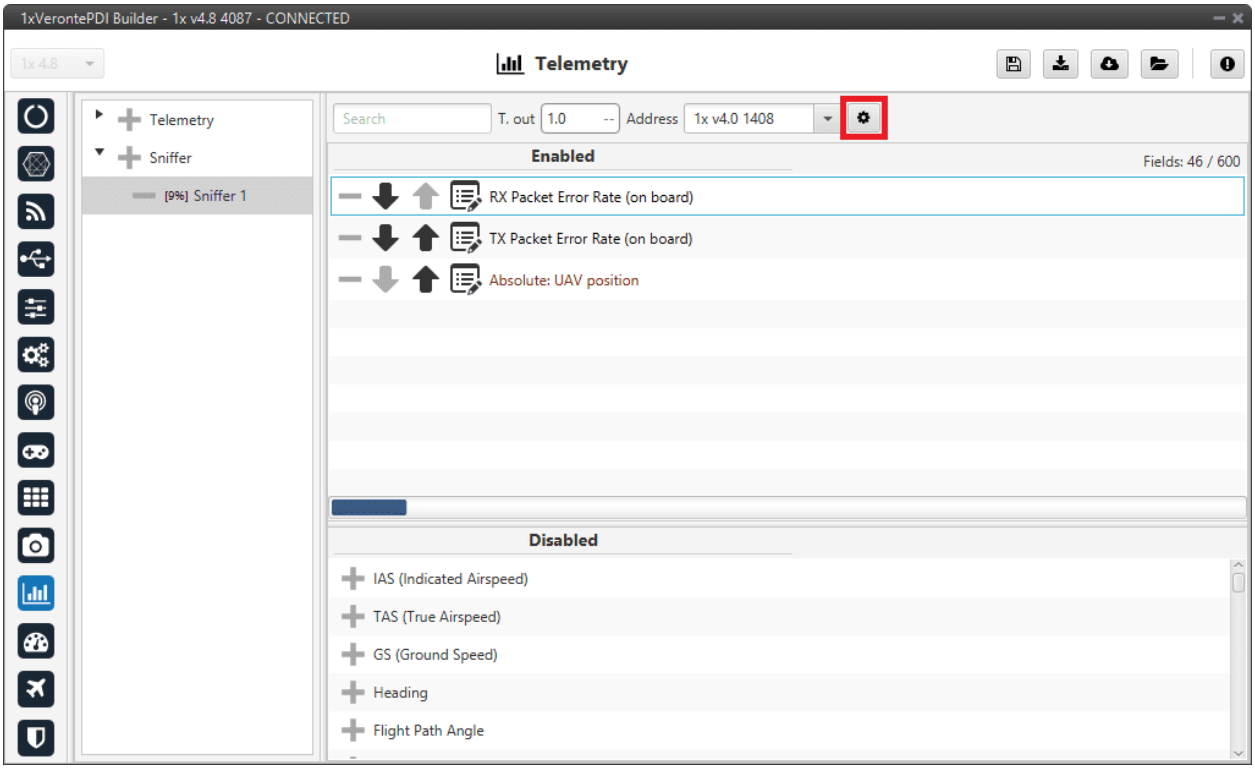


Fig. 491: **Sniffer panel**

The source UAV, in this case, is the ground station (1x v4.8 4087), which communicates to the 1x air unit (address: 1x v4.0 1408) its position and some variables related to link quality (RX and TX Packet Error Rates).

The sniffer is configured so that the air autopilot has information about the state of the communications, and it could perform an action when the link is lost. The aerial platform also receives information about the ground station position, so it can perform a mission in relation to that point.

The Autopilot 1x unit that sends the data has to be configured as well (1x ground unit), in the *Telemetry panel*. That unit will send telemetry through a **Data Link**.

By clicking on the  icon, the user can access the **Mapping Variables** configuration. Here, the variables send by the ground unit are indicated in the columm **From**, and they are stored in the variables indicated in **To** for its later use by the 1x air unit.

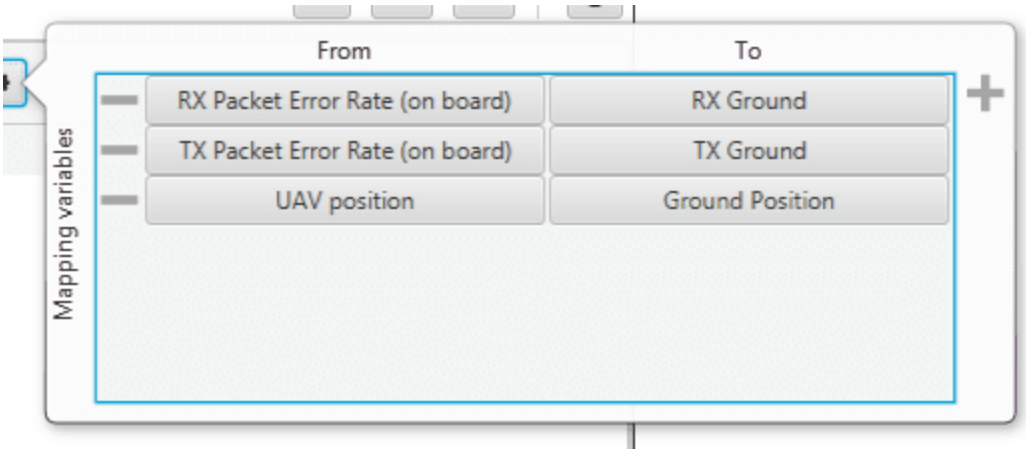


Fig. 492: **Mapping variables option**

An example of the configuration required for communication between 1x ground and air units can be found in *Data transmission between Veronte Autopilots 1x - Integration examples* section of this manual.

2.12 UI

In this menu, the user can manage operation elements and system variables.

2.12.1 Operation elements

In this panel, users have to declare (rename) operation elements they wish to use in the configuration/operation. Once declared, the value of operation elements must be set in the Operation Panel of **Veronte Ops** software. Finally, users will now be able to reference them throughout the configuration, as in *Automations menu*.

Warning: Not setting the value of these operation elements in **Veronte Ops** will result in **PDI errors** if used within the autopilot configuration, e.g., if the operation element is defined as a parameter of a **Block Program**.

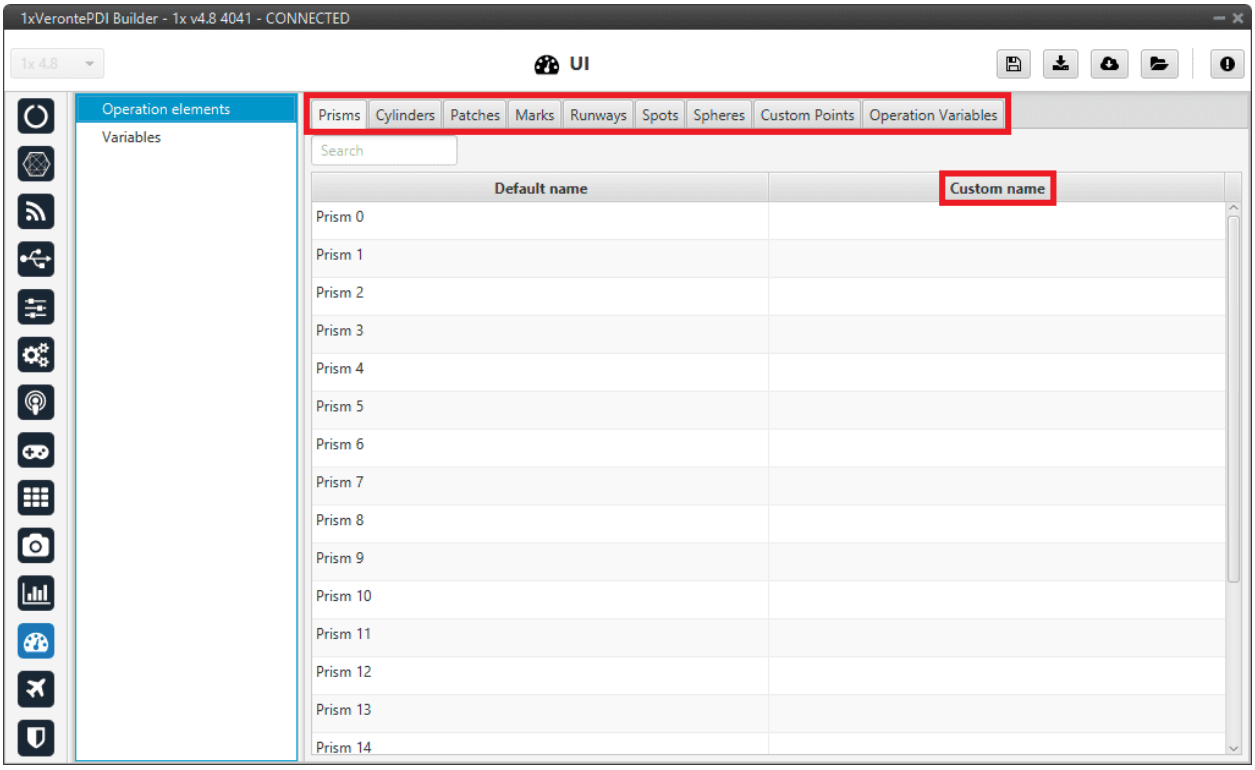


Fig. 493: **Operation elements panel**

Operation elements are divided into 9 different types:

- **Custom Points**: An operation custom point is a waypoint, a position variable (x,y,z) that can be used as a reference.
- **Patches**: A patch establishes a path that the UAV can fly to, they make up the route. Therefore patches include waypoints, segments, arcs and orbits.
- **Prisms**: A prism is a detection area.
- **Cylinders**: A cylinder is a circular prism.
- **Marks**: A mark is a reference that is placed in a patch and when the uav reaches it, an action takes place.
- **Runways**: These are the runways used during the take-off and landing phases.
- **Spots**: A spot is a kind of runway with an initial point, direction and azimuth.
- **Spheres**: A sphere is a detection area.
- **Operation Variables**: An operation guidance point is a value of the operation, such as the cruise speed.

  In addition to assigning a specific name, the user can also select the unit of this variable as well as its maximum and minimum values (in SI units).

  An example is shown below:

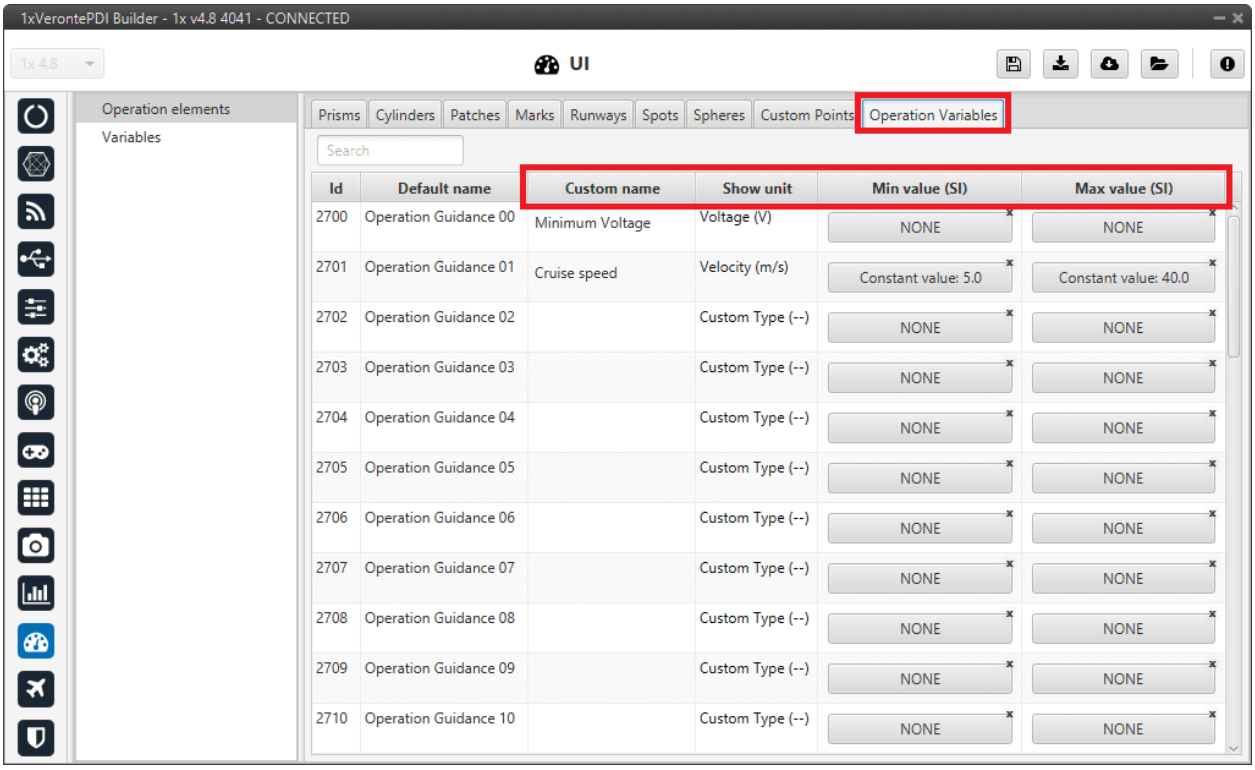


Fig. 494: **Operation elements panel - Operation Variables**

2.12.2 Variables

In this panel, the user can find the name of all system variables, as well as their units and initial values. This is very useful, for example, in the case of 'User Variables'.

Variables are divided into 4 different tabs:

- **Bits**: Bits variables, 1 bit.
- **Unsigned**: Unsigned integer variables, 16 bits.
- **Real Vars**: Real Variables, 32 bits.
- **Features**: Features variables, 64 bits.

Note:

- There are 300 User variables available for each class (Bits, Unsigned and Real Vars).
- For a detailed list of all variables available in Veronte applications, consult the List of variables section of the **1x Software Manual**.

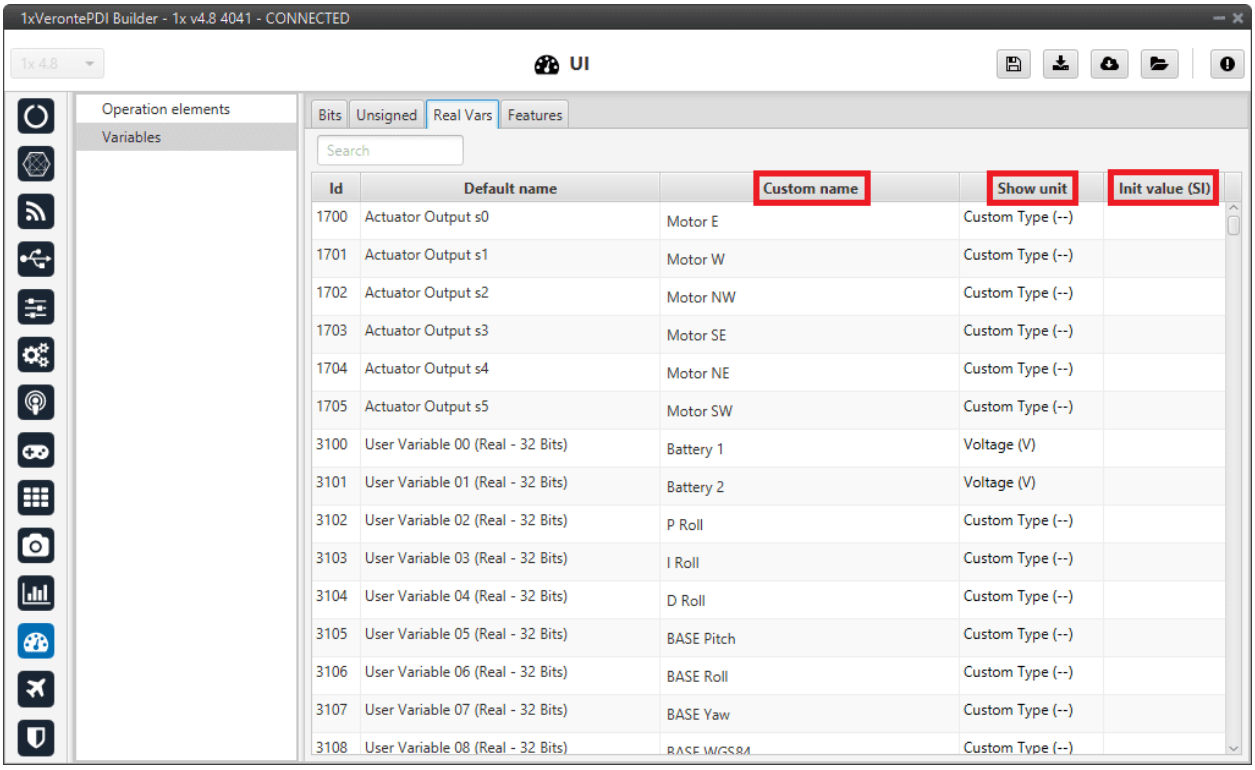


Fig. 495: **Variables panel**

To set a custom name for one of the system variables:

1. Click on the **Custom Name** cell of the desired variable and **introduce the new name** for it.
2. When the name is introduced press **Enter** to store the name on the system.
3. Press **Save** to save all changes.

Error: In this version, there is a maximum amount of characters that the user can use to rename variables, i.e. there is no limit per se but there is a maximum size of the configurable (xml size).

Besides changing their name, the user can also configure the measurement units of the variables, as well as the initial value (expressed in SI units) they will have each time the system (re)starts, using the **Show unit** and the **Init Value (SI)** cells.

By *right-clicking* on the **Show unit** cell, the user can select the desired units.

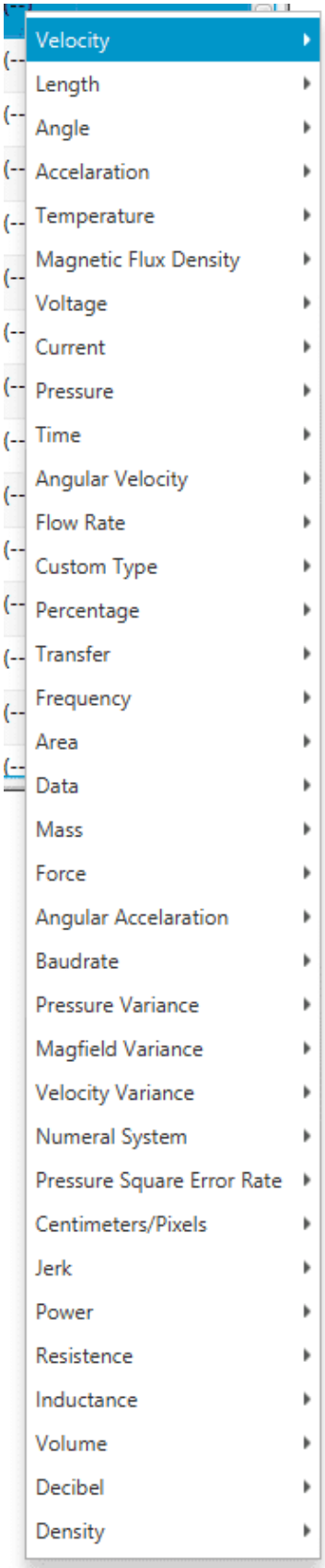


Fig. 496: **Variables units**

The table below shows all the available units in **1x PDI Builder**.

Variable Type	Units
Velocity	[m/s] [kt] [km/h] [mph] [ft/s] [mm/s] [ft/m]
Length	[m] [km] [mi] [NM] [yd] [ft] [in] [cm] [mm]
Angle	rad[$-\pi$; π] °[-180;180] °[0;360] [° ‘ “] [rad] rad[0; 2π] °
Acceleration	[m/s^2] [ft/s^2] [in/s^2] [g]
Temperature	[K] [°C] [°F]
Magnetic Flux Density	[T] [mG] [gauss] [nT]
Voltage	[V] [mV]
Current	[A] [mA]
Pressure	[Pa] [kPa] [bar] [mbar] [psi] [mmHg] [at] [atm]
Time	[s] [min] [h] [μs] [ms]
Angular Velocity	[rad/s] [rad/m] [rad/h] [rps] [rpm] [rph] [°/s]
Flow Rate	[m^3/s] [gal/s] [gal/h] [l/s] [l/h]
Custom Type	[- -]
Percentage	[x1] [%]
Transfer	[pkts/s]
Frequency	[Hz] [mHz] [kHz]
Area	[m^2] [cm^2] [mm^2] [km^2] [mile^2] [ft^2] [yd^2]
Data	[bit] [byte] [KB] [GB] [bytes/s]
Mass	[kg] [g] [tonnes] [lbs] [oz]
Force	[N] [kN] [lbf] [pdl]
Angular Acceleration	[rpm/s] [rad/s^2] [rad/m^2] [rad/h^2] [°/s^2] [°/m^2] [°/h^2]
Baudrate	[Bd] [kBd] [MBd]
Pressure Variance	[Pa^2]
Magfield Variance	[T^2]
Velocity Variance	[(m/s)^2] [(cm/s)^2] [(mm/s)^2]
Numeral System	[bin] [octal] [dec] [hex]
Pressure Square Error Rate	[Pa^2/s]
Centimeters/Pixels	[cm/pixel]
Jerk	[m/s^3]
Power	[W] [kW] [Kgm/s] [erg/s] [CV]
Resistence	[Ω]
Inductance	[H]
Volume	[m^3] [dm^3] [mm^3] [L] [mL]
Decibel	[db]
Density	[kg/m^3]

2.13 HIL

Professional Hardware In the Loop (HIL) Simulator package is a powerful tool for Autopilot 1x integration, development and operator training; allowing to extensively operate the system in a safe environment, prior to conducting real flight operations. For more information, please visit HIL Simulator user manual.

The user can link the variables on Autopilot 1x with the corresponding ones in the simulator. In this menu, simulator variables are available on the left side (**Disabled**). In addition, it can be seen 2 more panels, **To Simulator** and **To Veronte**.

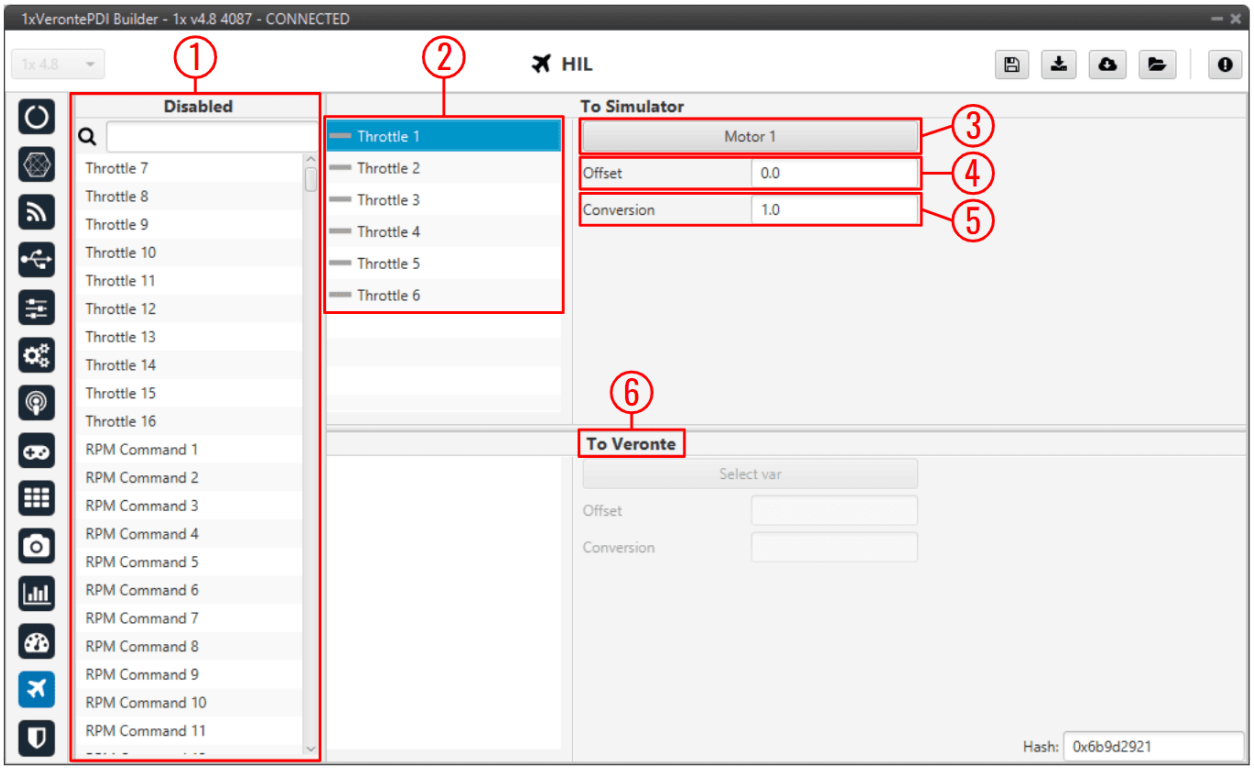


Fig. 497: **HIL menu**

In order to configure the simulation variables, users have to:

1. **Disabled**: Select the simulator variables that have been configured in the aircraft model. Just drag and drop them into "**To Simulator**" panel.
2. Here users can see all the variables selected and sent to the simulator. Select the one to be configured.

   To remove a variable from the list, simply click on the icon next to it.
3. Select the actuator variable (Control Output) of Autopilot 1x that matches with the one in the simulator. A new window will be displayed for each variable.

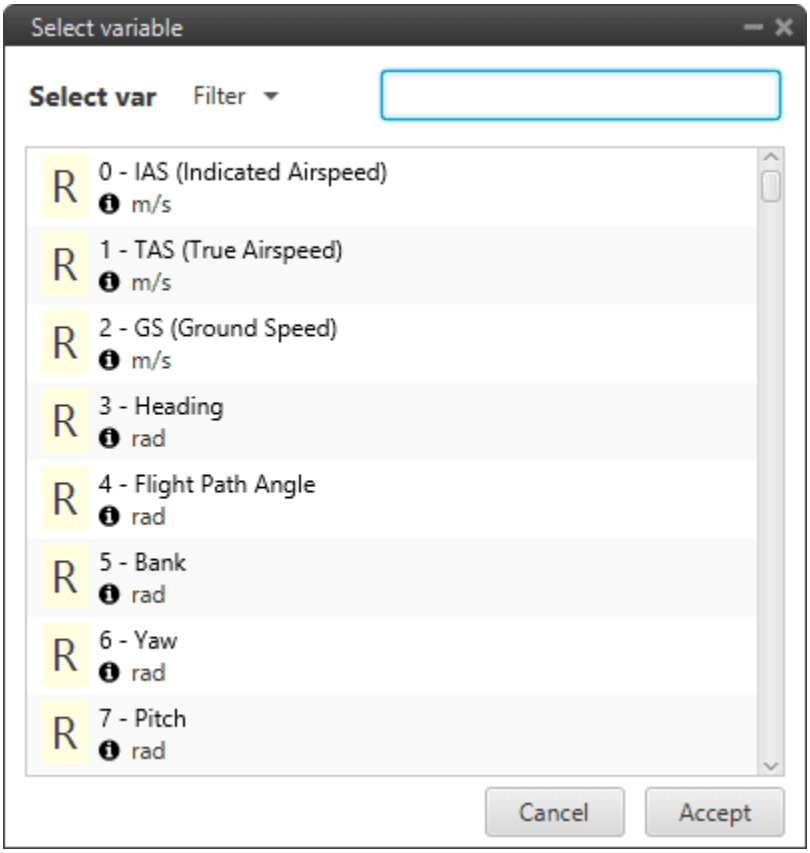


Fig. 498: **Autopilot 1x variables**

4. **Offset**: Set an offset, if it is necessary.
5. **Conversion**: Set a conversion factor, if it is necessary. It multiplies the Autopilot 1x output signal and can be used in case units on 1x and the simulator do not match. For example, in X-Plane simulator, the unit of angles is radians.

Note: To be sure of which units the simulator has, please refer to the relevant simulator manual (X-Plane, Microsoft Flight Simulator, etc.).

Warning: Always make sure that surfaces are moving in the right direction and with the correct deflection angle.

6. **To Veronte**: The user can also select variables to be sent from the simulator to Autopilot 1x. **Veronte Autopilot** can receive a **maximum of 16 simulation variables**.

Hint: An interesting variable could be the RPM of the motor.

2.13.1 Simulation variables

The following table shows the correspondence between the nomenclature of simulation variables in 1x PDI Builder, X-Plane 11 and Microsoft Flight Simulator.

Note: **1x PDI Builder** variables represent the indicated variables in this table, depending on which simulator is in use.

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Throttle 1	Throttle	25	0
Throttle 2	(commanded)	25	1
Throttle 3		25	2
Throttle 4		25	3
Throttle 5		25	4
Throttle 6		25	5
Throttle 7		25	6
Throttle 8		25	7
Throttle 9	–	–	–
Throttle 10	–	–	–
Throttle 11	–	–	–
Throttle 12	–	–	–
Throttle 13	–	–	–
Throttle 14	–	–	–
Throttle 15	–	–	–
Throttle 16	–	–	–
RPM Command 1	–	–	–
RPM Command 2	–	–	–
RPM Command 3	–	–	–
RPM Command 4	–	–	–
RPM Command 5	–	–	–
RPM Command 6	–	–	–
RPM Command 7	–	–	–
RPM Command 8	–	–	–
RPM Command 9	–	–	–
RPM Command 10	–	–	–
RPM Command 11	–	–	–
RPM Command 12	–	–	–
RPM Command 13	–	–	–
RPM Command 14	–	–	–
RPM Command 15	–	–	–
RPM Command 16	–	–	–
Governor	–	–	–
JATO	–	–	–
Jettison	–	–	–
Elevator All	Flight controls aileron/elevator/rudder	11	0
Aileron All	Flight controls aileron/elevator/rudder	11	1
Rudder All	Flight controls aileron/elevator/rudder	11	2
AileronRight 1	Aileron deflections	70	1
Aileron Right 2	1	70	3

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Aileron Right 3		70	5
Aileron Right 4		70	7
Aileron Right 5	Aileron deflections	71	1
Aileron Right 6	2	71	3
Aileron Right 7		71	5
Aileron Right 8		71	7
Aileron Left 1	Aileron deflections	70	0
Aileron Left 2	1	70	2
Aileron Left 3		70	4
Aileron Left 4		70	6
Aileron Left 5	Aileron deflections	71	0
Aileron Left 6	2	71	2
Aileron Left 7		71	4
Aileron Left 8		71	6
Elevator 1	Elevator deflections	74	0
Elevator 2		74	1
Elevator 3		74	2
Elevator 4		74	3
Rudder 1	Rudder deflections	75	0
Rudder 2		75	1
Rudder 3		75	2
Rudder 4		75	3
Flap All	Trim, flap, stats, & speedbrakes	13	4
Flap Right 1	–	–	–
Flap Right 2	–	–	–
Flap Left 1	–	–	–
Flap Left 2	–	–	–
Flap Handle	Trim, flap, stats, & speedbrakes	13	3
Landing Gear	Gear & brakes	14	0
Landing Gear Steering 0	Landing gear	134	0
Landing Gear Steering 1	steering	134	1
Landing Gear Steering 2		134	2
Landing Gear Steering 3		134	3
Landing Gear Steering 4		134	4
Landing Gear Steering 5		134	5
Landing Gear Steering 6		134	6
Landing Gear Steering 7		134	7
Brake All	Gear & brakes	14	1
Brake Right	Gear & brakes	14	3
Brake Left	Gear & brakes	14	2
Speed Brake 1	Trim, flap, stats, & speedbrakes	13	6
Speed Brake 2	Trim, flap, stats, & speedbrakes	13	7
Roll Cyclic 1	Roll cyclic disc tilts	81	0
Roll Cyclic 2	Roll cyclic disc tilts	81	1
Roll Cyclic 3	–	–	–
Roll Cyclic 4	–	–	–
Roll Cyclic 5	–	–	–
Roll Cyclic 6	–	–	–
Roll Cyclic 7	–	–	–

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Roll Cyclic 8	–	–	–
Roll Cyclic 9	–	–	–
Roll Cyclic 10	–	–	–
Roll Cyclic 11	–	–	–
Roll Cyclic 12	–	–	–
Roll Cyclic 13	–	–	–
Roll Cyclic 14	–	–	–
Roll Cyclic 15	–	–	–
Roll Cyclic 16	–	–	–
Pitch Cyclic 1	Pitch cyclic disc tilts	80	0
Pitch Cyclic 2	–	80	1
Pitch Cyclic 3	–	–	–
Pitch Cyclic 4	–	–	–
Pitch Cyclic 5	–	–	–
Pitch Cyclic 6	–	–	–
Pitch Cyclic 7	–	–	–
Pitch Cyclic 8	–	–	–
Pitch Cyclic 9	–	–	–
Pitch Cyclic 10	–	–	–
Pitch Cyclic 11	–	–	–
Pitch Cyclic 12	–	–	–
Pitch Cyclic 13	–	–	–
Pitch Cyclic 14	–	–	–
Pitch Cyclic 15	–	–	–
Pitch Cyclic 16	–	–	–
Pitch Cyclic 16	–	–	–
Clutch 0	Clutch & artificial stability switches	111	0
Clutch 1		111	1
Clutch 2		111	2
Joystick Alieron	Joystick aileron/elevator/rudder	111	0
Joystick Elevator		111	1
Joystick Rudder		111	2
Collective 1	Propeller pitch	39	0
Collective 2		39	1
Collective 3		39	2
Collective 4		39	3
Collective 5		39	4
Collective 6		39	5
Collective 7		39	6
Collective 8		39	7
Collective 9	–	–	–
Collective 10	–	–	–
Collective 11	–	–	–
Collective 12	–	–	–
Collective 13	–	–	–
Collective 14	–	–	–
Collective 15	–	–	–
Collective 16	–	–	–
Motors Tilt Pitch 1	Wing sweep & thrust vectoring	12	3

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Motors Tilt Pitch 2	–	–	–
Motors Tilt Pitch 3	–	–	–
Motors Tilt Pitch 4	–	–	–
Motors Tilt Pitch 5	–	–	–
Motors Tilt Pitch 6	–	–	–
Motors Tilt Pitch 7	–	–	–
Motors Tilt Pitch 8	–	–	–
Motors Tilt Pitch 9	–	–	–
Motors Tilt Pitch 10	–	–	–
Motors Tilt Pitch 11	–	–	–
Motors Tilt Pitch 12	–	–	–
Motors Tilt Pitch 13	–	–	–
Motors Tilt Pitch 14	–	–	–
Motors Tilt Pitch 15	–	–	–
Motors Tilt Pitch 16	–	–	–
Motors Tilt Roll 1	–	–	–
Motors Tilt Roll 2	–	–	–
Motors Tilt Roll 3	–	–	–
Motors Tilt Roll 4	–	–	–
Motors Tilt Roll 5	–	–	–
Motors Tilt Roll 6	–	–	–
Motors Tilt Roll 7	–	–	–
Motors Tilt Roll 8	–	–	–
Motors Tilt Roll 9	–	–	–
Motors Tilt Roll 10	–	–	–
Motors Tilt Roll 11	–	–	–
Motors Tilt Roll 12	–	–	–
Motors Tilt Roll 13	–	–	–
Motors Tilt Roll 14	–	–	–
Motors Tilt Roll 15	–	–	–
Motors Tilt Roll 16	–	–	–
Wing Tilt 1	Wing sweep & thrust vectoring	12	5
Wing Tilt 2	–	–	–
Wing Tilt 3	–	–	–
Wing Tilt 4	–	–	–
Wing Tilt 5	–	–	–
Wing Tilt 6	–	–	–
Wing Tilt 7	–	–	–
Wing Tilt 8	–	–	–
Wing Dihedral 1	Wing sweep & thrust vectoring	12	6
Wing Dihedral 2	–	–	–
Wing Dihedral 3	–	–	–
Wing Dihedral 4	–	–	–
Wing Dihedral 5	–	–	–
Wing Dihedral 6	–	–	–
Wing Dihedral 7	–	–	–
Wing Dihedral 8	–	–	–
Wing Sweep 1	Wing sweep & thrust vectoring	12	4

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Wing Sweep 2	–	–	–
Wing Sweep 3	–	–	–
Wing Sweep 4	–	–	–
Wing Sweep 5	–	–	–
Wing Sweep 6	–	–	–
Wing Sweep 7	–	–	–
Wing Sweep 8	–	–	–
Parachute	Miscellaneous switches	112	7
Throttle Actual 1	Throttle (actual)	26	0
Throttle Actual 2		26	1
Throttle Actual 3		26	2
Throttle Actual 4		26	3
Throttle Actual 5		26	4
Throttle Actual 6		26	5
Throttle Actual 7		26	6
Throttle Actual 8		26	7
Throttle Actual 9	–	–	–
Throttle Actual 10	–	–	–
Throttle Actual 11	–	–	–
Throttle Actual 12	–	–	–
Throttle Actual 13	–	–	–
Throttle Actual 14	–	–	–
Throttle Actual 15	–	–	–
Throttle Actual 16	–	–	–
Motor RPM 1	Engine RPM	37	0
Motor RPM 2		37	1
Motor RPM 3		37	2
Motor RPM 4		37	3
Motor RPM 5		37	4
Motor RPM 6		37	5
Motor RPM 7		37	6
Motor RPM 8		37	7
Motor RPM 9	–	–	–
Motor RPM 10	–	–	–
Motor RPM 11	–	–	–
Motor RPM 12	–	–	–
Motor RPM 13	–	–	–
Motor RPM 14	–	–	–
Motor RPM 15	–	–	–
Motor RPM 16	–	–	–
Rotor RPM 1	Propeller RPM	38	0
Rotor RPM 2		38	1
Rotor RPM 3		38	2
Rotor RPM 4		38	3
Rotor RPM 5		38	4
Rotor RPM 6		38	5
Rotor RPM 7		38	6
Rotor RPM 8		38	7
Rotor RPM 9	–	–	–

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Rotor RPM 10	–	–	–
Rotor RPM 11	–	–	–
Rotor RPM 12	–	–	–
Rotor RPM 13	–	–	–
Rotor RPM 14	–	–	–
Rotor RPM 15	–	–	–
Rotor RPM 16	–	–	–
Power 1	Engine power	34	0
Power 2		34	1
Power 3		34	2
Power 4		34	3
Power 5		34	4
Power 6		34	5
Power 7		34	6
Power 8		34	7
Power 9	–	–	–
Power 10	–	–	–
Power 11	–	–	–
Power 12	–	–	–
Power 13	–	–	–
Power 14	–	–	–
Power 15	–	–	–
Power 16	–	–	–
EGT (Exhaust Gas Temperature) 1	Exhaust has temperature (EGT)	47	0
EGT (Exhaust Gas Temperature) 2	–	–	–
EGT (Exhaust Gas Temperature) 3	–	–	–
EGT (Exhaust Gas Temperature) 4	–	–	–
EGT (Exhaust Gas Temperature) 5	–	–	–
EGT (Exhaust Gas Temperature) 6	–	–	–
EGT (Exhaust Gas Temperature) 7	–	–	–
EGT (Exhaust Gas Temperature) 8	–	–	–
EGT (Exhaust Gas Temperature) 9	–	–	–
EGT (Exhaust Gas Temperature) 10	–	–	–
EGT (Exhaust Gas Temperature) 11	–	–	–
EGT (Exhaust Gas Temperature) 12	–	–	–
EGT (Exhaust Gas Temperature) 13	–	–	–
EGT (Exhaust Gas Temperature) 14	–	–	–
EGT (Exhaust Gas Temperature) 15	–	–	–
EGT (Exhaust Gas Temperature) 16	–	–	–
Oil Pressure 1	Oil pressure	49	0
Oil Pressure 2		49	1
Oil Pressure 3		49	1
Oil Pressure 4		49	2
Oil Pressure 5		49	2
Oil Pressure 6		49	3
Oil Pressure 7		49	3
Oil Pressure 8		49	4
Oil Pressure 9	–	–	–
Oil Pressure 10	–	–	–

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Oil Pressure 11	–	–	–
Oil Pressure 12	–	–	–
Oil Pressure 13	–	–	–
Oil Pressure 14	–	–	–
Oil Pressure 15	–	–	–
Oil Pressure 16	–	–	–
IAT (Inlet Air Temperature) 1	–	–	–
IAT (Inlet Air Temperature) 2	–	–	–
IAT (Inlet Air Temperature) 3	–	–	–
IAT (Inlet Air Temperature) 4	–	–	–
IAT (Inlet Air Temperature) 5	–	–	–
IAT (Inlet Air Temperature) 6	–	–	–
IAT (Inlet Air Temperature) 7	–	–	–
IAT (Inlet Air Temperature) 8	–	–	–
IAT (Inlet Air Temperature) 9	–	–	–
IAT (Inlet Air Temperature) 10	–	–	–
IAT (Inlet Air Temperature) 11	–	–	–
IAT (Inlet Air Temperature) 12	–	–	–
IAT (Inlet Air Temperature) 13	–	–	–
IAT (Inlet Air Temperature) 14	–	–	–
IAT (Inlet Air Temperature) 15	–	–	–
IAT (Inlet Air Temperature) 16	–	–	–
Oil Temperature 1	Oil temperature	50	0
Oil Temperature 2		50	1
Oil Temperature 3		50	2
Oil Temperature 4		50	3
Oil Temperature 5		50	4
Oil Temperature 6		50	5
Oil Temperature 7		50	6
Oil Temperature 8		50	7
Oil Temperature 9	–	–	–
Oil Temperature 10	–	–	–
Oil Temperature 11	–	–	–
Oil Temperature 12	–	–	–
Oil Temperature 13	–	–	–
Oil Temperature 14	–	–	–
Oil Temperature 15	–	–	–
Oil Temperature 16	–	–	–
Coolant Tempreature 1	–	–	–
Coolant Tempreature 2	–	–	–
Coolant Tempreature 3	–	–	–
Coolant Tempreature 4	–	–	–
Coolant Tempreature 5	–	–	–
Coolant Tempreature 6	–	–	–
Coolant Tempreature 7	–	–	–
Coolant Tempreature 8	–	–	–
Coolant Tempreature 9	–	–	–
Coolant Tempreature 10	–	–	–
Coolant Tempreature 11	–	–	–

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
Coolant Tempreature 12	–	–	–
Coolant Tempreature 13	–	–	–
Coolant Tempreature 14	–	–	–
Coolant Tempreature 15	–	–	–
Coolant Tempreature 16	–	–	–
Fuel mixture 1	Mixture setting	29	0
Fuel mixture 2	–	–	–
Fuel mixture 3	–	–	–
Fuel mixture 4	–	–	–
Fuel mixture 5	–	–	–
Fuel mixture 6	–	–	–
Fuel mixture 7	–	–	–
Fuel mixture 8	–	–	–
Fuel mixture 9	–	–	–
Fuel mixture 10	–	–	–
Fuel mixture 11	–	–	–
Fuel mixture 12	–	–	–
Fuel mixture 13	–	–	–
Fuel mixture 14	–	–	–
Fuel mixture 15	–	–	–
Fuel mixture 16	–	–	–
Total Fuel	Payload weights & center of gravity (CG)	63	2
Total Fuel Flow	–	–	–
Fuel 1	Fuel weights	62	0
Fuel 2		62	1
Fuel 3		62	2
Fuel 4		62	3
Fuel 5		62	4
Fuel 6		62	5
Fuel 7		62	6
Fuel 8		62	7
Fuel Flow 1	Fuel flow (FF)	45	0
Fuel Flow 2		45	1
Fuel Flow 3		45	2
Fuel Flow 4		45	3
Fuel Flow 5		45	4
Fuel Flow 6		45	5
Fuel Flow 7		45	6
Fuel Flow 8		45	7
visRatio	Frame rate	0	3
grndRatio		0	5
flitRatio		0	6
presBar	System pressures	7	0
eas	Speeds	3	1
tas		3	2
gs		3	3
Speed	Autopilot values	118	0
Heading		118	1
VVI	Mach, VVI, g-load	4	2

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
distNm	Location, velocity, & distance traveled	21	7
magComp	Magnetic compass	19	0
magvar	–	–	–
	–	–	–
AMpress	Aircraft atmosphere	6	0
AMtemperature		6	1
LEtemperature		6	2
dens		6	3
A		6	4
Q		6	5
Position IAS	Speeds	3	0
Position Latitude	Latitude, longitude,	20	0
Position Longitude	& altitude	20	1
Position altitude Msl		20	2
Position altitude Agl		20	3
Position altitude Ind		20	5
X	Location, velocity,	21	0
Y	& distance traveled	21	1
Z		21	2
vX		21	3
vY		21	4
vZ		21	5
Position gNormal	Mach, VVI, g-load	4	4
Position gAxial		4	5
Position gSide		4	6
Orientation Pitch	Pitch, roll, &	17	0
Orientation Roll	headings	17	1
Orientation Heading True		17	2
Heading Mag		17	3
Orientation Pitch rate	Angular velocities	16	0
Orientation Roll Rate		16	1
Orientation Yaw Rate		16	2
Orientation alpha	Angle of attack, sideslip, & paths	18	0
Orientation beta	Angle of attack, sideslip, & paths	18	1
thrust1	Engine thrust	35	0
thrust2		35	1
torque1	Engine torque	36	0
torque2		36	1
fdirMode	Autopilot, flight director, & HUD switches	108	1
navArm	Autopilot armed	116	0
altArm	status	116	1
appArm		116	2
autoThrottle	Autopilot values	117	0
modeHeading		117	1
modeAlt		117	2
cgControl	Payload weights & center of gravity (CG)	63	7
realTime	Times	1	0
totalTime		1	1
missnTime		1	2

Table 2 – continued from previous page

1x PDI Builder	X-Plane 11		
	Variable name	Index	Subindex
timer		1	3
diAlt	Autopilot values	118	3
useAlt	–	–	–
empty	Payload weights & center of gravity (CG)	63	0
payload		63	1
fuel		63	2
jetti		63	3
current		63	4
maximum		63	5
cg		63	7
Speed North	–	–	–
Speed East	–	–	–
Speed Down	–	–	–

2.14 Safety

In this menu the user can create checklists for each phase, avoid changing certain parameters, settings or programs and define safety bits lists.

2.14.1 Checklist

This feature is used to make sure that some requirements have been accomplished, for example, prior to a phase change or to avoid a possible malfunction.

These checklists will appear in a panel called **Checklist** of **Veronte Ops** (for more information about this, visit Veronte Ops manual).

Note: There are 3 different types of checks:

- Checks that are performed automatically by Veronte Autopilot 1x, such as "**In Range check**".
- Checks that need a command to Autopilot 1x, e.g. "**Calibrate Atmosphere**".
- Checks for operator information only, which are performed with type "**None**".

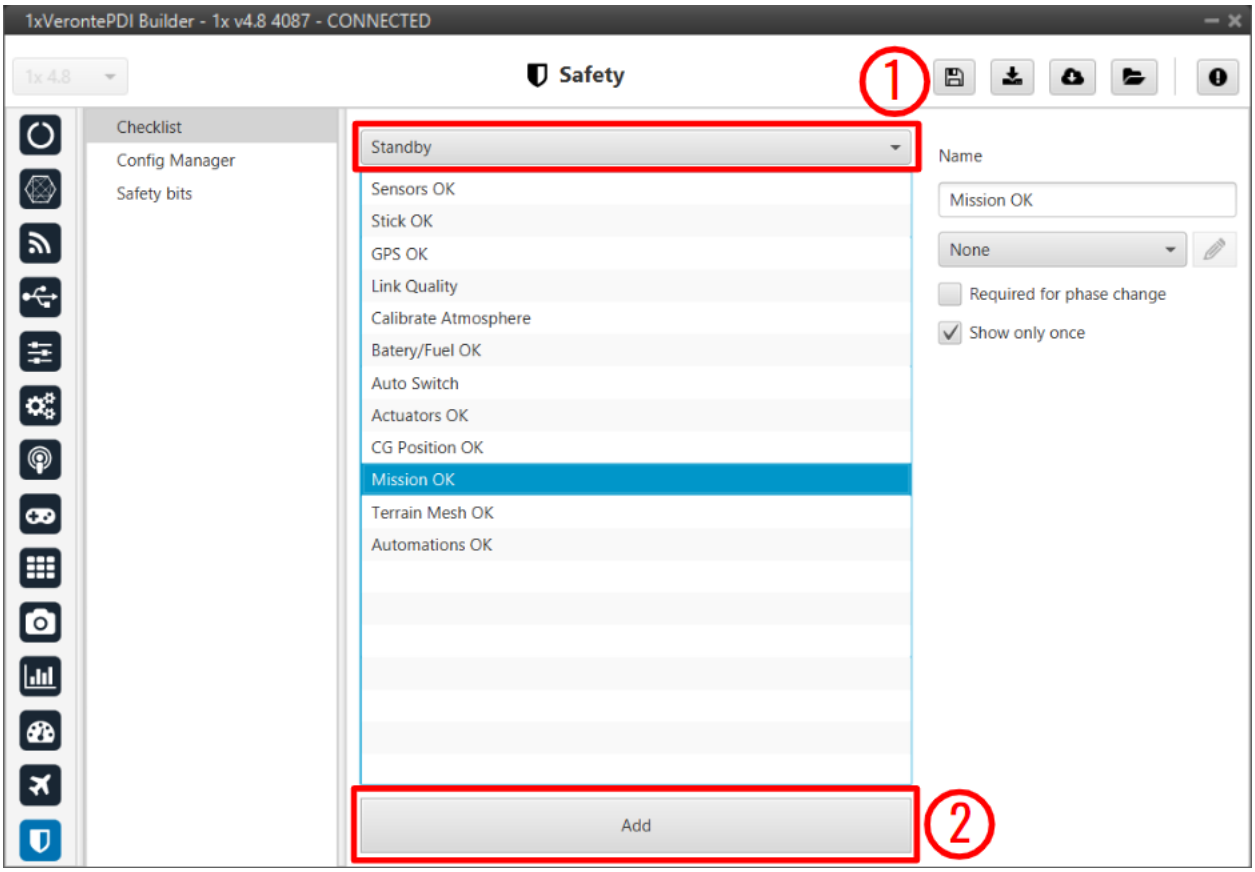


Fig. 499: **Checklist panel**

In (1), the user will find all the phases configured for the operation. In each one of them, new elements for the checklist can be added with the button **Add** (2). The user can modify the checklist order of the phase by selecting and dragging elements in the list to the desired position.

The configurable parameters for each element are:

- **Name**: The name that will identify the element.
- **Type**: The element chosen from the checklist can be one of the following types:
  - **Calibrate Atmosphere**: The user can request the calibration of the atmosphere model.
  - **Calibrate DEM**: The user can request the calibration of the DEM.
  - **Command Position**: Send to the UAV a position.
  - **Command Yaw**: Send to the UAV a yaw angle.
  - **Enter Wind Information**: Enter initial values for wind state to the UAV.
  - **In Range Check**: Allows checking if a variable is between the range selected.

> **Error:** In order for **Veronte Ops** to correctly execute this functionality, variables that are added to a checklist as **In Range Check** must also be added to the mandatory telemetry vector *Data vectors*.
>
> This is because if they are not added to this telemetry vector, these variables are initialized in **Veronte Ops** with a value of **0**, which in reality may not be the actual value of these variables and may cause

confusion in some range.

  - **None**: Any action is performed, been just a check for the user to do something external.
  - **Trim arcade**: The user can request the stick calibration for arcade commands.
- **Required for phase change**: If **enabled**, the element must be checked to switch to another phase.
- **Show only once**: If **enabled**, the check will only appear the first time its phase is executed.
- **Automatic check**: This option is only available when **'In Range check'** is selected.

An example of '**In Range Check**' can be shown below:

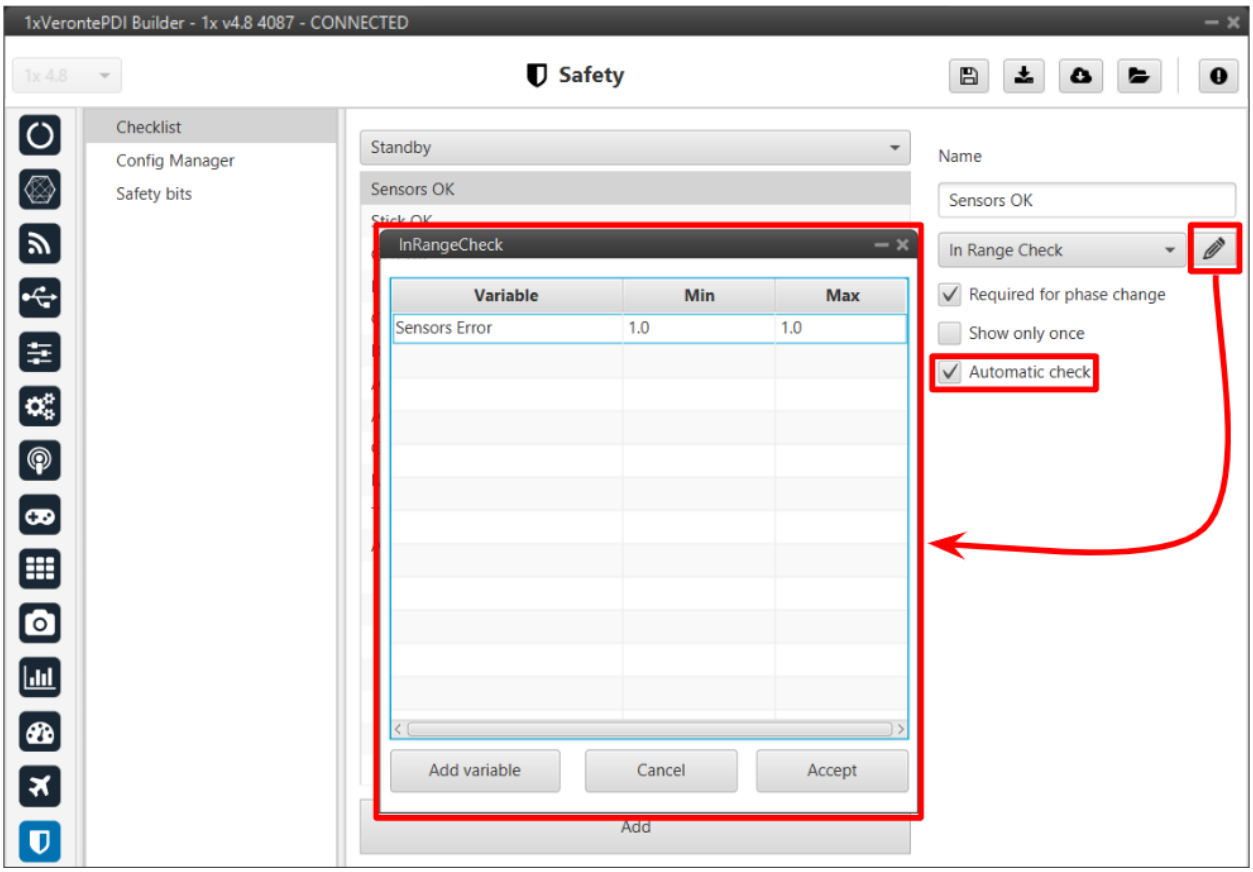


Fig. 500: **Checklist example**

2.14.2 Config Manager

Config Manager avoids the user changing certains parameters, settings or programs of Autopilot 1x. It is shown in the picture below:

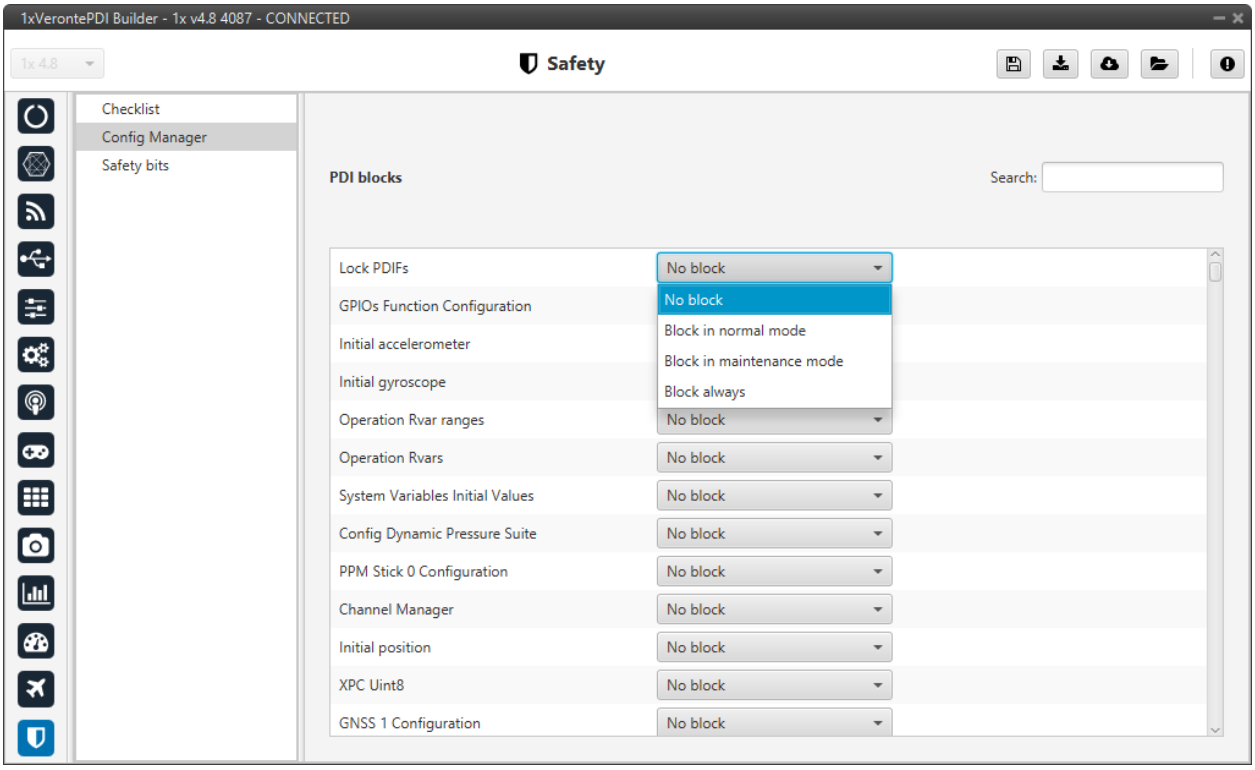


Fig. 501: **Config Manager panel**

The user can choose between:

- No block
- Block in normal mode
- Block in maintenance mode
- Block always

2.14.3 Safety bits

In this panel the user can configure 3 different safety bits lists.

The bits included in these lists are added to the set of default system bits that trigger the **System error** variable, and therefore trigger the **FTS**. The user can refer to this list of default system bits in the Activation System Error bits section of the **1x Software Manual**.

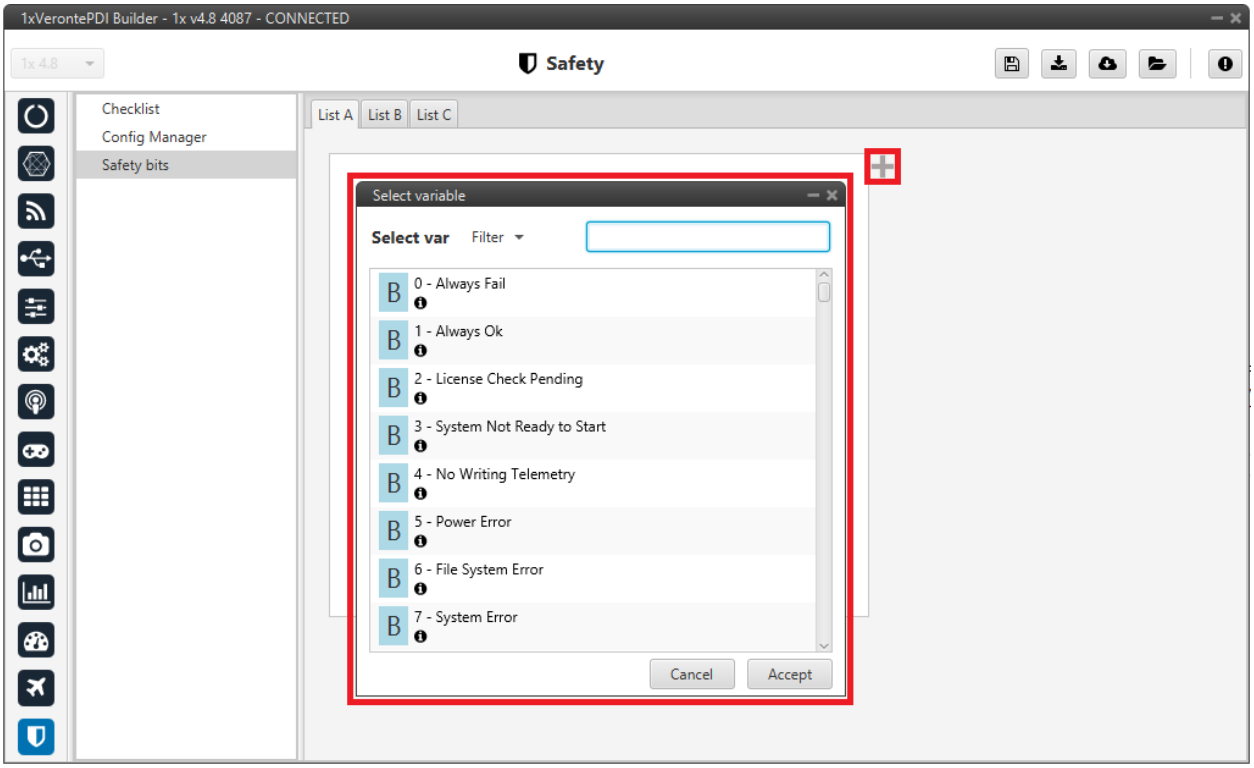


Fig. 502: **Safety bits panel**

By default, there is no bit defined in any Safety bits list. To add them, just press + icon and select the desired bits. A common user bit to add to these lists is the '**Sensors Error' bit**, so that if one of the sensors fails, the FTS is triggered.

In addition, the user can switch between the different lists with an action. For more information, see *Actions - Automations* section of this manual.

CHAPTER

THREE

INTEGRATION EXAMPLES

In this section, a series of examples will be presented so that the user knows how to perform **certain customizations in the 1x PDI Builder**. In addition, some examples of integration between the **Autopilot 1x and external devices** are presented.

3.1 AP communication with PC

Since Veronte Autopilot 1x can be connected to a computer via a USB or serial interface, the configuration for both connections is already set by default in **1x PDI Builder**.

However, users should check that this configuration has not been modified to ensure a correct communication via both ways in case one of them is lost. For this:

Go to Input/Output menu → **I/O Setup panel**. Each **USB, RS232 and RS485 Producers must be bidirectionally connected to a Commgr port**:

Important: Users should also check that the Commgr ports to which USB and serial ports are connected are **not routed**. For more information on Routing, see *Ports - Communications* section of this manual.

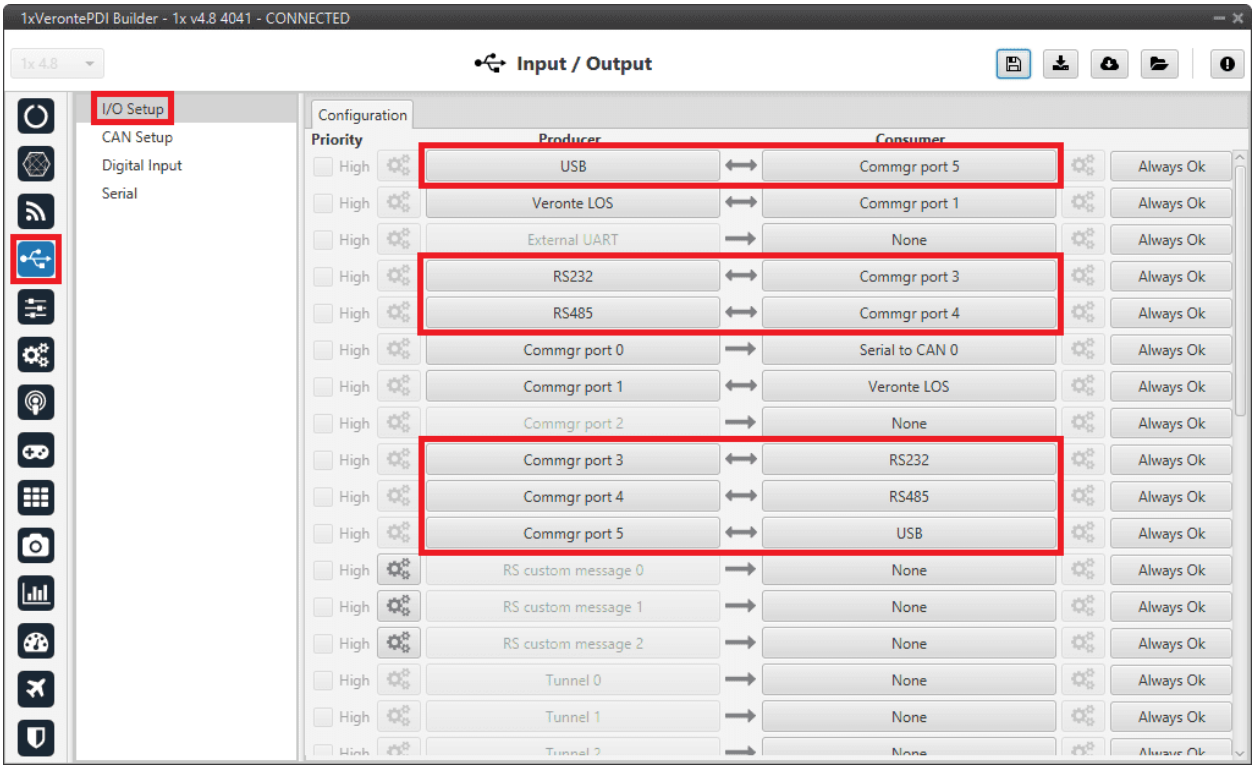


Fig. 1: **USB/RS232/RS485 ↔ Commgr port**

3.2 ArcTrim Button

The ArcTrim button allows the user to trim the stick signal directly from the stick position, before the operation, by simply clicking on it. In addition, this button is considered as an '**action button**' that can be embedded **in the Veronte Panel**.

To do this, the following steps should be followed:

1. Go to **Block Programs menu**.
    - Create a program to make the necessary connection to the *Arc Trim block*.

      Usually the user has a **Stick program** where the blocks that are related to the stick are implemented.
    - Add the *Arc Trim block* and connect the input and output variables to it.

      Usually the **input** variables are **Stick Input u0-u3** and the **output** variables **Stick Input d0-d3**.
    - Finally, **enable the block to be commandeded** by simply clicking on the  icon.

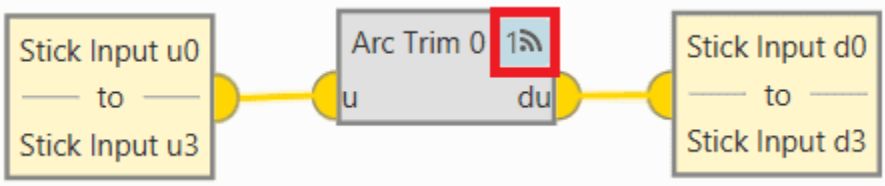


Fig. 2: **ArcTrim Button - Block**

2. **Configure the trim vector** of the *Arc Trim block*.

   Depending on the range of the signal, the following values are recommended:

   - If the signal ranges from **0 to 1** ⇒ **0.5**.
   - If the signal ranges from **-1 to 1** ⇒ **0**.

   In this example, since the signal is in the range 0-1, 0.5 is set:

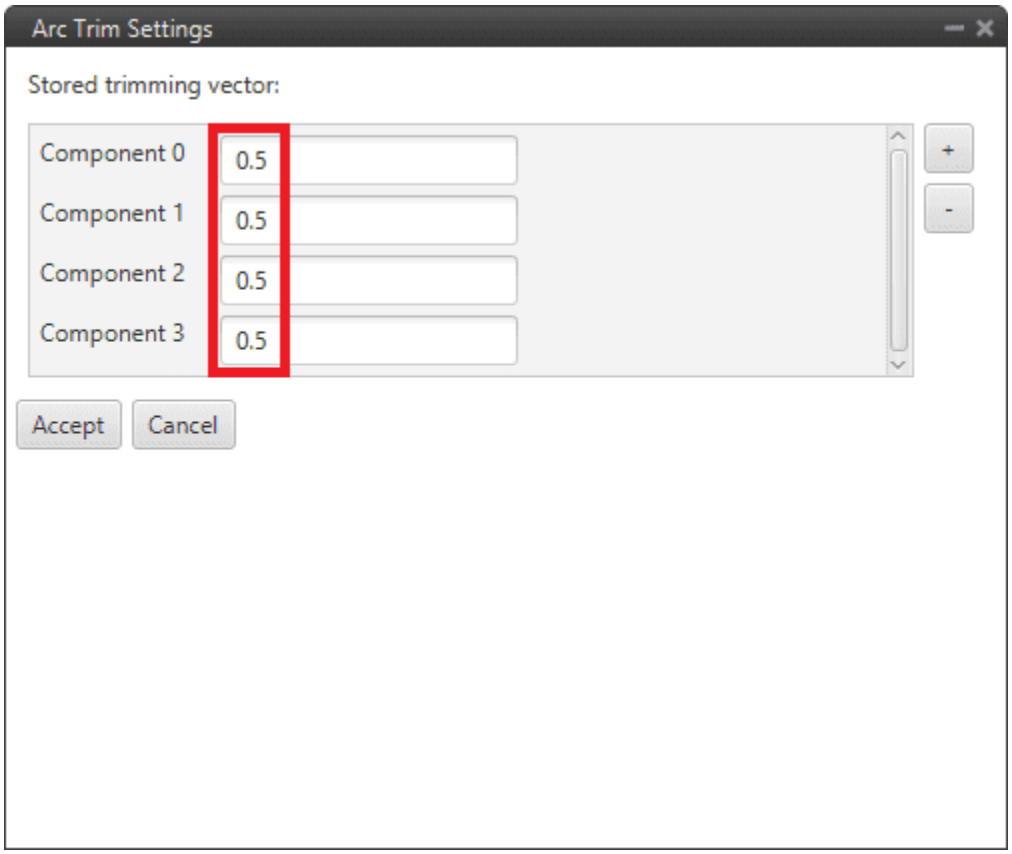


Fig. 3: **ArcTrim Button - Block configuration**

3. Go to **Automations menu** → create a **New automation** → go to **Events**.

   Select the **Button** option and choose the desired icon for this button.

   In addition, it is recommended to activate the **Confirmation** checkbox, to avoid trimming the stick by mistake.

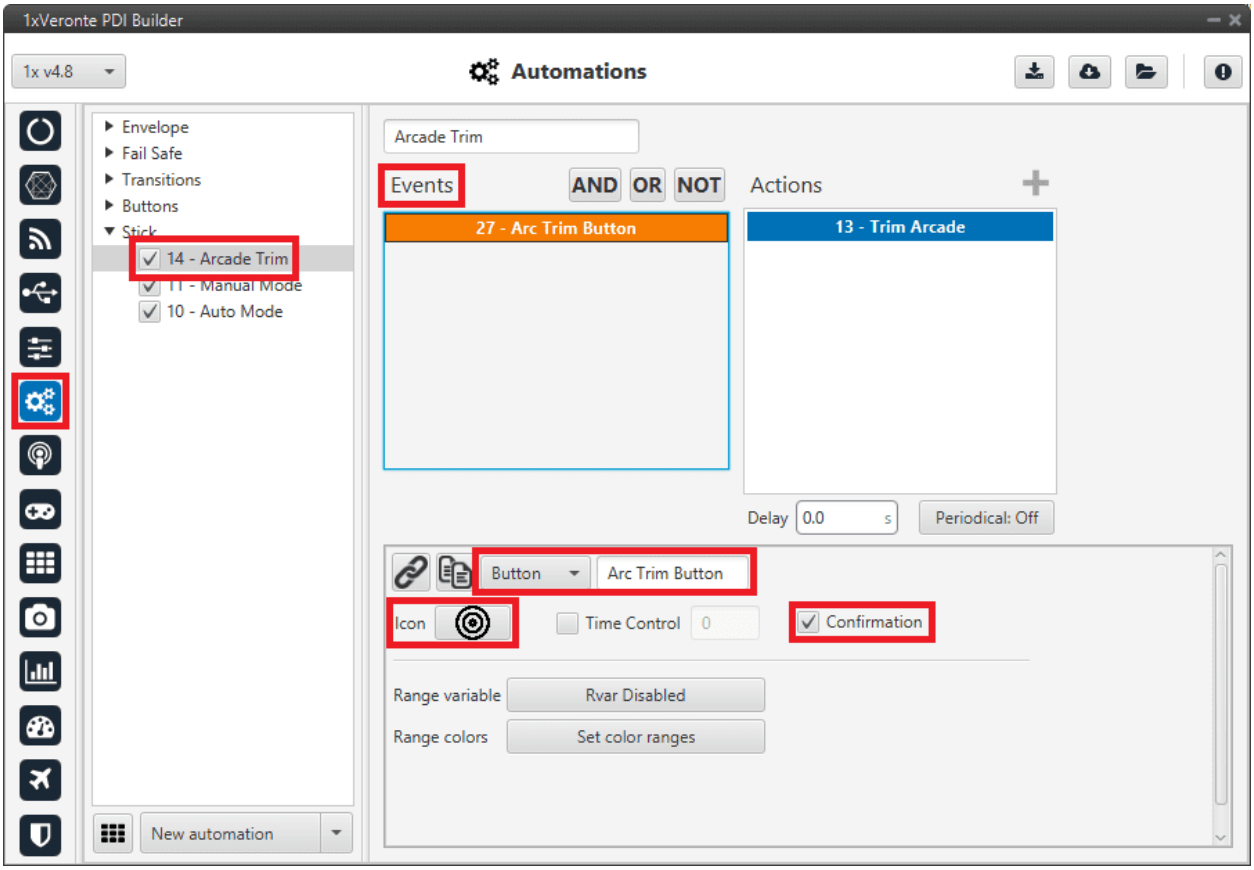


Fig. 4: **ArcTrim Button - Events**

4. In the created automation, go to **Actions**.
    - Add the **Command block** action.
    - Select **ArcTrim** block to command and choose the commandable **Id**.
    - Finally, it is recommended to activate both checkboxes:

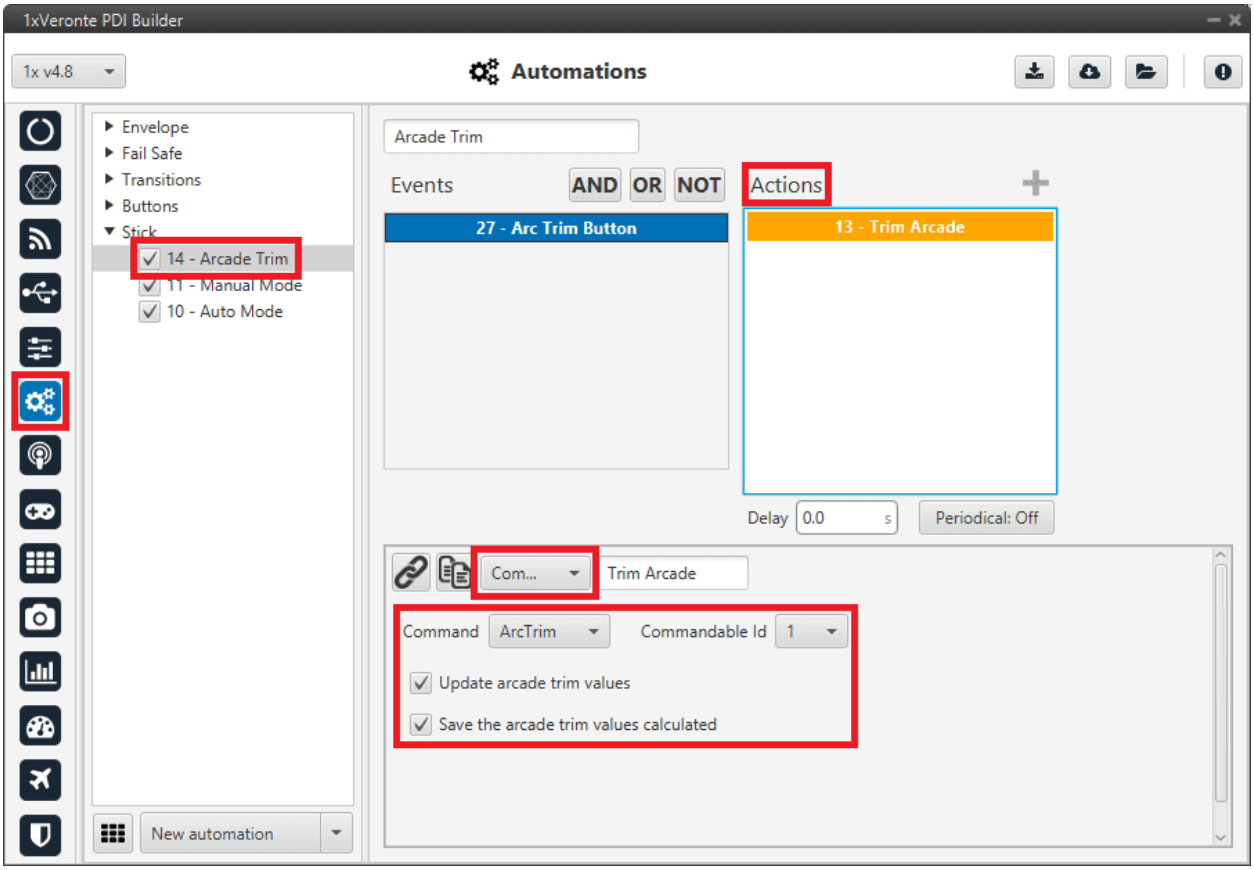


Fig. 5: **ArcTrim Button - Actions**

5. In **Veronte Ops**, this button will appear embedded in the **Veronte Panel**.

Note: This action button will only appear on the Veronte Panel if the action buttons have been enabled to be shown on it. For more information on this, see Veronte Panel - Main widgets section on the **Veronte Ops** user manual.

Fig. 6: **ArcTrim Button - Veronte Panel**

When clicking on it, the following confirmation message will be displayed (as the confirmation checkbox has been activated in the automation):

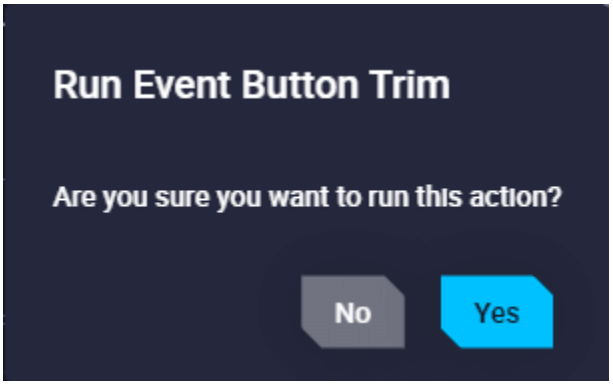


Fig. 7: **ArcTrim Button - Confirmation message**

Now, the stick is trimmed.

3.3 CAN communication

Here are described the steps to be followed in order to correctly receive and transmit **CAN messages**, both via CAN and serial interfaces.

3.3.1 CAN messages transmission

This section summarizes the configuration to be carried out **to send CAN streams over a CAN Bus**.

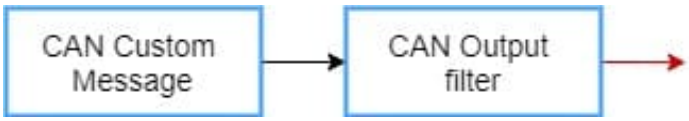


Fig. 8: **CAN messages transmission diagram**

1. Go to Input/Output menu → CAN Setup panel → **Custom message 0 tab**.

   Select the fields to send in **TX** or TX Ini, as it is a Producer. More information on the configuration of CAN messages can be found in the *TX/TX Ini Messages (Custom Messages) - Input/Output* section of this manual.

   For example, a CAN message set to ID 12:

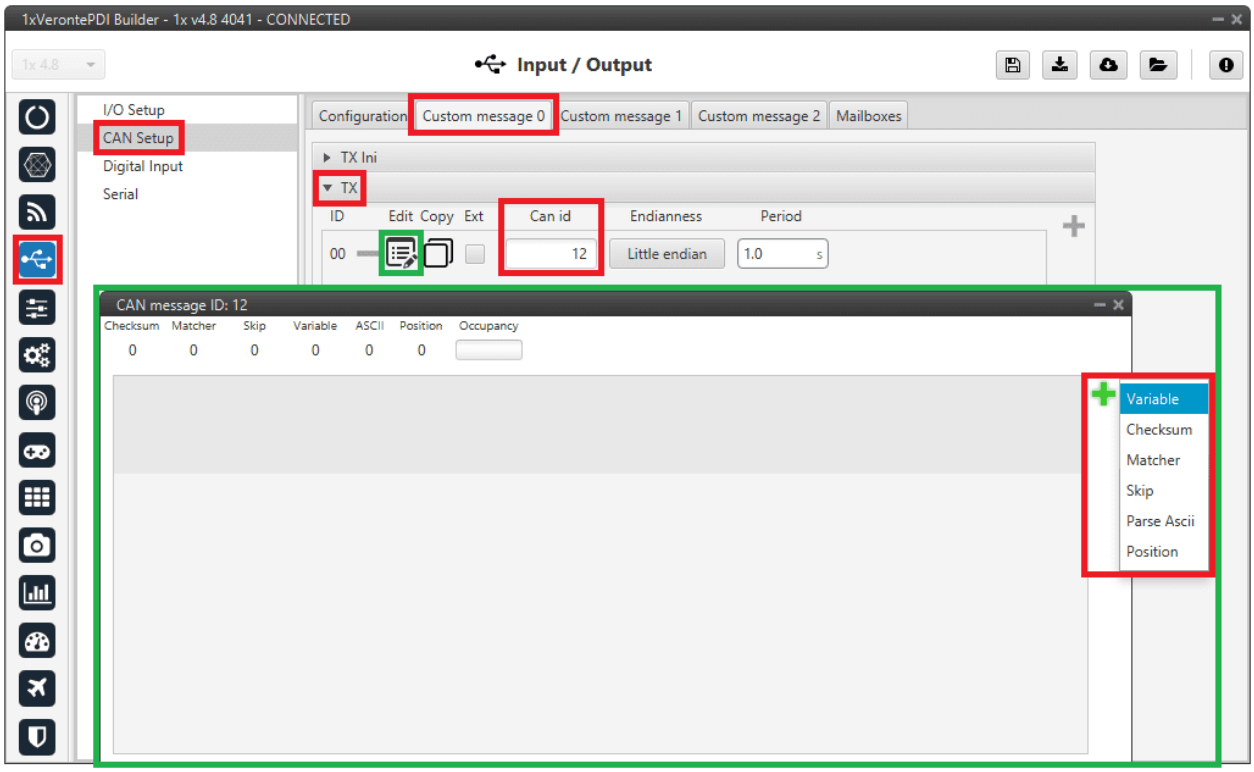


Fig. 9: **CAN messages transmission - Custom message configuration**

2. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect **CAN custom message 0** producer (as the message has been configured in the Custom Message **0** tab) to an **Output filter** as follows:

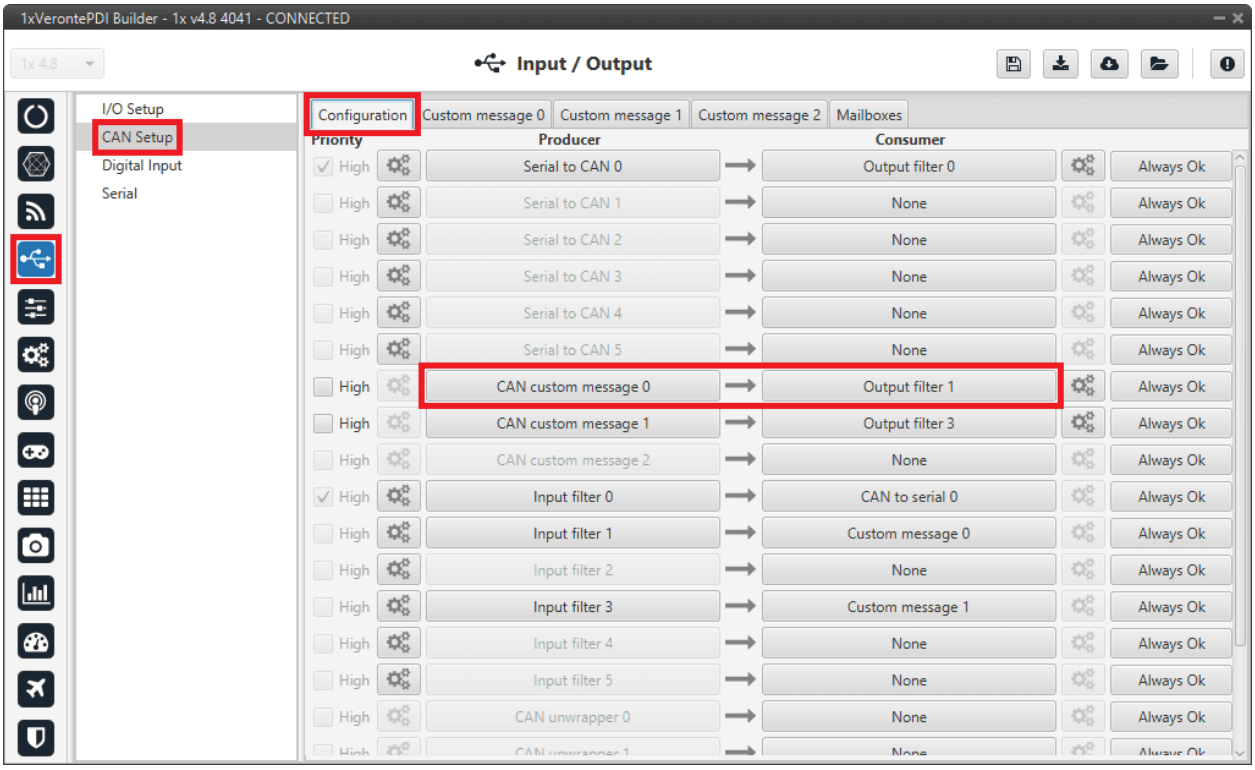


Fig. 10: **CAN messages transmission - CAN Setup configuration**

Warning: Remember that it is necessary to have **at least 1 free mailbox for TX messages**.

3.3.2 CAN messages reception

This section summarizes the configuration to be carried out **to receive CAN streams over a CAN Bus**.

Fig. 11: **CAN messages reception diagram**

1. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Configure the mailbox to receive a message with the appropiate **ID** (in this example ID 28 has been configured):

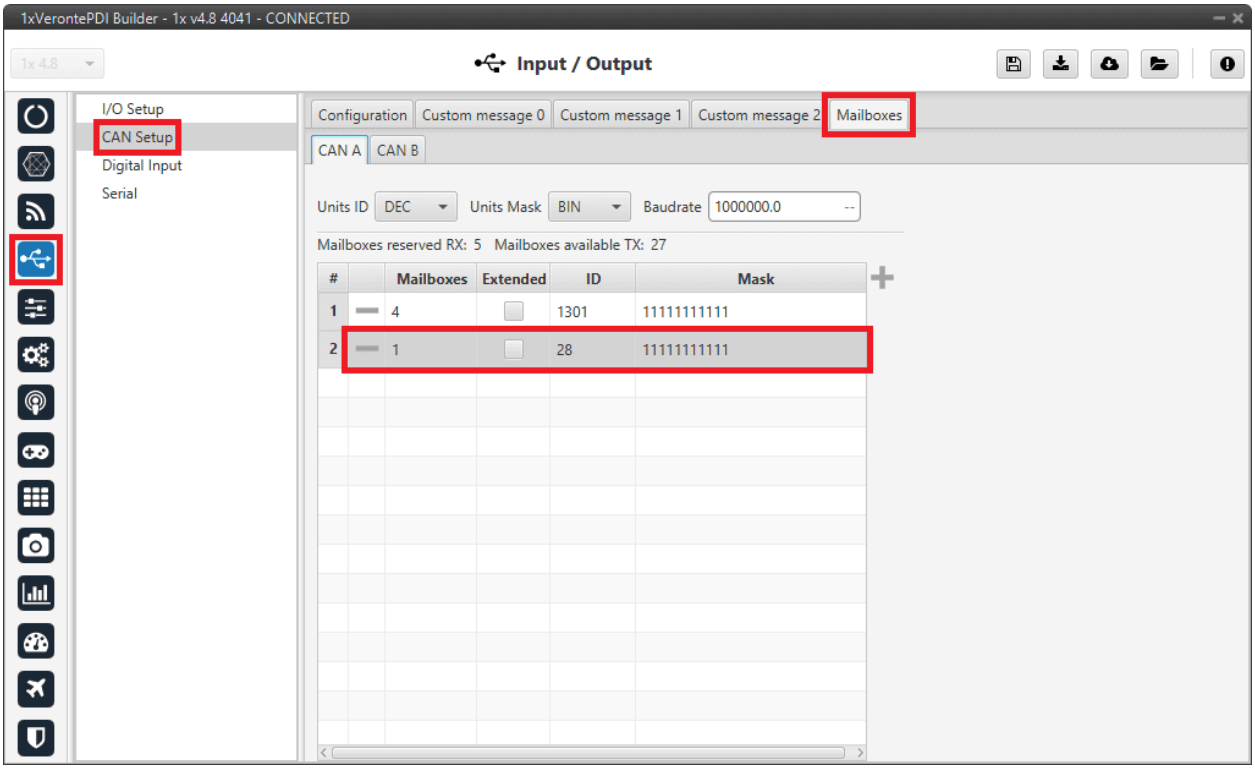


Fig. 12: **CAN messages reception - Mailboxes configuration**

2. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect an **Input filter** with the **right CAN ID** to a **Custom message consumer**:

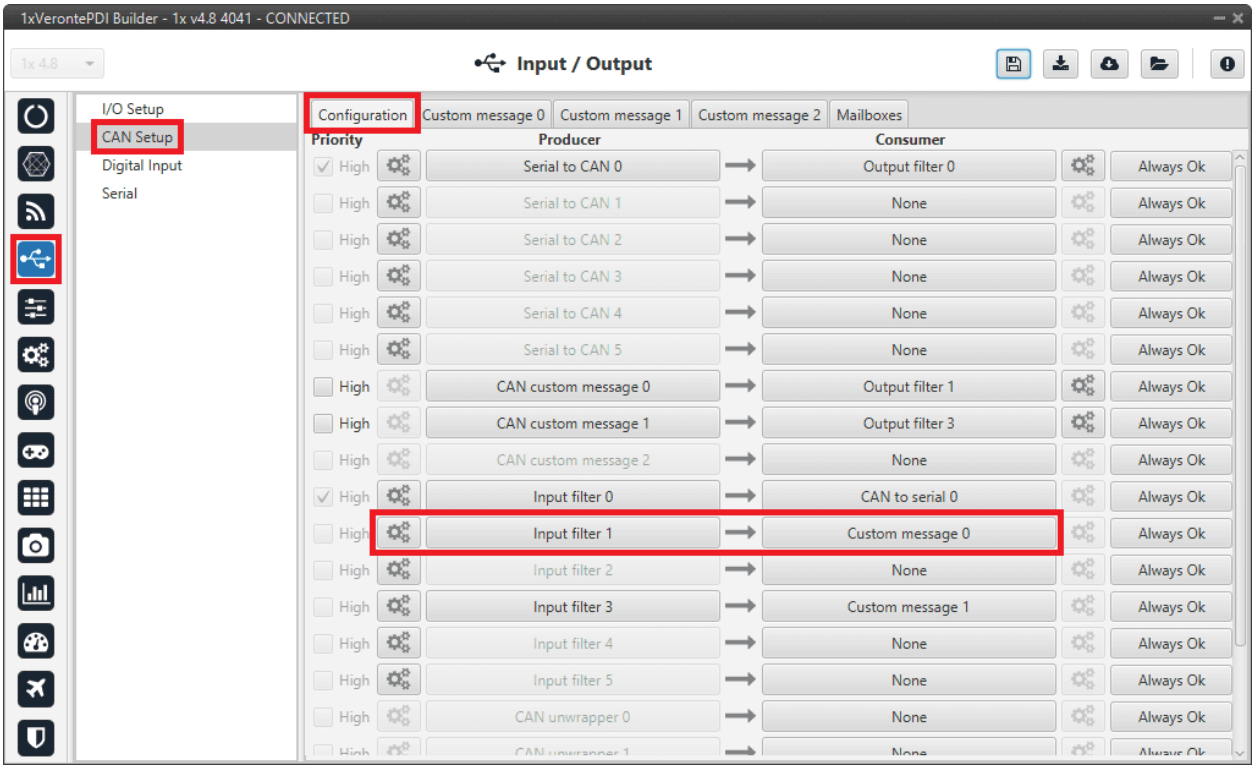


Fig. 13: **CAN messages reception - CAN Setup configuration**

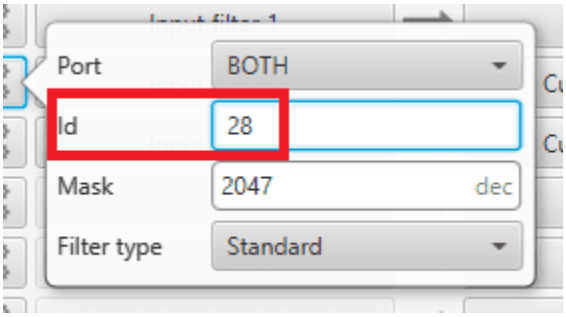


Fig. 14: **CAN messages reception - Input filter configuration**

3. Go to Input/Output menu → CAN Setup panel → **Custom message 0 tab** (as Custom Message **0** has been selected as consumer).

   Configure the message reading as desired in **RX** by setting the correct CAN ID.

   The different options and parameters to be configured are explained in the *RX Messages (Custom Messages) - Input/Output* section of this manual.

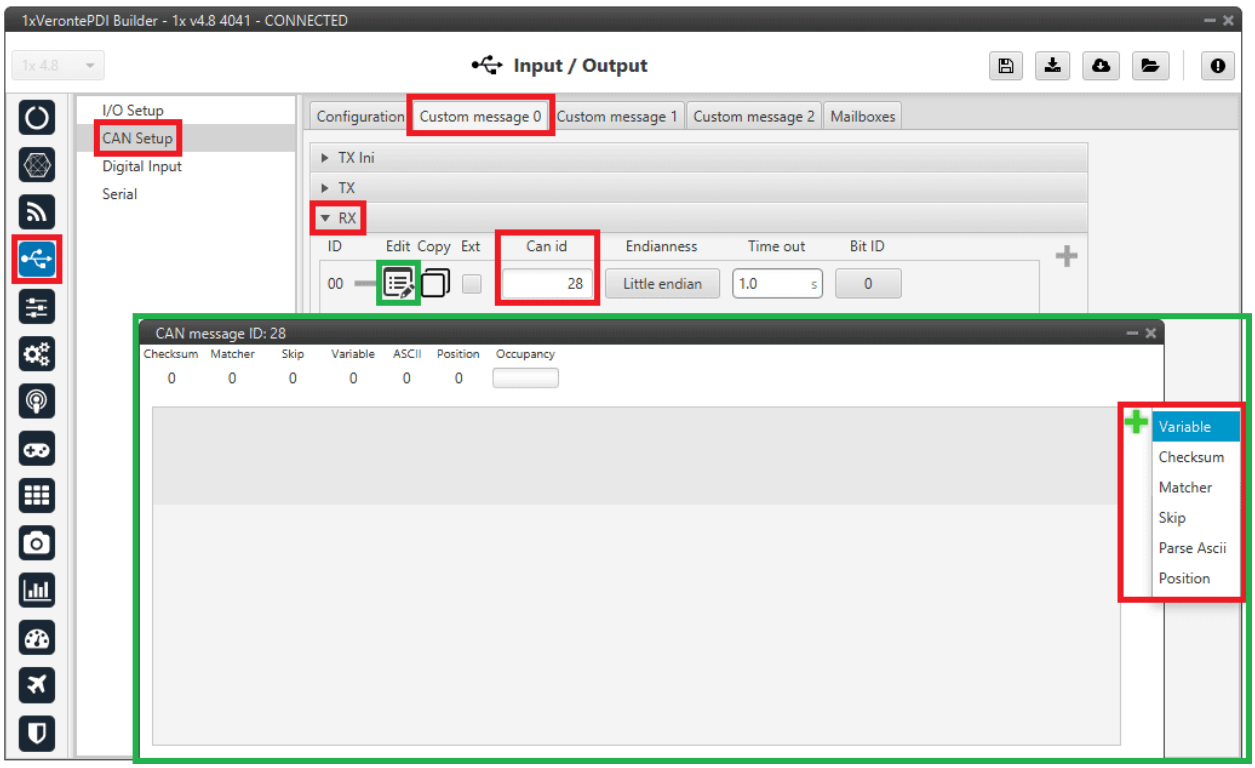


Fig. 15: **CAN messages reception - Custom message configuration**

3.3.3 CAN messages transmission via serial

This section summarizes the configuration to be carried out **to send CAN streams over a serial Bus**.

Note: For sending CAN streams over a serial bus, a **CAN wrapper** port is needed. For further details of this port, please consult the *CAN wrapper/CAN unwrapper - I/O Setup* section of the present manual.

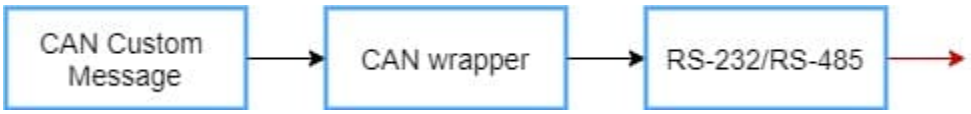


Fig. 16: **CAN messages transmission via serial diagram**

1. Go to Input/Output menu → CAN Setup panel → **Custom message 0 tab**.

   Important: The index of the **Custom message tab** must match the index of the **CAN custom message producer** to configure. In this example, tab **0** is selected.

   Select the fields to send in **TX** or TX Ini, as it is for transmission. More information on the configuration of CAN messages can be found in the *TX/TX Ini Messages (Custom Messages) - Input/Output* section of this manual.

   For example, a CAN message set to ID 12:

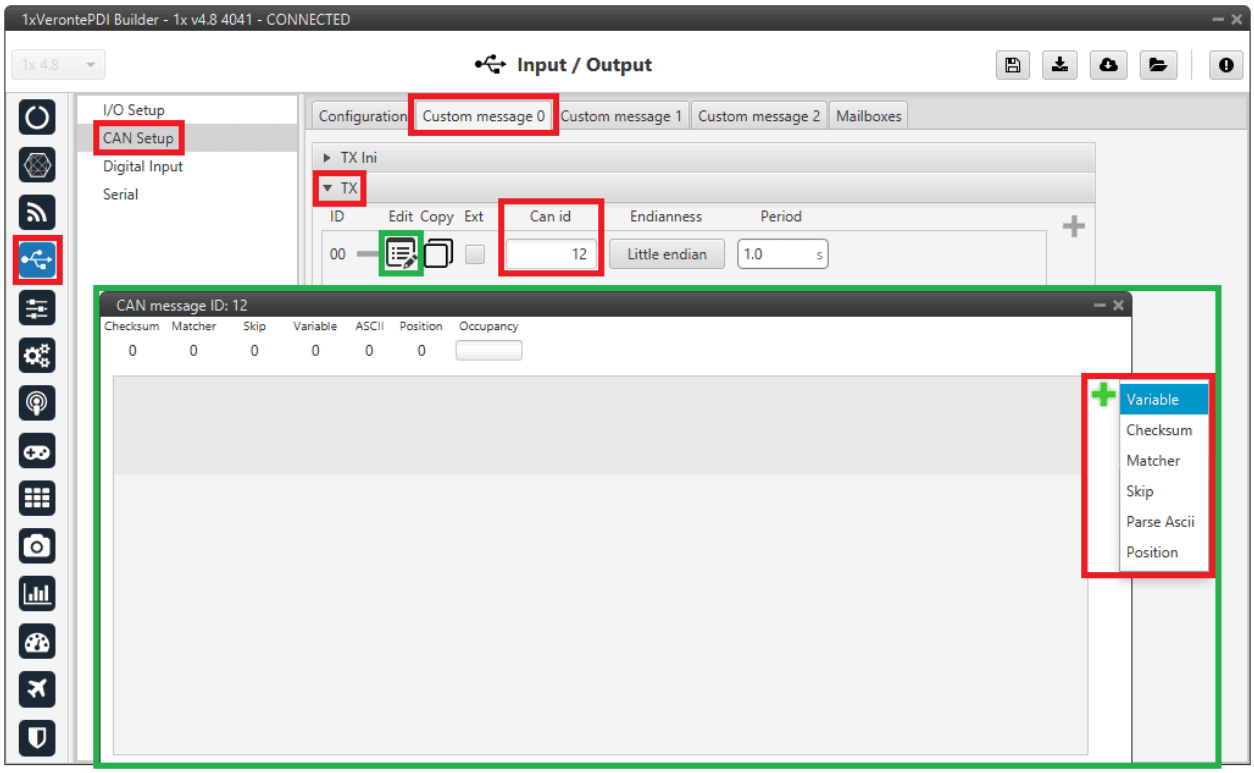


Fig. 17: **CAN messages transmission via serial - Custom message configuration**

2. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect **CAN custom message 0** producer (as the message has been configured in the Custom message **0** tab) to a **CAN wrapper** consumer as follows:

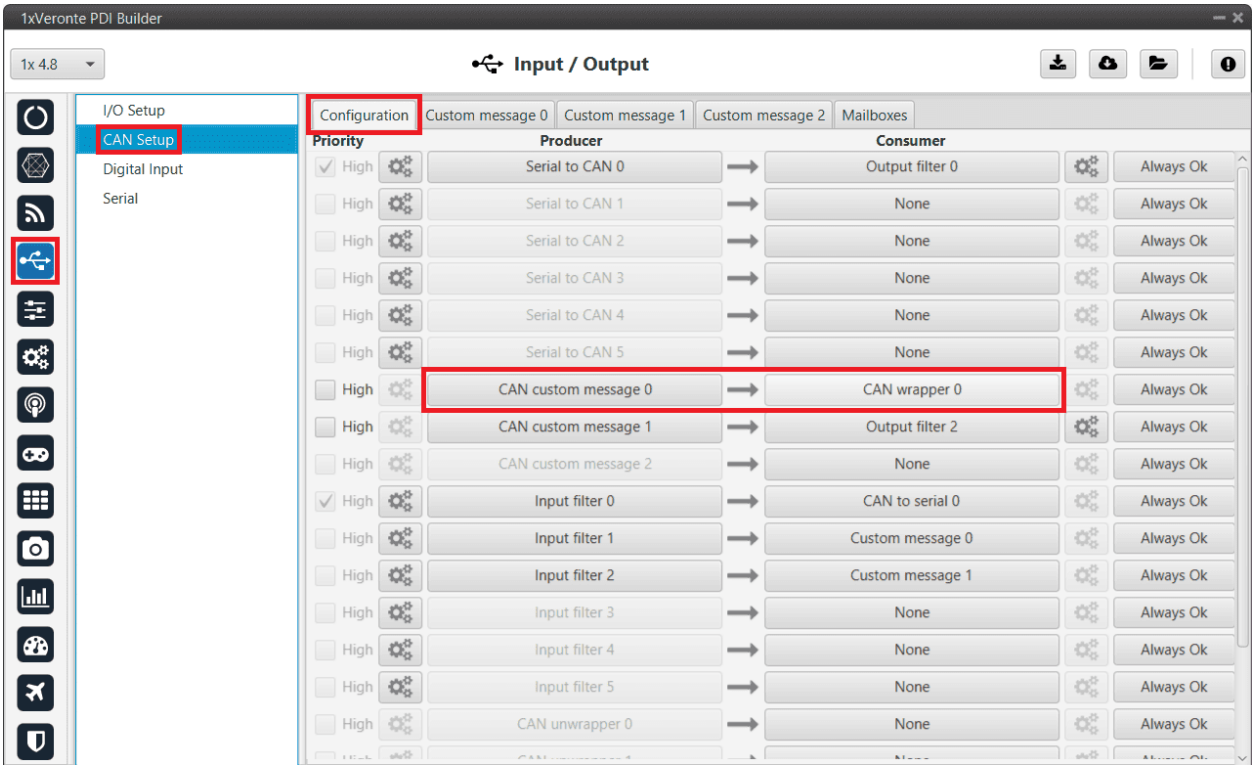


Fig. 18: **CAN messages transmission via serial - CAN Setup configuration**

3. Go to Input/Output menu → I/O Setup panel → **Configuration tab**.

   Connect the **CAN wrapper 0** producer to a **RS232** or **RS485** consumer as shown below.

Important: The **index of the CAN wrapper producer** must match **the index of the CAN wrapper consumer** configured in the previous step.

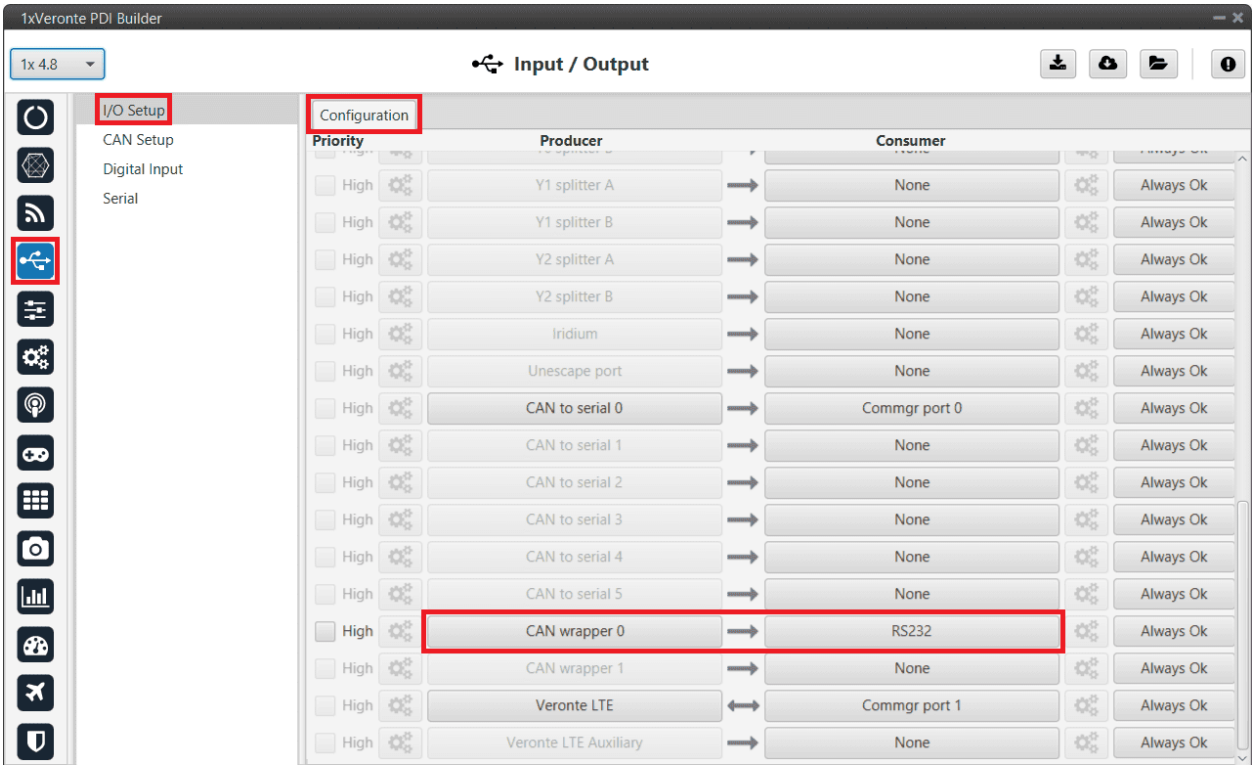


Fig. 19: **CAN messages transmission via serial - I/O Setup configuration**

Warning: Remember that it is necessary to have **at least 1 free mailbox for TX messages**.

3.3.4 CAN messages reception via serial

This section summarizes the configuration to be carried out **to receive CAN streams over a serial Bus**.

Note: For receiving CAN streams over a serial bus, a **CAN unwrapper** port is needed. For further details of this port, please consult the *CAN wrapper/CAN unwrapper - I/O Setup* section of the present manual.

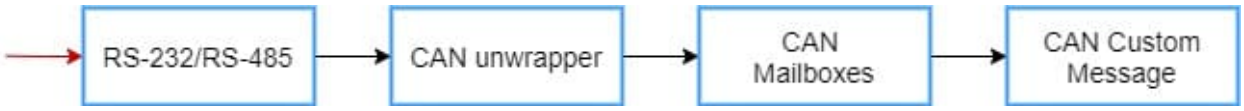


Fig. 20: **CAN messages reception via serial diagram**

1. Go to Input/Output menu → I/O Setup panel → **Configuration tab**.

   Connect a **RS232/RS485** producer to a **CAN unwrapper** consumer.

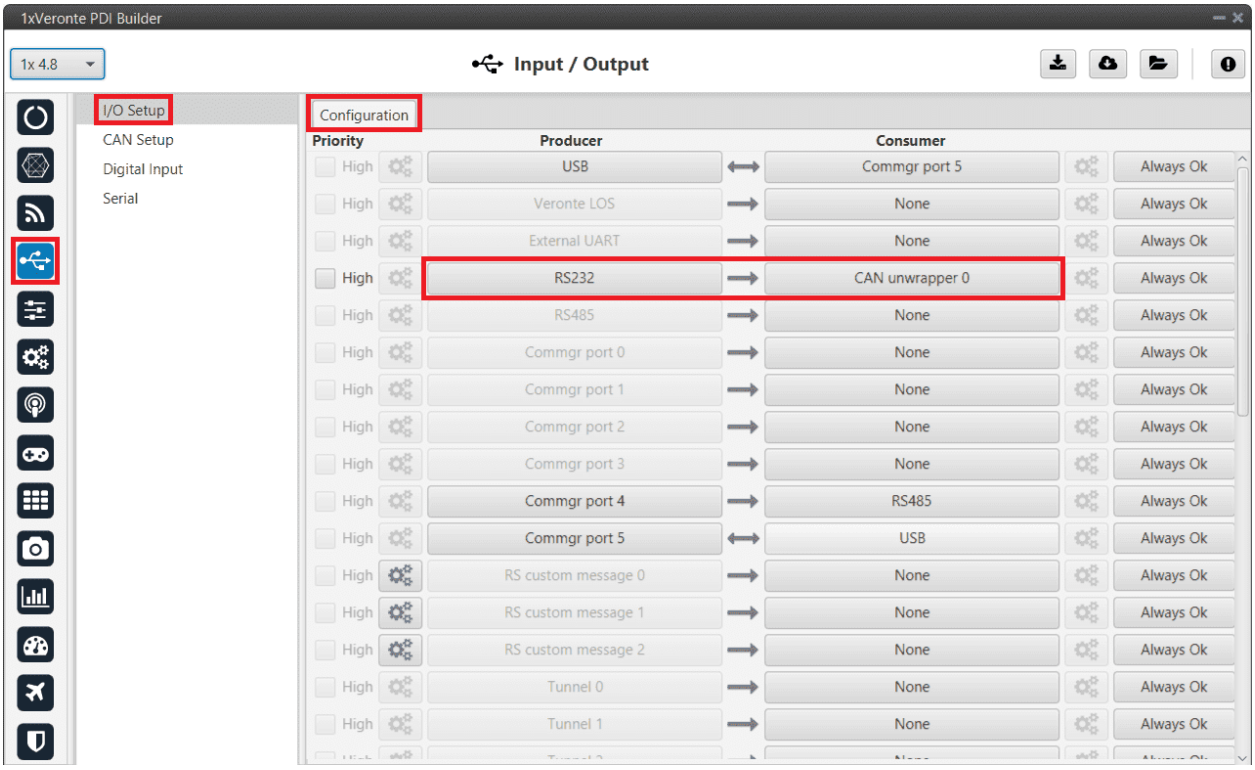


Fig. 21: **CAN messages reception via serial - I/O Setup configuration**

2. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Configure the mailbox to receive a message with the appropiate **ID** (in this example ID 28 has been configured):

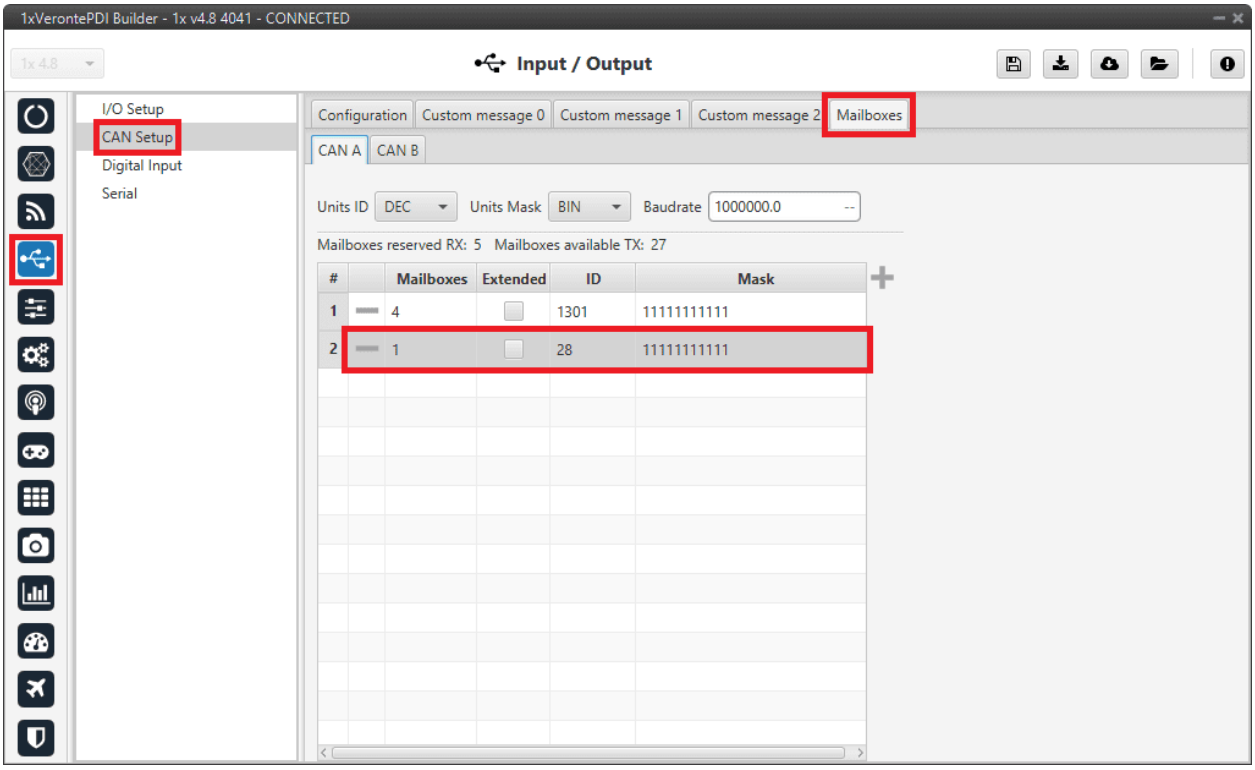


Fig. 22: **CAN messages reception via serial - Mailboxes configuration**

3. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect a **CAN unwrapper** producer to a **Custom message** consumer. In this example, **Custom message 0** consumer is selected.

Important: The **index of the CAN unwrapper producer** must match **the index of the CAN unwrapper consumer** configured in the previous step.

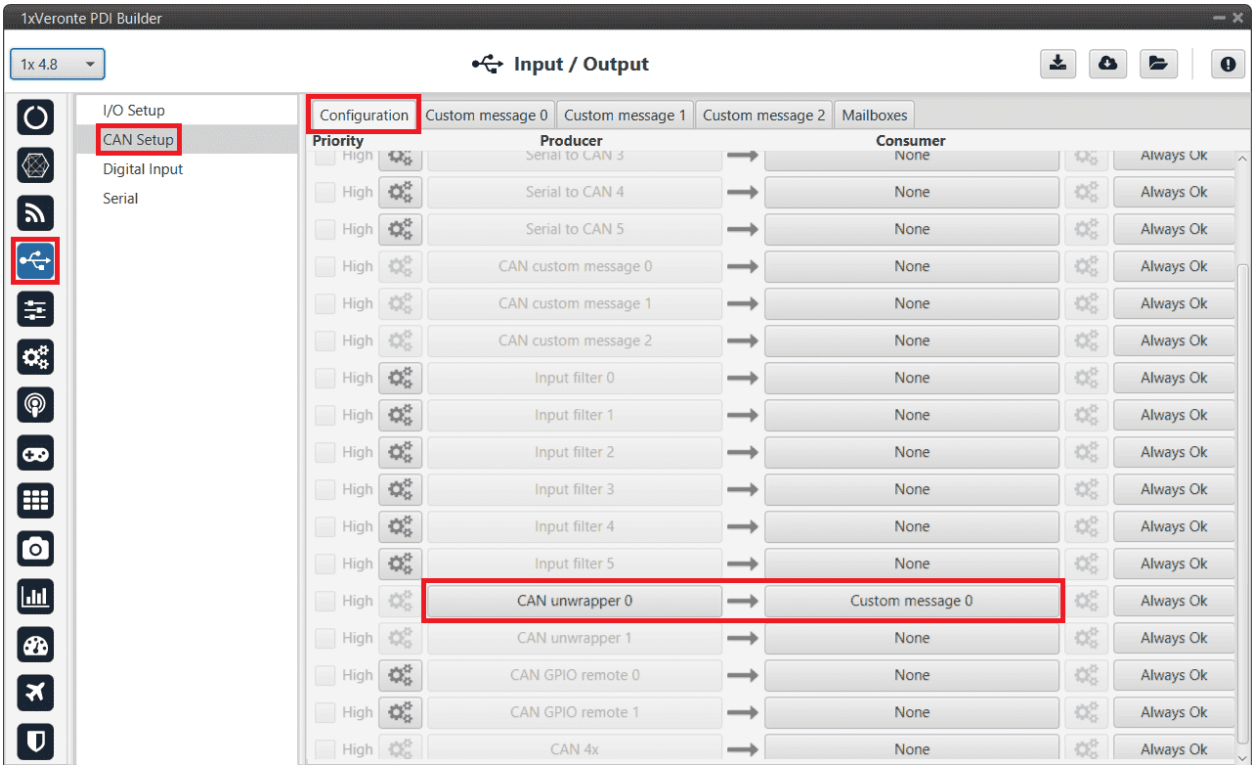


Fig. 23: **CAN messages reception via serial - CAN Setup configuration**

4. Go to Input/Output menu → CAN Setup panel → **Custom message 0 tab**.

Important: The index of the **Custom message tab** must match the index of the **Custom message consumer** to configure. In this example, tab **0** is selected.

Configure the message reading as desired in **RX** by setting the correct CAN ID.

The different options and parameters to be configured are explained in the *RX Messages (Custom Messages) - Input/Output* section of this manual.

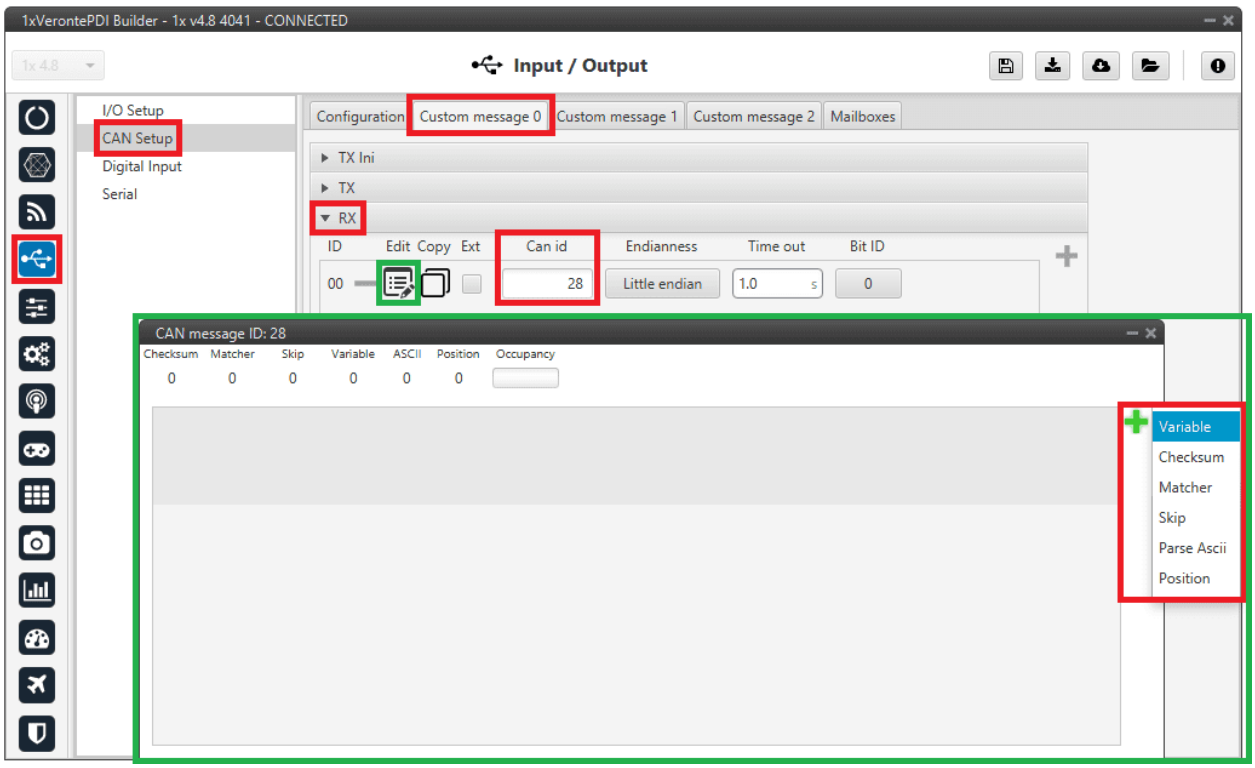


Fig. 24: **CAN messages reception via serial - Custom message configuration**

3.4 Data transmission between Veronte Autopilots 1x

To establish a proper communication between the ground and air units, the **telemetry** and **sniffer** menus must be configured, respectively.

A simple example of use between a ground unit and an air unit is shown below:

In the **1x ground unit**:

1. Go to Telemetry menu → Telemetry panel → **Data link to VApp tab** (for more information about this, see *Data vectors - Telemetry* section of this manual).
2. Add the variables: *Absolute: UAV position*, *Yaw*, *Pitch* and *Roll*.
3. Set a **Frequency**, it is recommended to set it to **10 Hz**.
4. On **Address**, point to the **1x air unit** (it is needed to have both units connected through the radio in order to be able to see them on the menu).

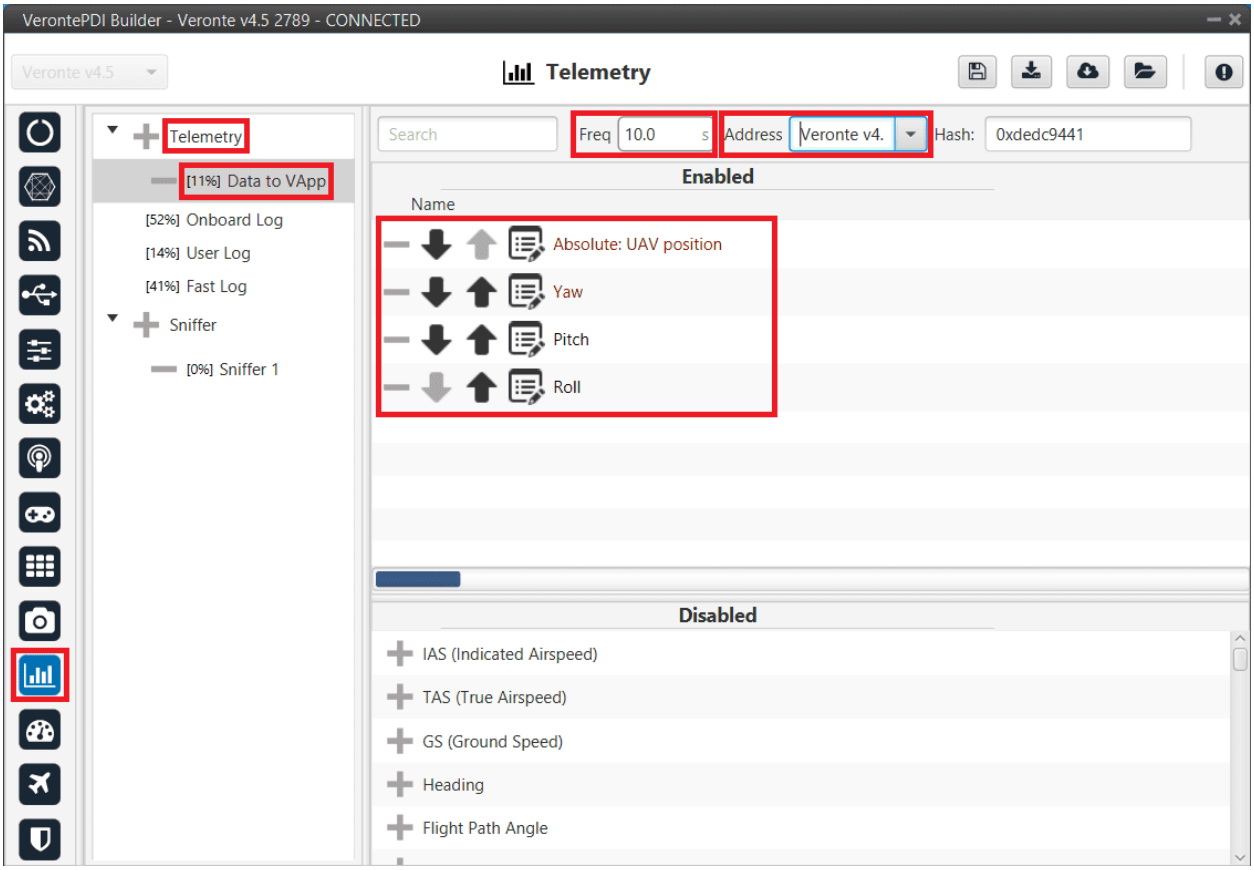


Fig. 25: **1x ground unit - Telemetry**

For the **1x air unit**:

1. Go to Telemetry menu → **Sniffer panel** (for more information about this, see *Sniffer - Telemetry* section of this manual).
2. Add a new Sniffer.
3. Configure the **same variables** (keeping the **same order**) than in the ground unit.
4. On **Address**, point to the **1x ground unit**.
5. In the gear next to it, configure the 4 incoming variables as System Variables: assign UAV Position to **Moving Object** and the 3 variables from attitude to 3 different **User Variables** (keeping the **same order** as well).

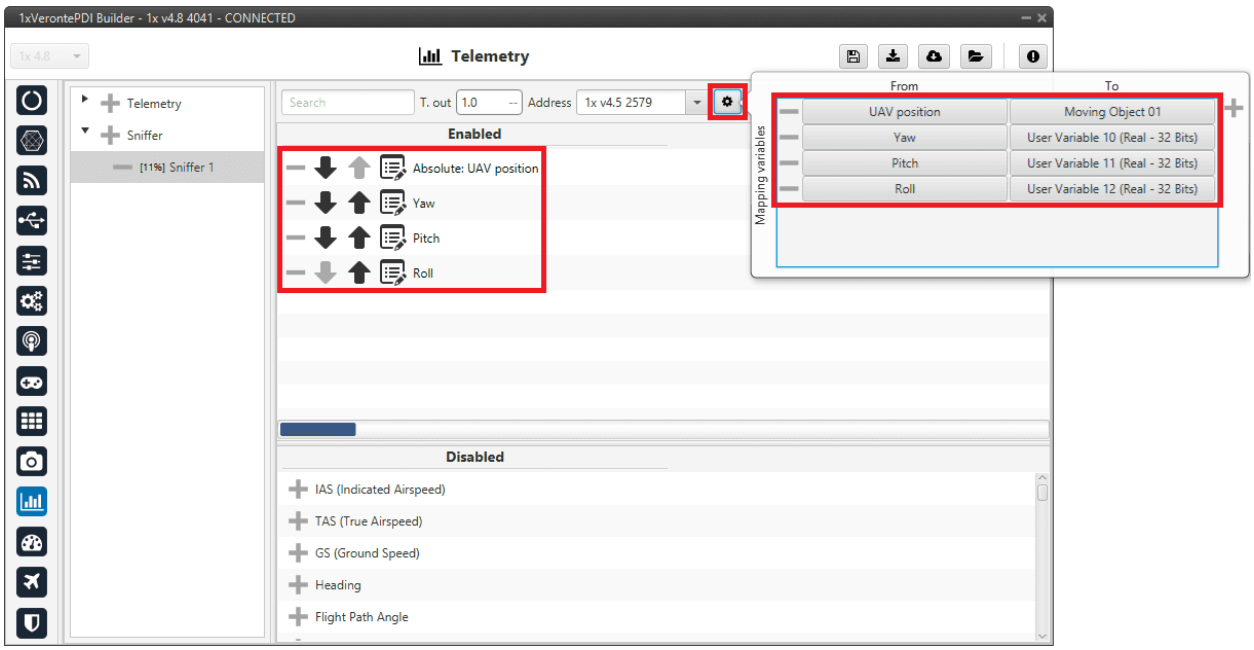


Fig. 26: **1x air unit - Sniffer**

3.5 Flare and Decrab phase configuration

As flare and decrab guidance are not included in the landing algorithm, since the decrab is control dependent (yaw must be aligned with the runway direction), the following shows how to implement this guidance in a generic fixed wing configuration:

1. First, it can be useful to create a program that computes the altitude above the touchpoint, the lateral error over the desired route and the heading error. If all conditions met, the flare phase is triggered:

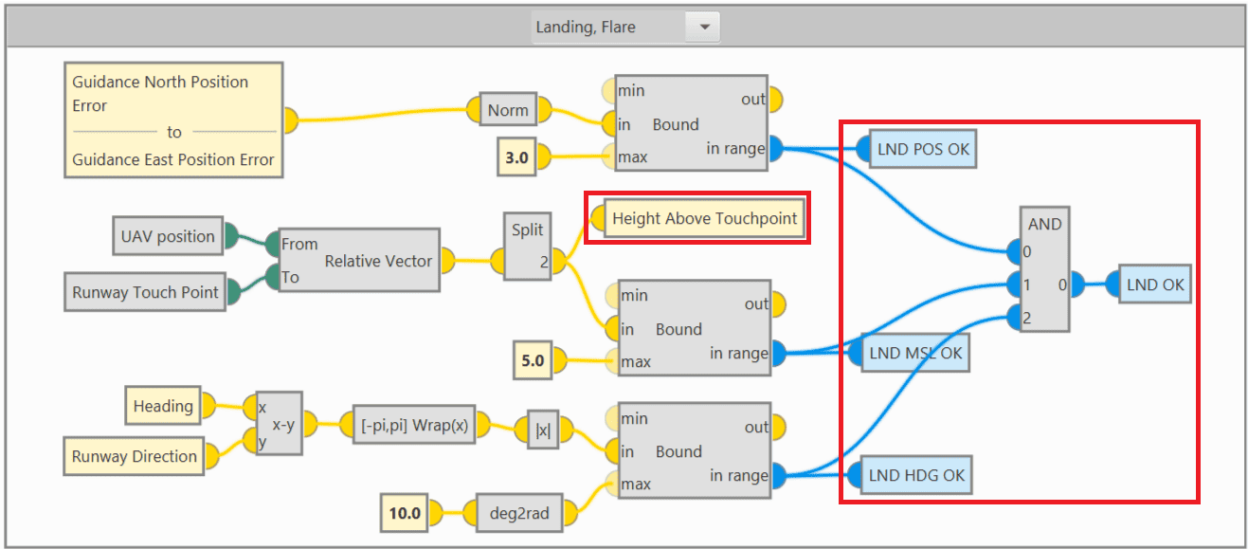


Fig. 27: **Flare and Decrab phase - Auxiliary program**

2. **Flare and Decrab guidance definition**

   In the **Guidance program** (where the guidances for all phases are defined), add the **Flare phase** to the *Phase Switch* block. Then, the **flare and decrab guidance** must be built as follows:

   - The **desired vertical speed** is overwritten to be a function of the height above the runway. In this case the height is calculated in an auxiliary program (**Height Above Touchpoint** variable calculated in *step 1*), but the AGL could be used.
   - The **desired yaw** is overwritten directly by the **runway direction**.
   - The **desired roll** is overwritten to 0º.
   - The **desired IAS** is overwritten to a speed slightly above the stall speed, in this example 15 m/s.

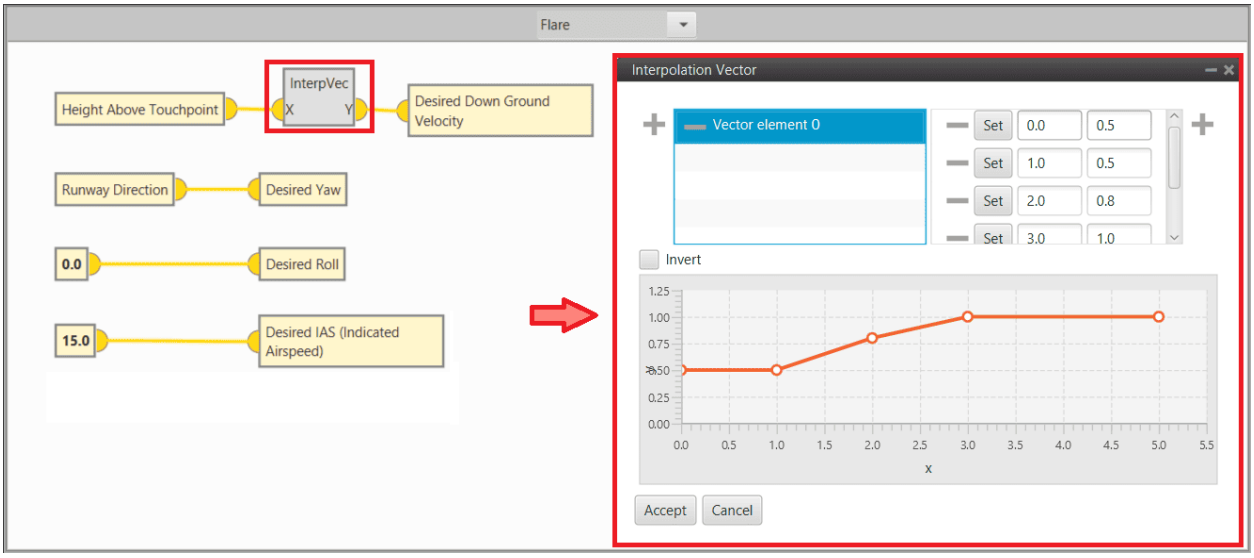


Fig. 28: **Flare and Decrab phase - Flare and decrab guidance definition**

3. **Pitching control**

   The **desired vertical speed** is used for **pitch control** as shown below:

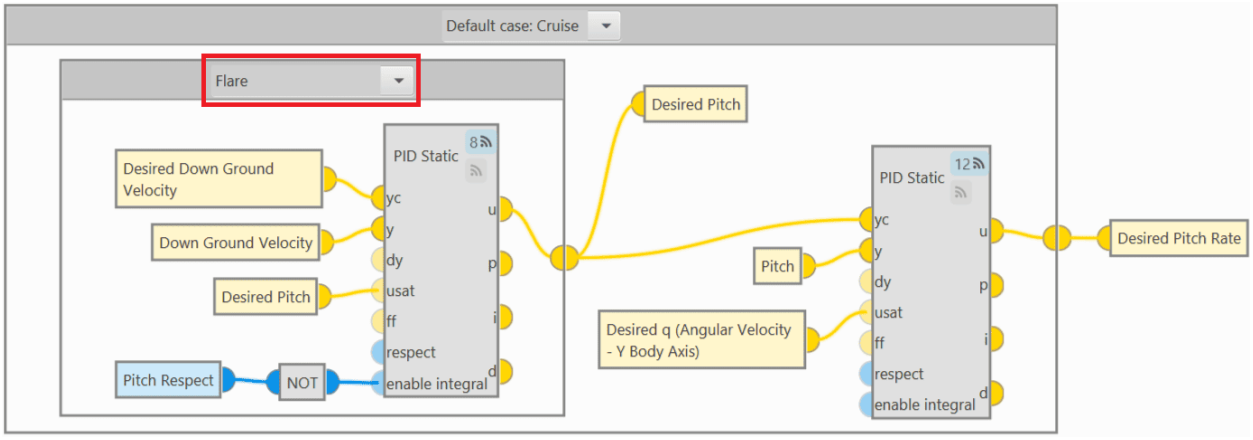


Fig. 29: **Flare and Decrab phase - Pitching control**

4. **Thrusting control**

   The **throttle** is used to maintain the IAS (slightly above stall) and when a height above touchpoint of 1 m is reached, the motor is cut off. Again, a user variable is used here (**Height Above Touchpoint** variable calculated in *step 1*), but the AGL could be used:

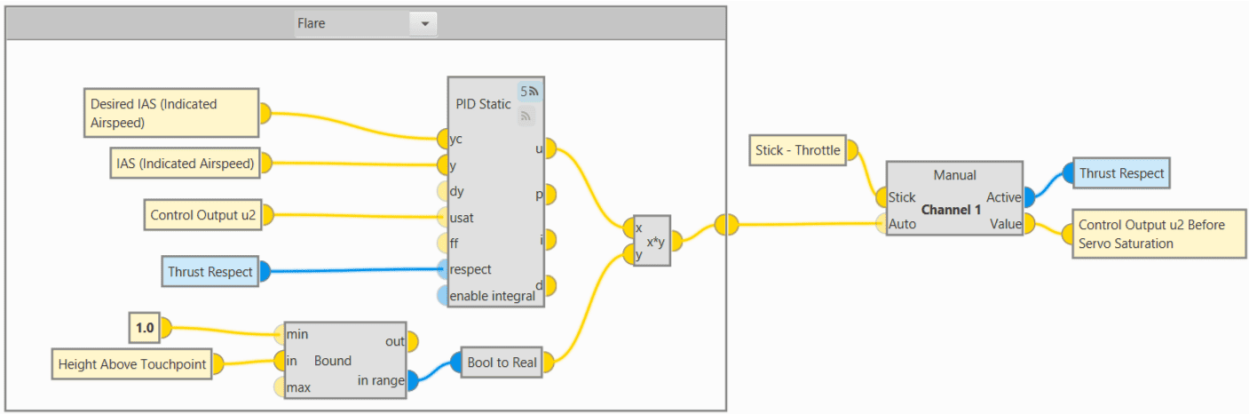


Fig. 30: **Flare and Decrab phase - Thrusting control**

5. **Rolling control**

   For the **roll**, simply try to maintain the desired roll (0º):

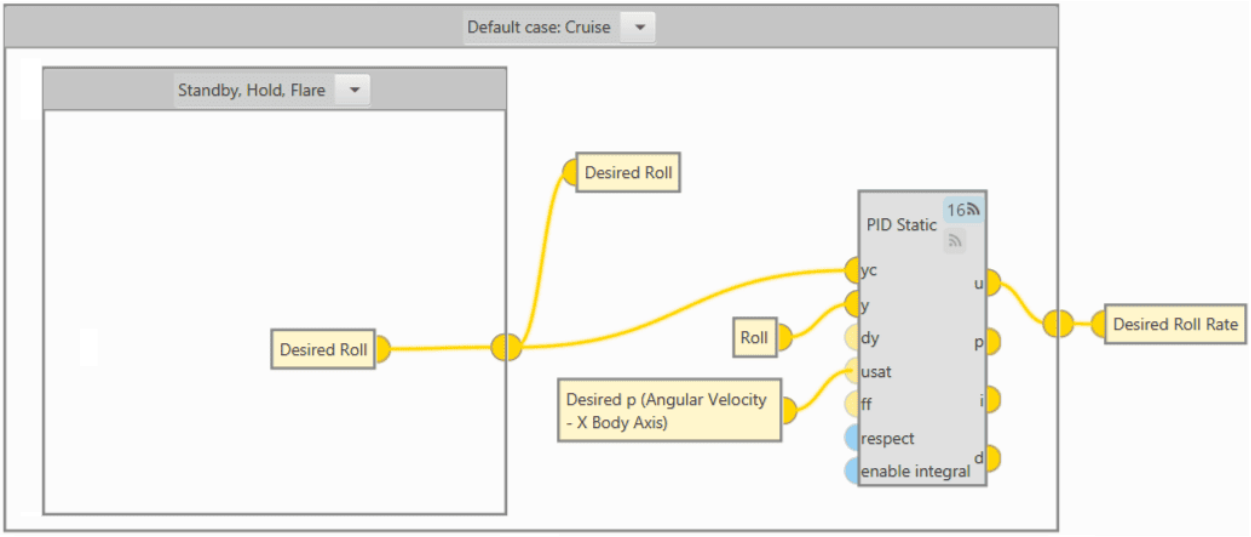


Fig. 31: **Flare and Decrab phase - Rolling control**

6. **Yawing control**

   The **yaw control** (rudder) is based on the **desired yaw**:

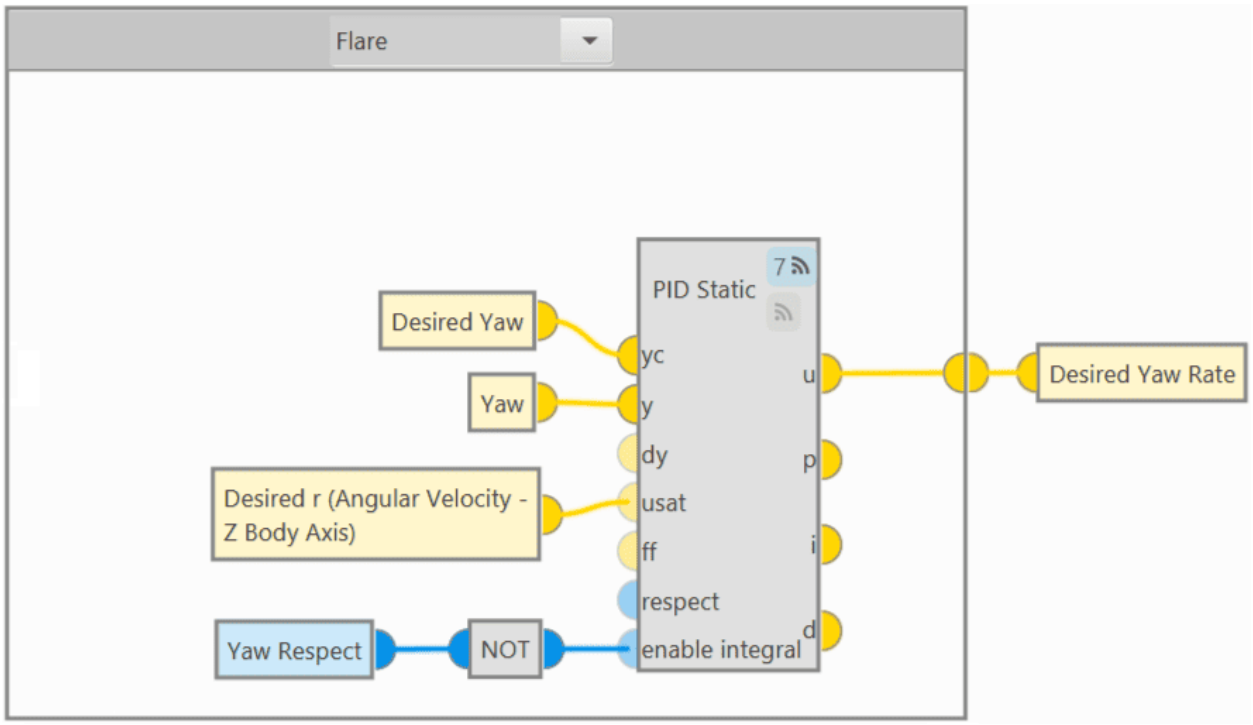


Fig. 32: **Flare and Decrab phase - Yawing control**

Important: This algorithm requires a good yaw estimation if users want to have **Decrab**, so **magnetometer** or **GNSS Compass** is required.

3.5.1 Flare phase configuration

However, for a **Flare** phase **without Decrab**, the **yaw** control should be based on **heading** instead of being based on yaw. To do so, the **Guidance and Yawing programs** must look as shown below:

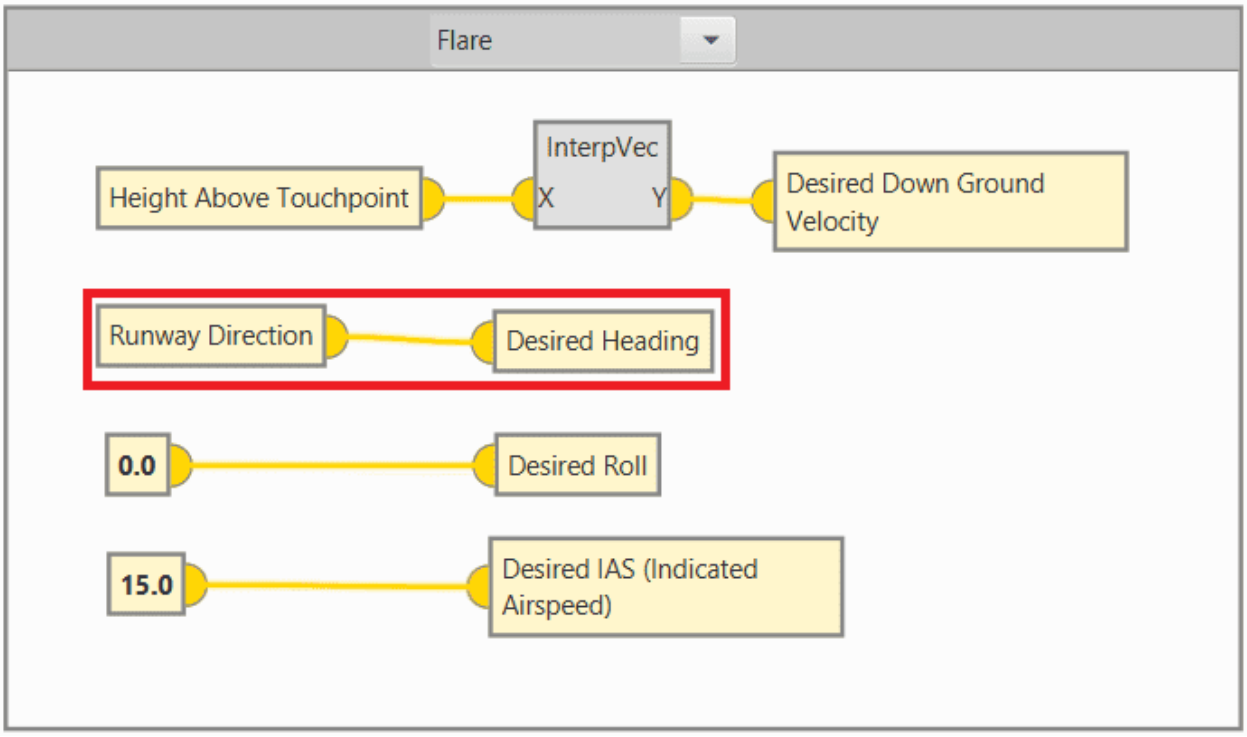


Fig. 33: **Flare phase - Flare guidance definition**

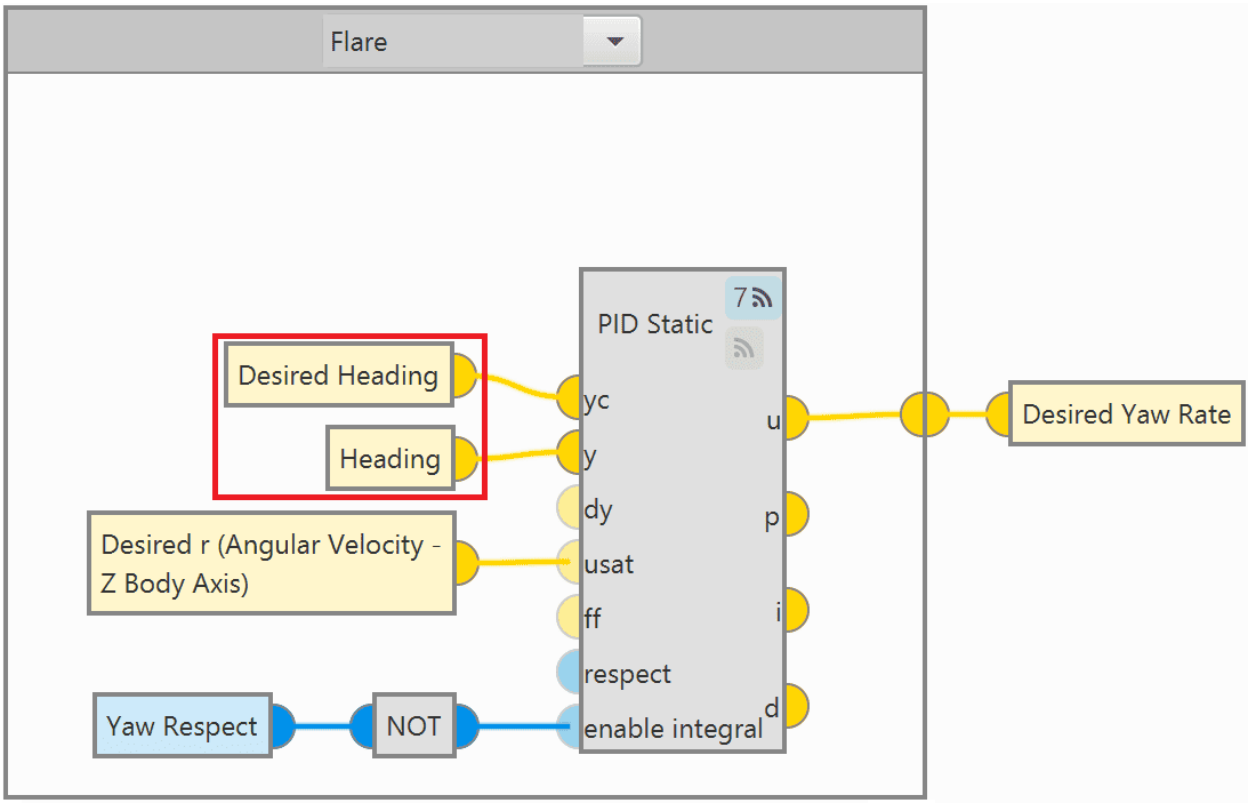


Fig. 34: **Flare phase - Yawing control**

For more information on block programs, please refer to *Block Programs* section of this manual.

3.6 RTK Configuration

The RTK configuration is performed through the **GNSS sensor block** with the help of the **RTK Wizard**.

Fig. 35: **RTK Configuration - GNSS block**

Users must use this wizard to perfom the corresponding settings for the 1x air and ground units:

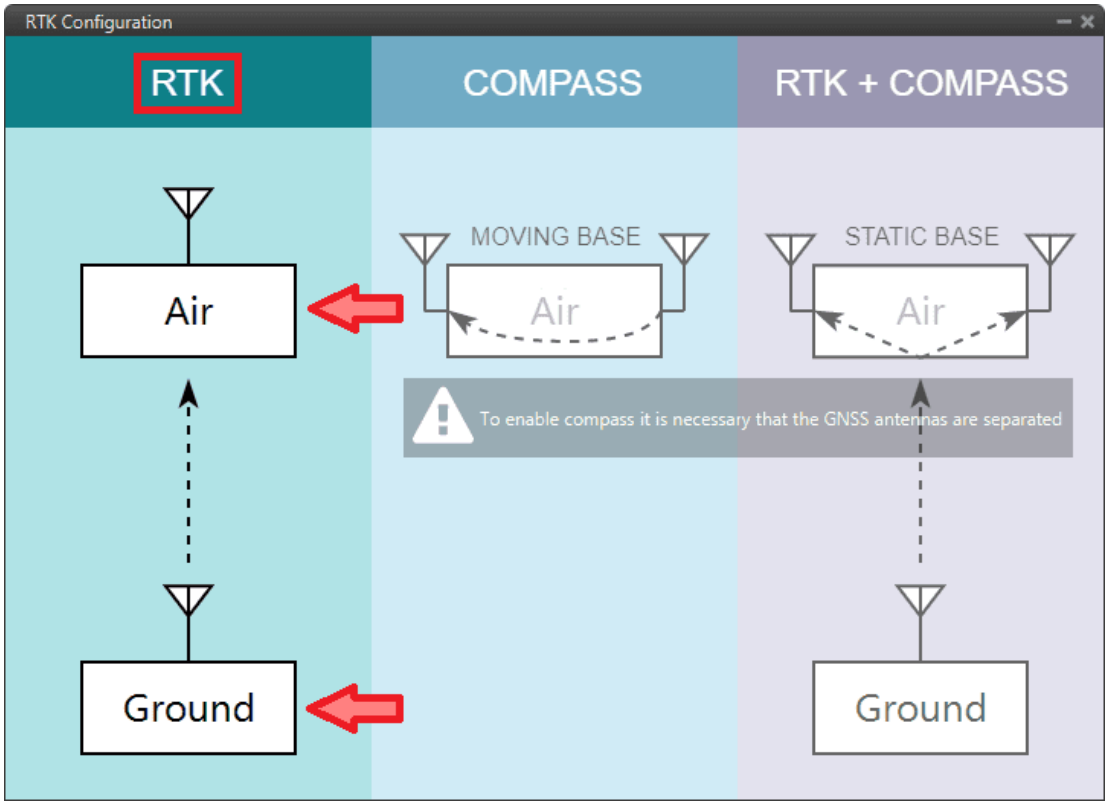


Fig. 36: **RTK Configuration - RTK Wizard**

Danger: This wizard uses **by default** the **Y0 splitter** port. Consequently, if it has been configured for another use, the configuration will be now **overwritten**.

Therefore, to avoid system malfunction when using RTK, it is **highly recommended** to leave the **Y0 splitter** on the **ground unit** "free".

This is especially important if the ground unit is a PCS.

- For the **1x air unit**, click on "Air" in the RTK column and the corresponding RTK configuration will be automatically applied to the user's configuration.
- For the **ground unit**, click on "Ground" in the RTK column and the corresponding RTK configuration will be automatically applied to the user's configuration.

  In the *Ublox preset* parameter, the **Static base** option will be selected. However, if user wishes to consult the configuration made, just select the **Custom** option:

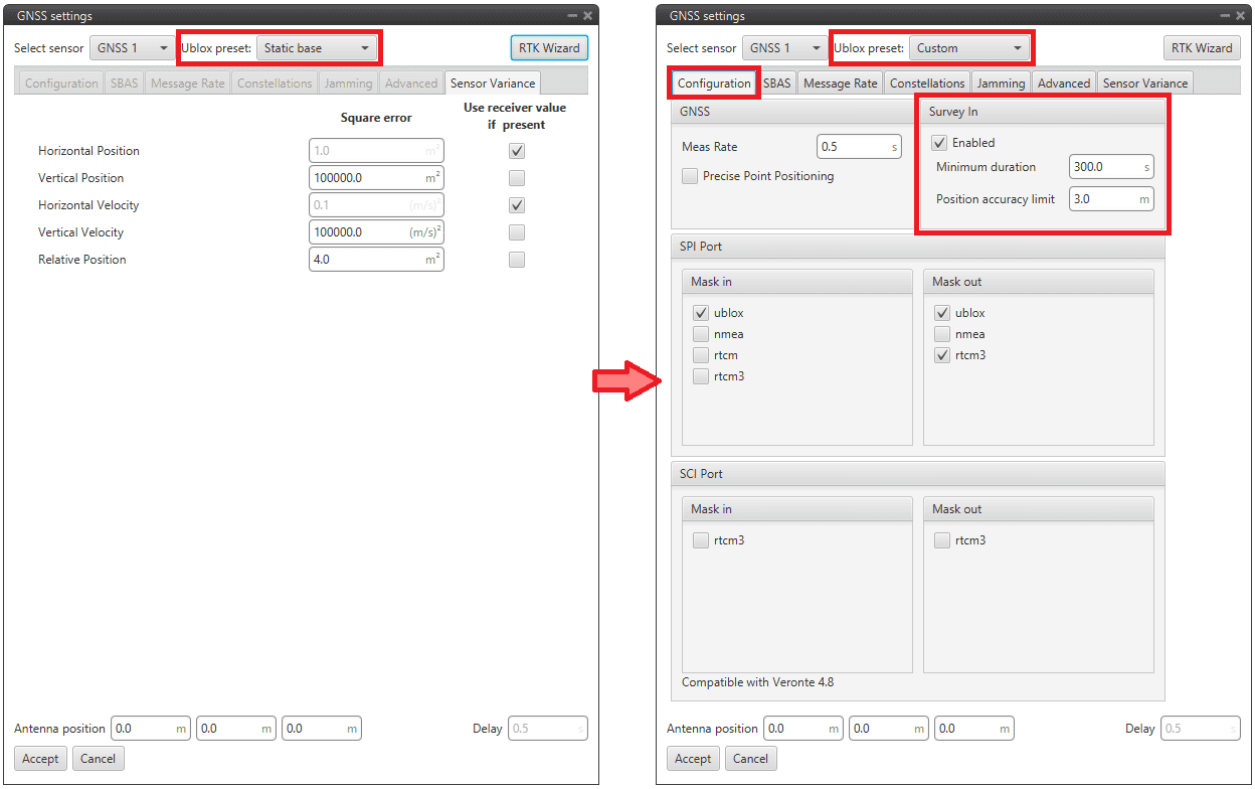


Fig. 37: **RTK Configuration - GNSS block ground configuration**

Important: Please pay specific attention to the "**survey configuration**" options.

For a detailed explained of the **GNSS sensor block**, please refer to *GNSS sensor - Sensors blocks* of **Block Programs** section.

With the previous configurations, when **GNSS accuracy < 3 m** ⇒ **GNSS sensor** will start the **survey in during 300 seconds** while taking measurements for **RTK correction**.

To verify correct **survey in** behavior, in **Veronte Ops** users can query the status of the following variables.

- In the **ground unit**, if the survey in has **started**, 'GNSS1 Survey In Off' bit should be in "success" mode ⇒ **GNSS1 Survey In** (if the label has the default setting, it will be green).

  Therefore, when **finished**, the bit will switch to "fail" mode ⇒ **GNSS1 Survey In Off** (if the label has the default setting, it will be colored red).

- On the 1x **air unit**, once the survey in has **finished**, check that 'DGNSS1 Input Off' and 'DGNSS1 Navigation Off' bits are in "success" mode ⇒ **DGNSS1 Input On** and **DGNSS1 Navigation On** respectively (if the labels have the default setting, they will be colored green).

  At this time, the **GNSS1 Accuracy** value in the **air unit**, should be very low, probably between 0.01-0.05 m.

3.7 External devices

The step-by-step instructions for the following external devices will be explained in detail in the following sections:

- *Altimeters*
- *External sensors*
- *Radios*
- *Servos*
- *Stick*
- *Veronte products*

3.7.1 Altimeters

3.7.1.1 Lidar

The integration between Veronte Autopilot 1x and a lidar is performed using a variety of interfaces depending on the lidar device. The most common interfaces are I2C or analog although serial or CAN bus can also be used if the lidar is compatible.

3.7.1.1.1 ADC lidar

An ADC lidar changes the voltage depending on the measured distance and therefore the connection to the Autopilot 1x is made using the **ADC pins** (see Pinout - Hardware Installation section of the **1x Hardware Manual**).

Once connected to Autopilot 1x, the value can be monitored in **1x PDI Builder** by using the variables ADC0 to ADC4.

Note: For pin *ANALOG_0* the correspondent ADC variable in **1x PDI Builder** is *ADC0*, for *ANALOG_1* is *ADC1* and so on.

---

1. Go to Connections menu → **ADC 0 panel** (This is only an example, the user must select the ADC pin where the signal is connected).

   Click on 'Create new program':

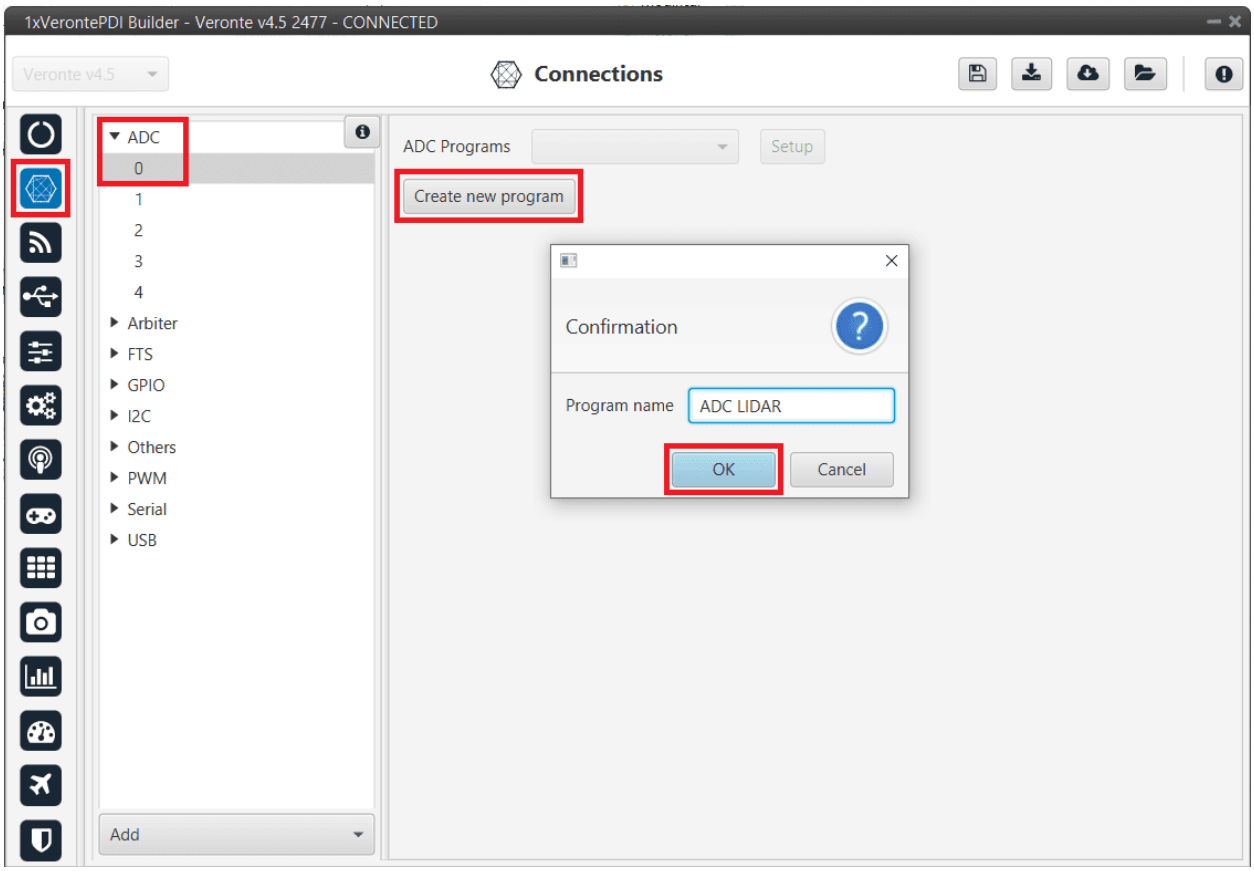


Fig. 38: **ADC lidar - Create ADC program**

2. Go to **Block programs menu**.

   Configure the following operation (for more information about blocks, see *Block Programs* section of this manual):

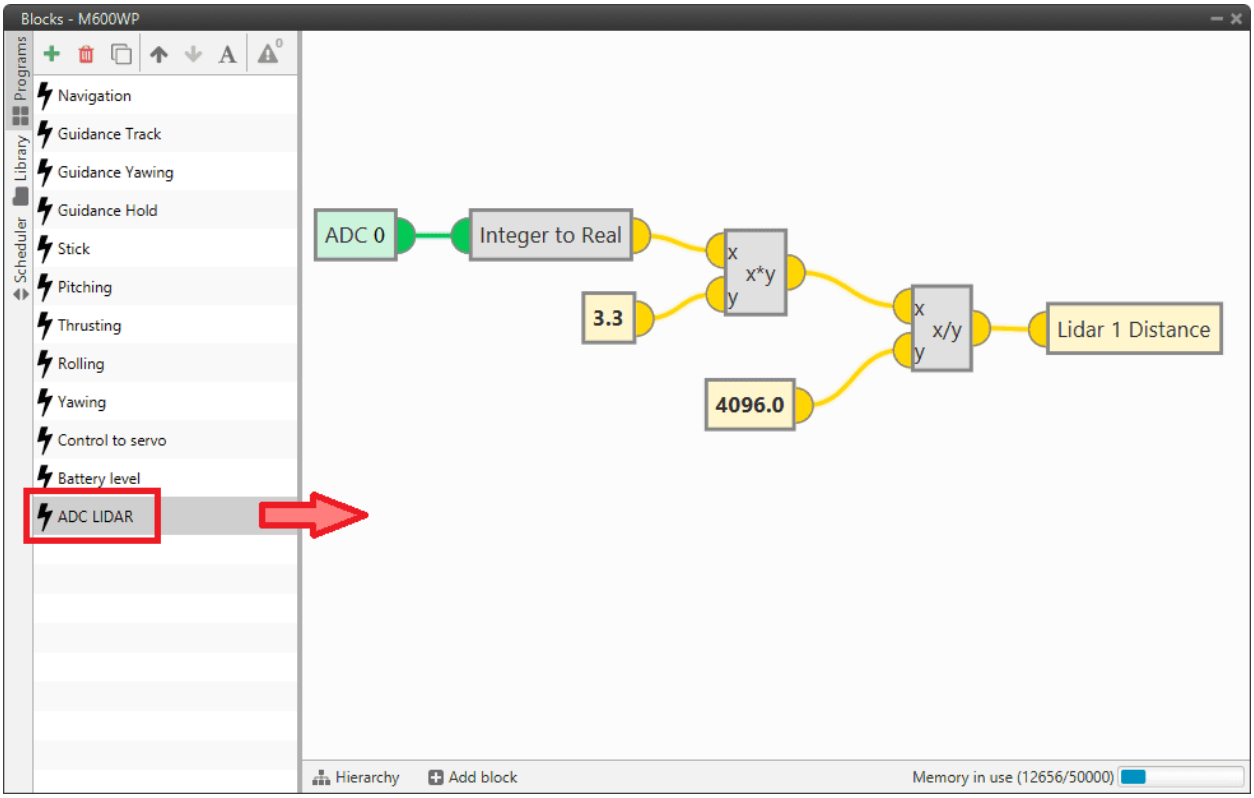


Fig. 39: **ADC lidar - Lidar operation**

After implementing the operation the variable *Lidar 1 Distance* will represent the distance measured by the lidar.

Note: As Autopilot 4x can read up to 36 V per each ADC, the 3.3 value of the ADC program must be changed to 36 if applicable.

3.7.1.1.2 I2C lidar

I2C lidars are configured slightly differently.

Connect the lidar following the pinout provided by the manufacturer and connect it to the Veronte Autopilot 1x **I2C bus** following the Pinout - Hardware Installation section of the **1x Hardware Manual**.

In this case it is not needed to transform the lidar readings, the readings will be reported directly in the selected lidar distance variable.

Go to Sensors menu → **Lidar panel**.

- Enable Lidar 0
- Select the desired Lidar from the drop-down menu
- Set the I2C address

More information on the available lidar options can be found in the *Lidar - Sensors* section of this manual.

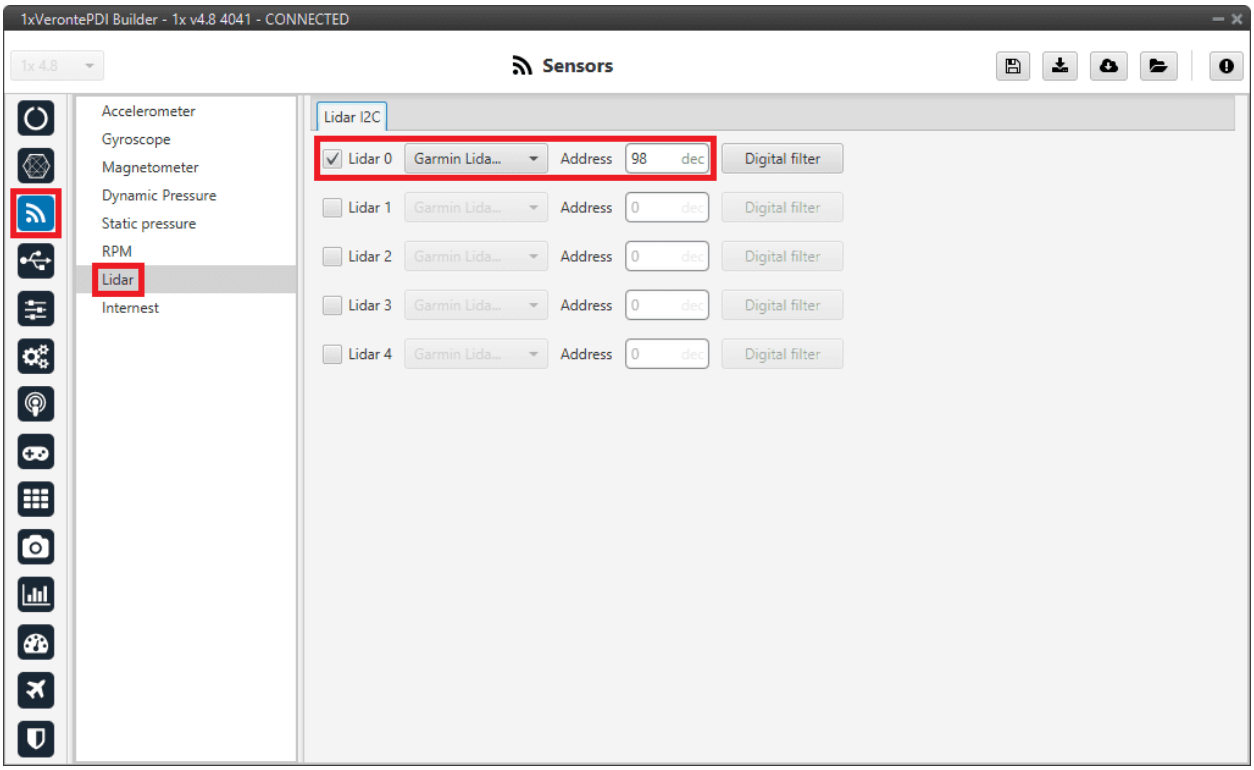


Fig. 40: **I2C lidar**

Warning: I2C address will be different for different devices make sure to define it properly by checking the manufacturer documentation.

3.7.1.1.2.1 Lightware LW 20 Lidar

To integrate the **Lightware LW 20 Lidar**, users must configure this menu as follows:

- Enable the desired lidar, in this case **Lidar 0** has been used
- Select the **SF11 Lidar** option from the drop-down menu
- Enter the address **102** in **decimal format**

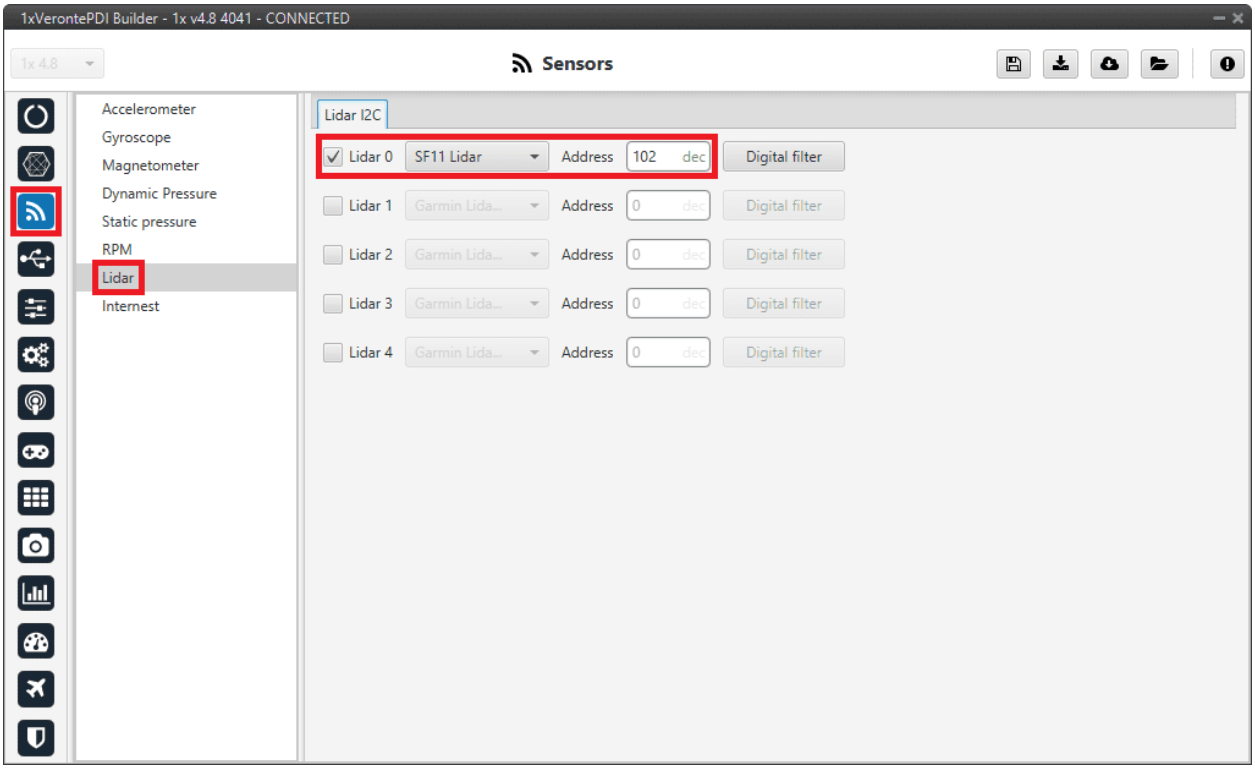


Fig. 41: **Lightware LW 20 Lidar**

3.7.1.1.3 Using lidar readings

Once the information provided by a lidar sensor is stored in a system variable as *Lidar Distance* via an ADC reading, I2C, serial or CAN, the user has to set how this data will be considered. Common uses are: to consider the lidar data as external sensor or to trigger an action based on a predefined event.

- **Altimeter configuration**: The following operation must be configured in the *Block Programs menu* to consider the lidar measurement as an EKF input.

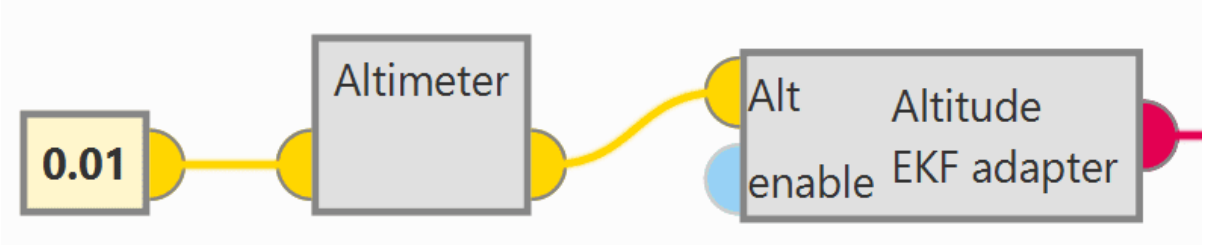


Fig. 42: **Altimeter connection in Block Programs**

The *Lidar Distance* variable where the lidar measurement is stored must be selected. In this example, *Lidar 1 Distance* has been used:

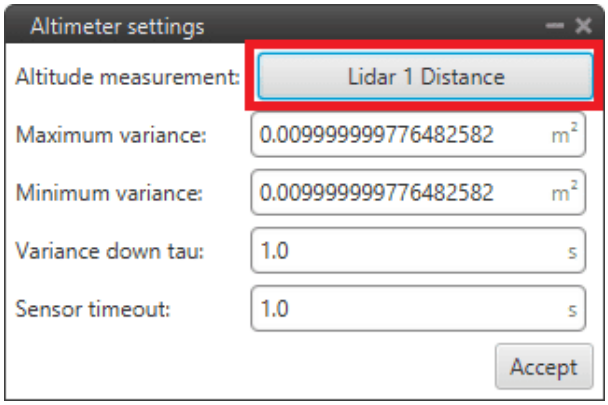


Fig. 43: **Altimeter sensor block configuration**

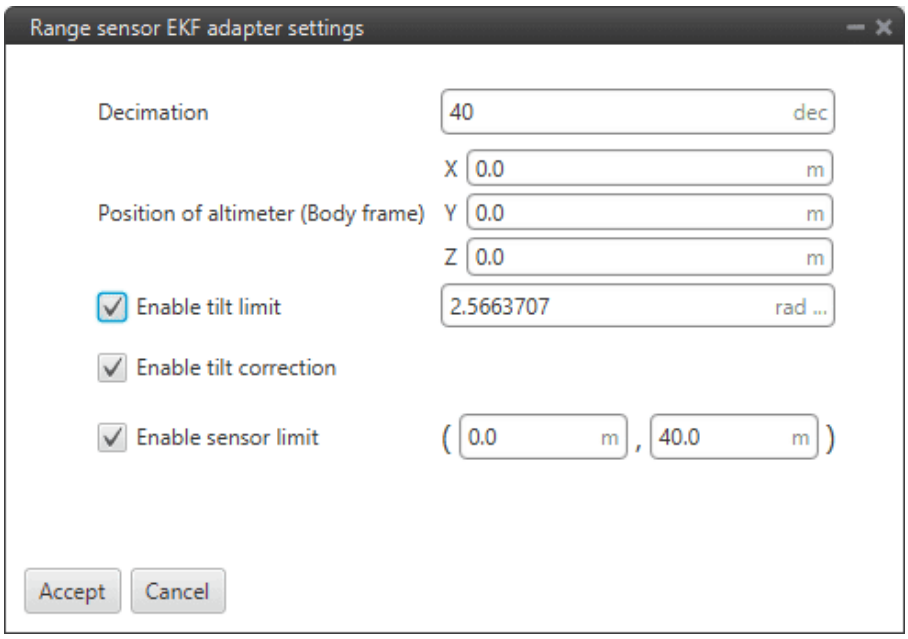


Fig. 44: **Altitude EKF adapter block configuration**

For more information on these blocks, see *Altimeter - Sensors blocks* and *Altitude EKF adapter - Navigation blocks* of **Block Programs** section.

- **Automation**: This automation will trigger a change of phase, Flare phase, when the aircraft is landing and at 5 m AGL.

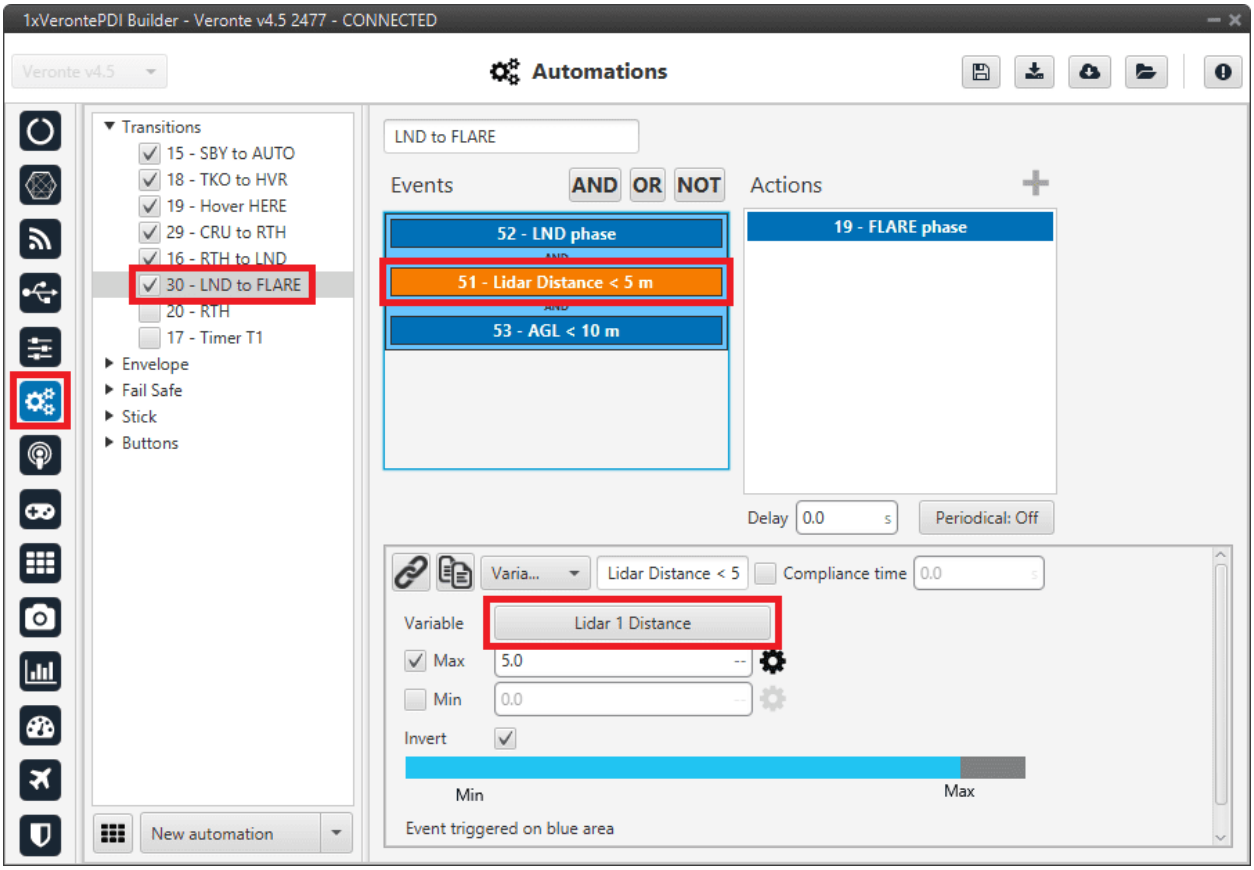


Fig. 45: **Lidar automation example**

For more information on automations, see *Automations* section of this manual.

3.7.1.2 Radar

Radar altimeters are common devices on aircrafts.

3.7.1.2.1 Ainstein CAN Radar

The following explanation corresponds to the integration of the **Ainstein CAN Radar**.

These settings will allow Autopilot 1x to read out via **CAN A** the radar altimeter reading, in particular **distance**.

Note: In the datasheet of the radar, the user can access the complete protocol of the device.

---

1. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect an **Input filter** producer to **Custom message 0**, in this example *Input filter 1* has been selected:

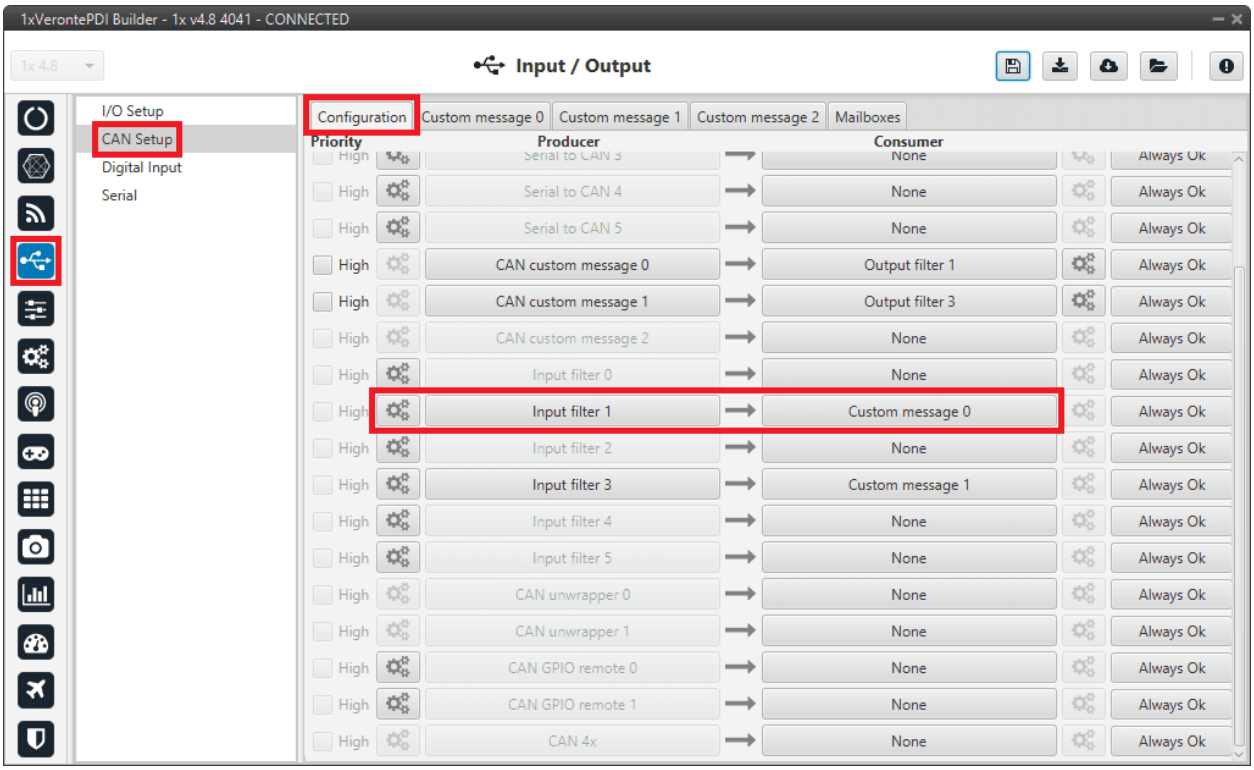


Fig. 46: **Ainstein CAN Radar - CAN Setup configuration**

Click on  to configure this Input filter to read from **CAN A**, with **Id 589826** and allow both types of messages to enter the input filter (since the radar altimeter uses **extended IDs**).

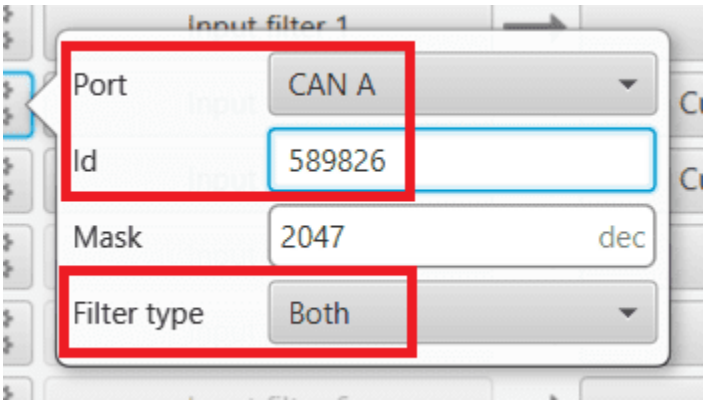


Fig. 47: **Ainstein CAN Radar - Input filter configuration**

2. After specifying that **Custom message 0** will receive the data from **CAN A**, go to **Mailboxes tab**.

   Configure a **CAN A mailbox** for **extended CAN ID message: 589826**:

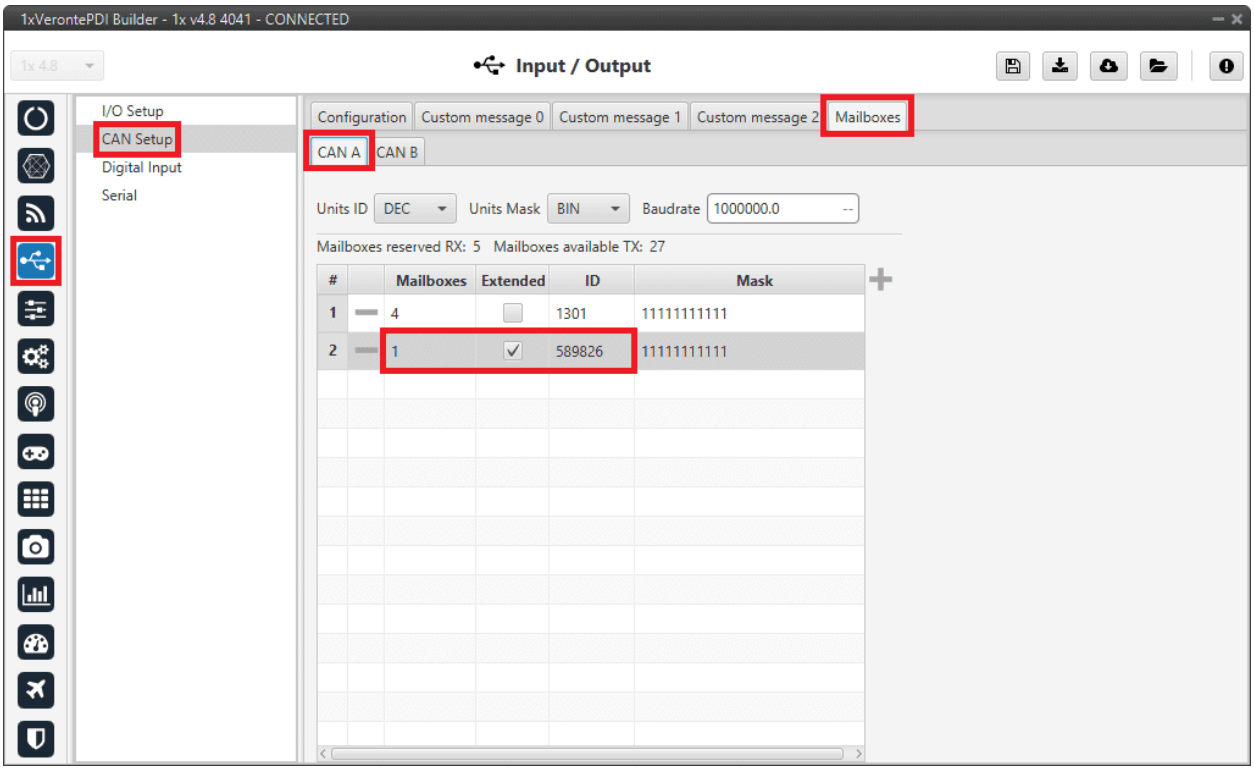


Fig. 48: **Ainstein CAN Radar - Mailboxes configuration**

3. Go to UI menu → Variables panel → **Real Vars tab**.

   Rename a User Variable that will be used to store the measurement read from the radar:

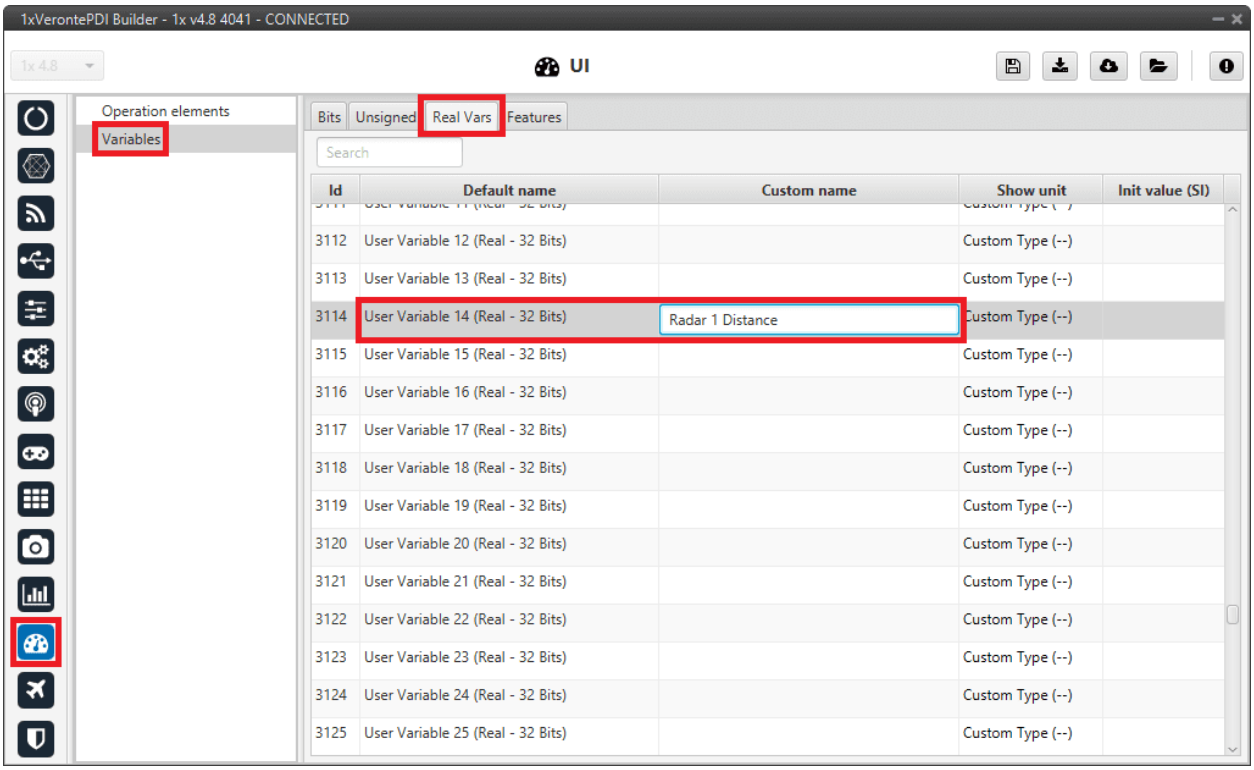


Fig. 49: **Ainstein CAN Radar - User Variable renamed**

4. Finally, go to Input/Output menu → CAN Setup panel → **Custom message 0 tab**.
    - Add a new message in **RX** with **extended ID 589826** and **Big endian**:

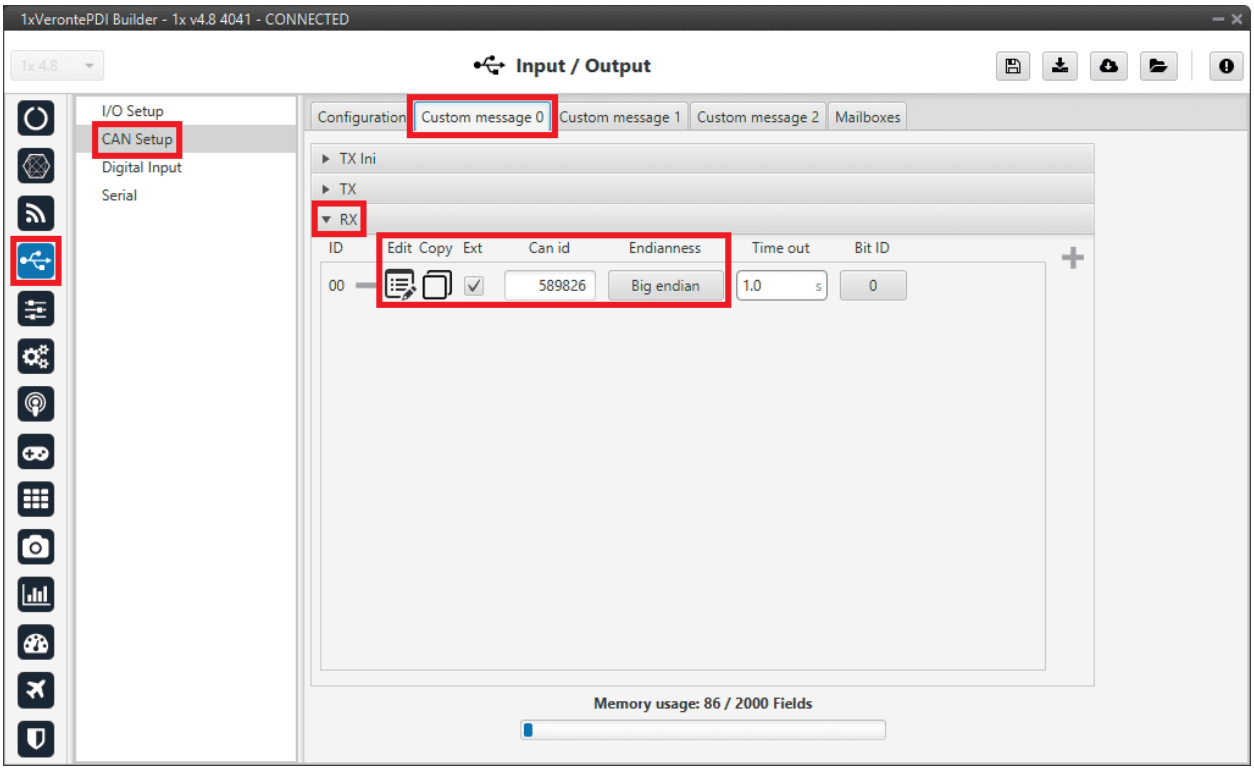


Fig. 50: **Ainstein CAN Radar - Custom message 0**

- Clicking on  , configure the reading of the message by setting the following values for the different parameters:
  - **Variable**: *Radar 1 Distance* (to store the received value in the user variable that has been renamed above)
  - **Compression**: Compress - Bits Unsigned
  - **Bits**: 16
  - **Encode - Min/Max**: 0.0/1.0
  - **Decode - Min/Max**: 0/100

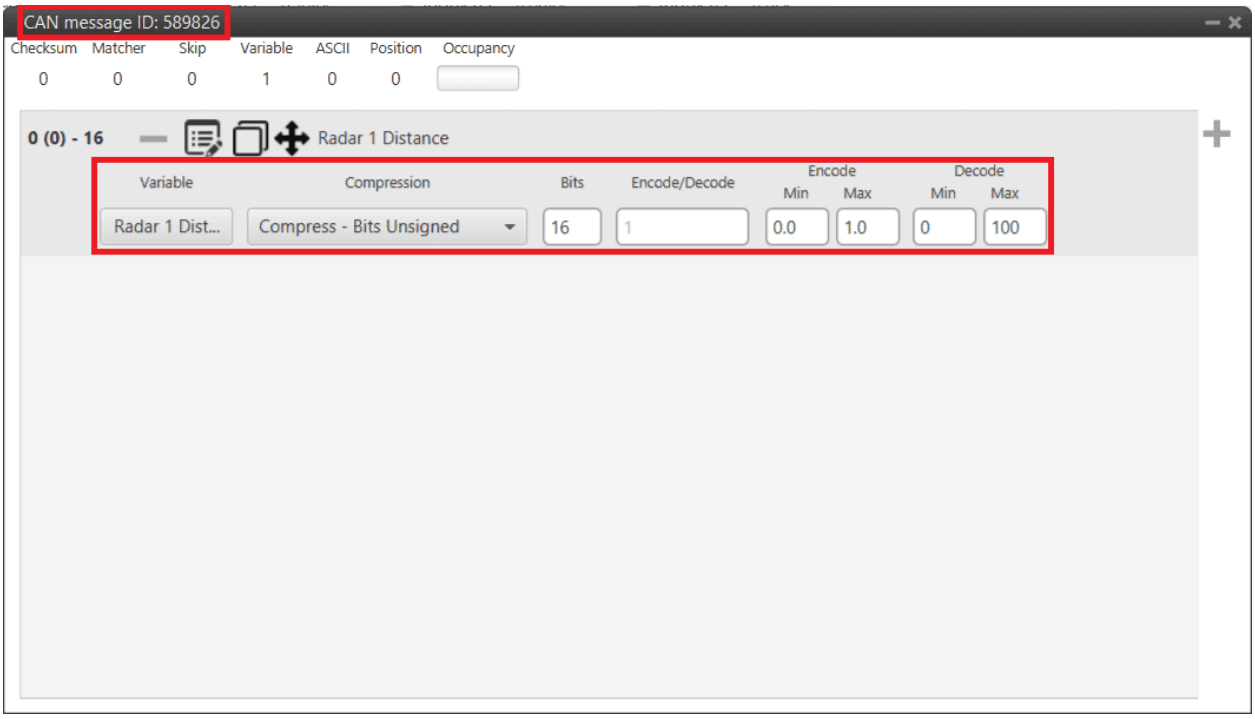


Fig. 51: **Ainstein CAN Radar - Custom message 0 configuration**

For more details on CAN configuration see *CAN Setup - Input/Output* section of this manual.

3.7.1.2.2 Smartmicro CAN Radar

The following explanation corresponds to the integration of the **Smartmicro CAN Radar**.

These settings will allow Autopilot 1x to read out via **CAN A** the radar altimeter readings, in particular **AGL and vertical speed**.

Note: In the datasheet the user can access the complete protocol of the device.

---

1. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect an **Input filter** to **Custom message 0**, in this example *Input filter 2*.

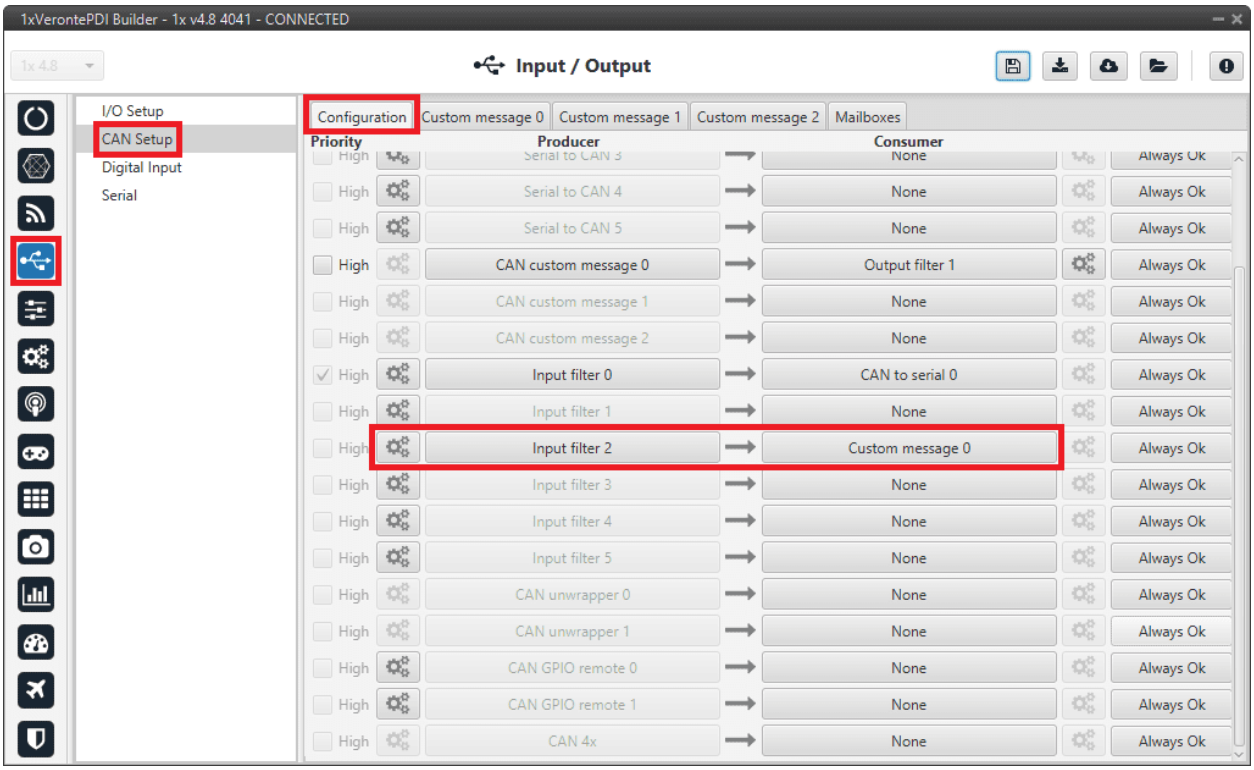


Fig. 52: **Smartmicro CAN Radar - CAN Setup configuration**

Configure this Input filter to read from **CAN A**, with **Id 1872**

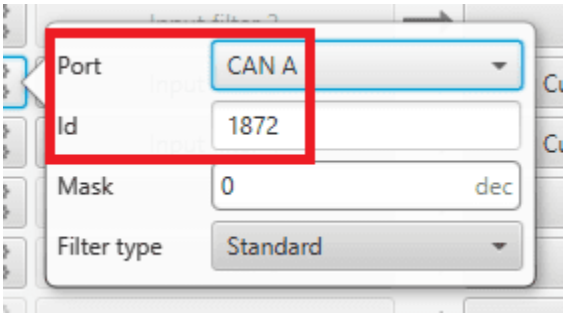


Fig. 53: **Smartmicro CAN Radar - Input filter configuration**

2. After specifying that **Custom message 0** will receive the data from **CAN A**, go to **Mailboxes tab**.

   Configure the **CAN A baudrate** and some mailboxes for **CAN ID message: 1872**:

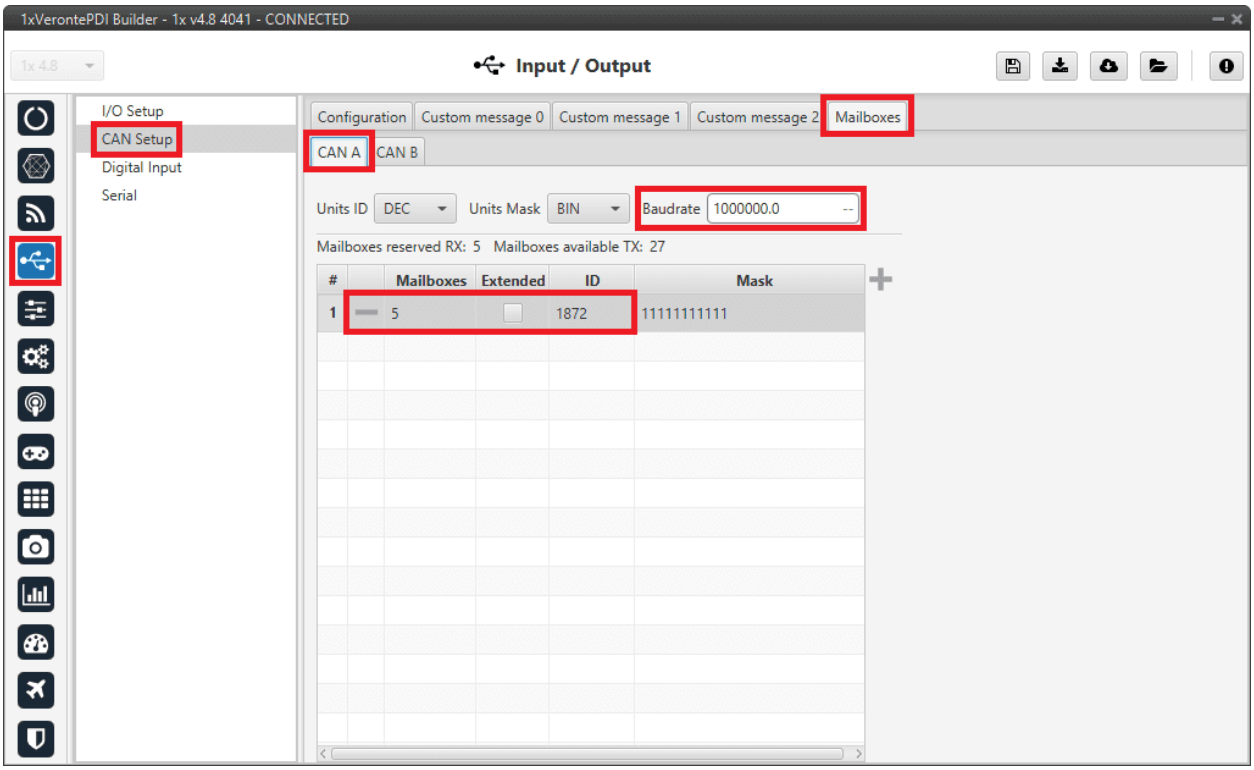


Fig. 54: **Smartmicro CAN Radar - Mailboxes configuration**

3. Go to **Custom message 0 tab**.
    - Add a new message in **RX** with **ID 1872**.

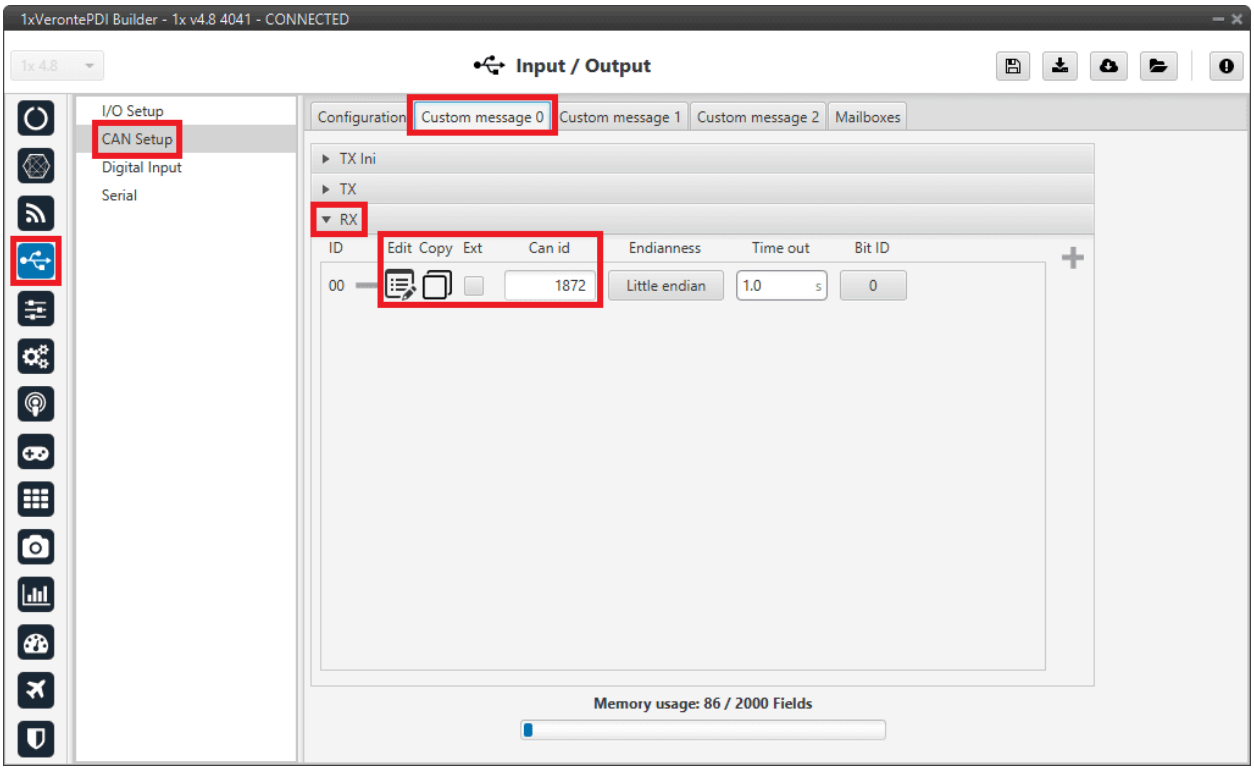


Fig. 55: **Smartmicro CAN Radar - Custom message 0**

- Define the content of the incoming message as desired.

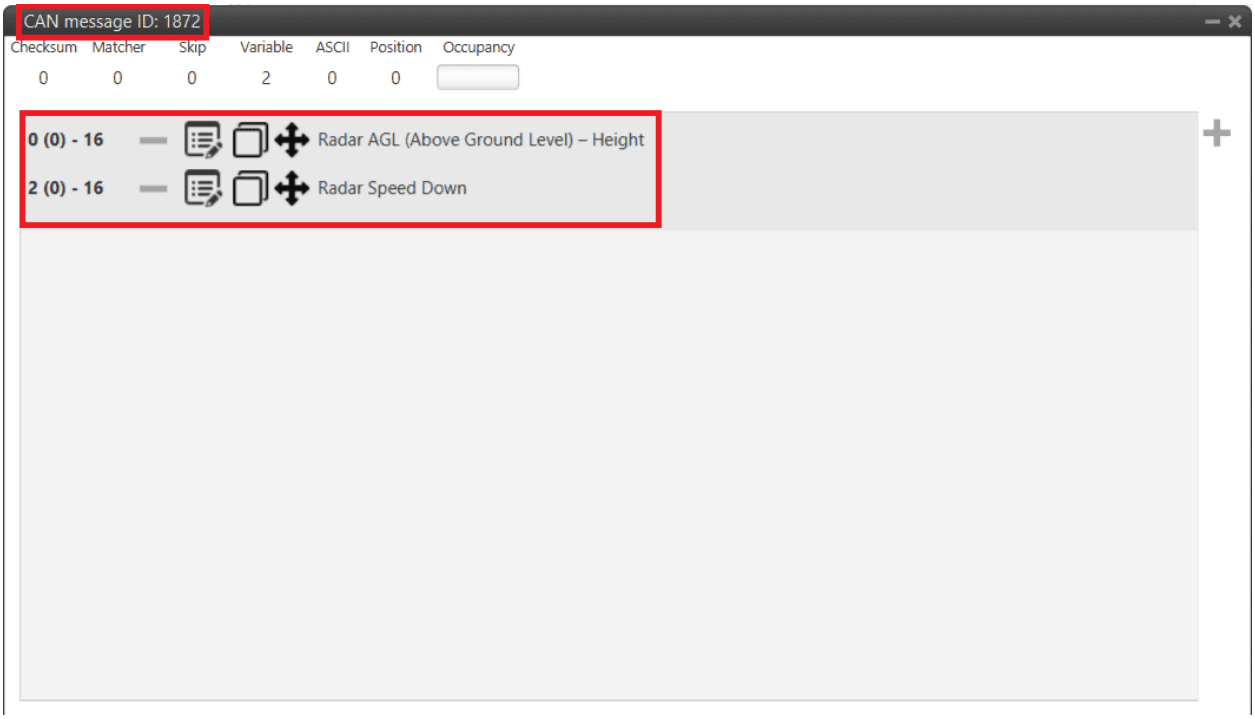


Fig. 56: **Smartmicro CAN Radar - Custom message 0 configuration**

For more details on CAN configuration see *CAN Setup - Input/Output* section of this manual.

Note: CAN ID messages and messages content will change for different Radar altimeters. Check the documentation of your device for further details.

3.7.2 External sensors

Veronte Autopilot 1x can be integrated with any external sensor that shares the communication interface.

External sensors can be configured to be considered as part of the sensors fusion.

3.7.2.1 High Speed Pitot Sensor

Once the hardware has been installed according to High Speed Pitot Sensor section of **1x Hardware Manual**, the value can be monitored in **1x PDI Builder** by using the variables ADC0 to ADC4.

Note: For pin *ANALOG_0* the correspondent ADC variable in **1x PDI Builder** is *ADC0*, for *ANALOG_1* is *ADC1* and so on.

Read the following steps to configure it:

1. Go to **Block Programs menu**  to create a new program where the blocks configuration for reading the sensor measurements will be built.

   To know the basics about **Block Programs**, read the *Block Programs* section of this manual.

2. The configuration of this new program should be as shown in the following image:

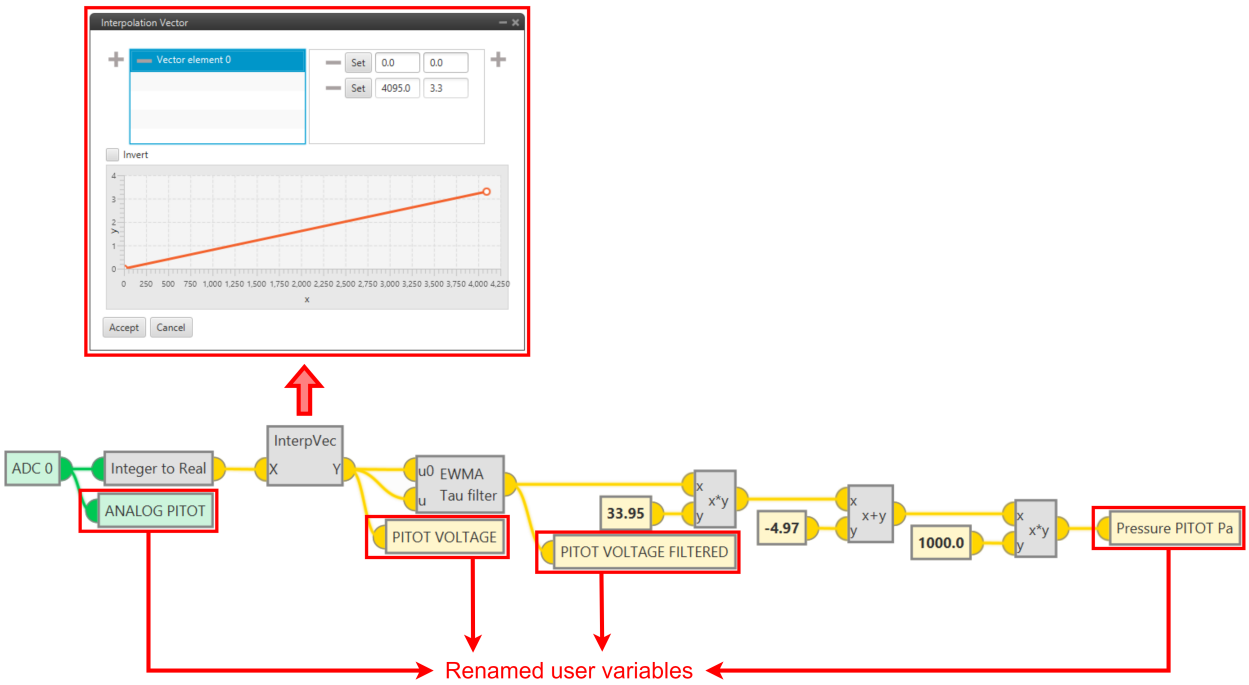


Fig. 57: **High Speed Pitot Sensor - Block program**

Note:

- The **EWMA filter** should have a value of **0.75** for Tau.
- This block program shows the configuration equivalent to the following equation: $\Rightarrow Pressure\,PITOT(Pa) = 1000 \cdot Pressure(KPa) \Rightarrow Pressure(KPa) = 33.95 \cdot PITOT\ VOLTAGE\ FILTERED - 4.97$

According to the previous image, pressure measurements are written in the variable **Pressure PITOT Pa** in pascals.

The ADC variable to select depends on the ANALOG pin used for receive data from **High Speed Pitot Sensor**. On the previous image, the ADC variable selected (*ADC 0*) reads data from pin 38.

For more information on Pinout, read the Pinout - Hardware Installation section of the **1x Hardware Manual**.

3. Go to Sensors menu → Dynamic Pressure panel → **Sensor tab**.

   Finally, enable a "Decimal var sensor" and assign it the previously calculated variable in pascals as the dynamic pressure measurement.

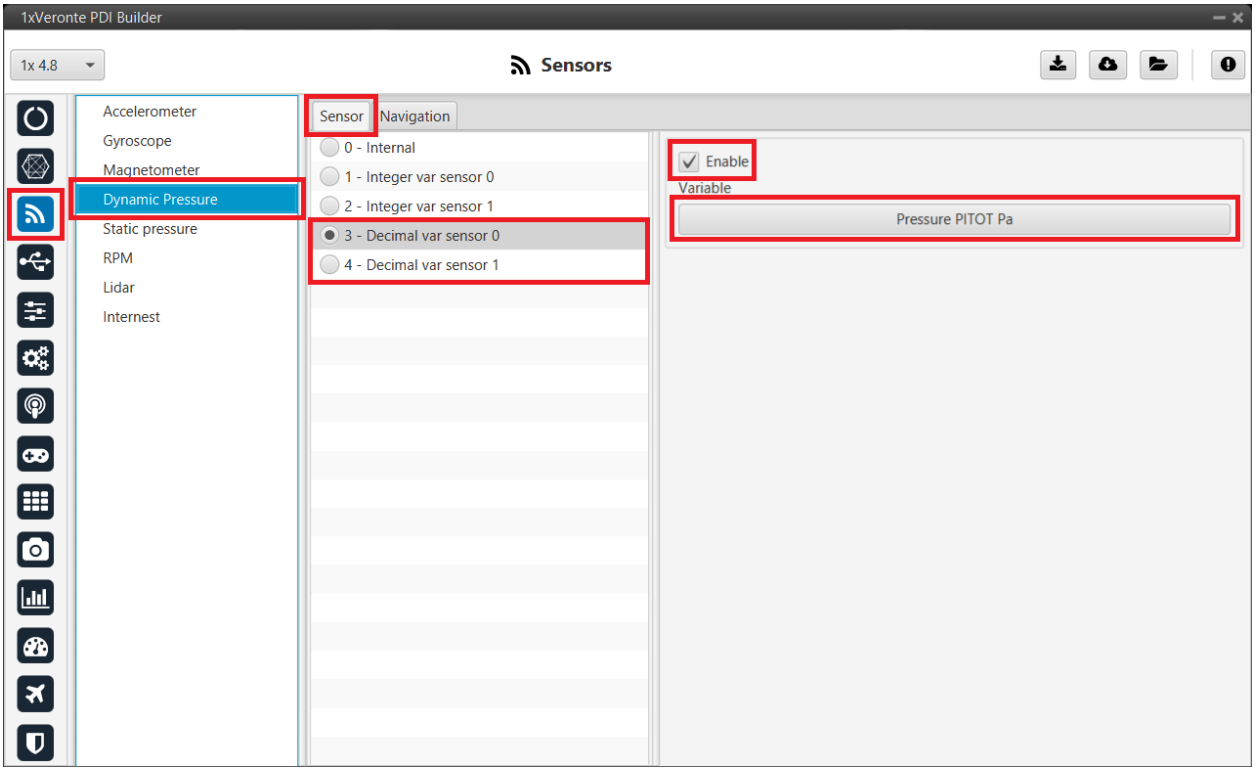


Fig. 58: **High Speed Pitot Sensor - Decimal var sensor configuration**

Note: The sensor is extremely sensitive at speed 0 due to its high resolution.

3.7.2.2 LM335 with Autopilot 4x

Once **LM335** sensor is wired and connected to **Autopilot 1x** or **4x** (according to Temperature sensor LM335 - Integration examples section of the **4x Hardware Manual**), the value can be monitored in **1x PDI Builder** by using the variables ADC0 to ADC4.

Note: For pin *ANALOG_0* the correspondent ADC variable in **1x PDI Builder** is *ADC0*, for *ANALOG_1* is *ADC1* and so on.

Read the following steps to configure a **Veronte Autopilot 1x**:

1. Go to Connections menu → **ADC 1 panel** (This is only an example, the user must select the ADC pin where the signal is connected).

   Click on 'Create new program':

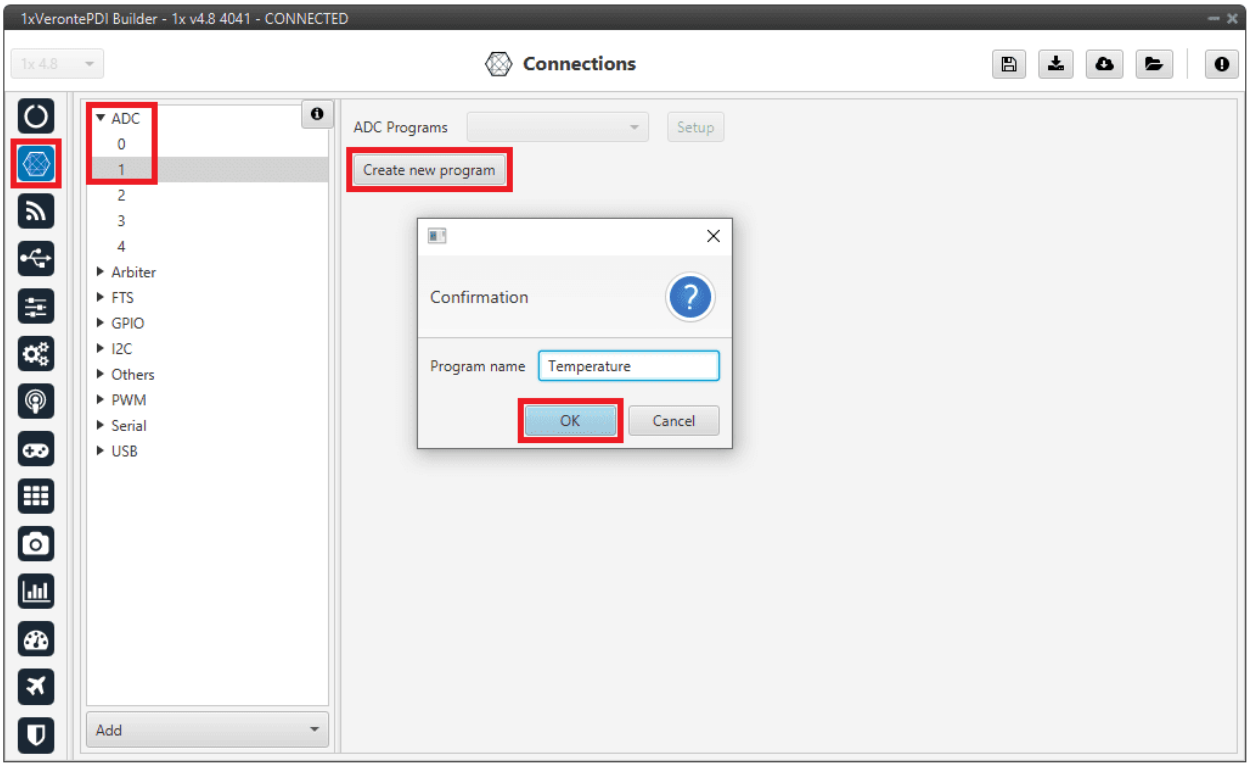


Fig. 59: **LM335 sensor - Create ADC program**

2. Go to **Block Programs menu** → **Launch Editor**.

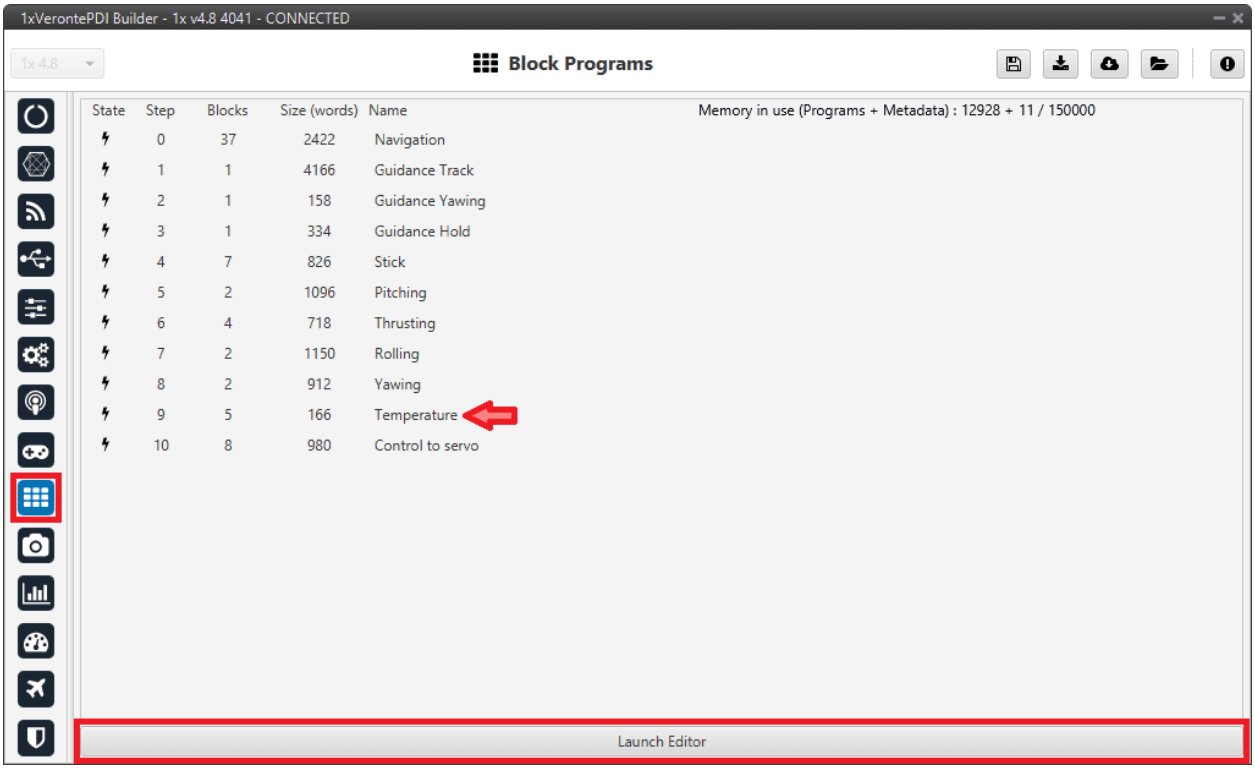


Fig. 60: **LM335 sensor - Block Programs**

Configure the following operation (for more information about blocks, read *Block Programs* section of this manual):

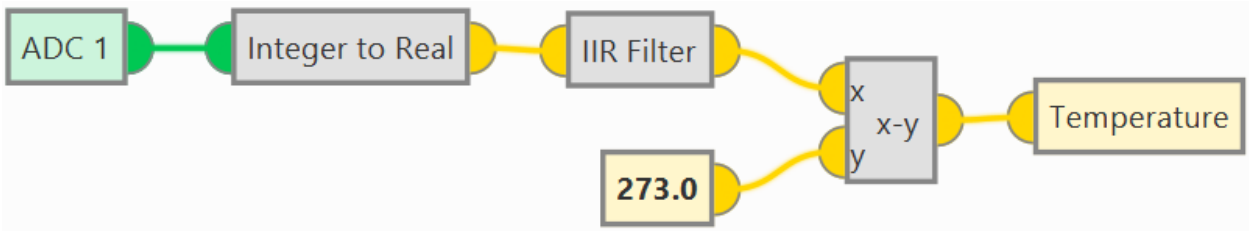


Fig. 61: **LM335 sensor - Block Programs operation**

Note:

- The **Temperature** variable is an **User Variable** which has been renamed.
- The equation to obtain temperature (in ºC) from voltage is: $T = Vout \cdot 100 - 273$. Nonetheless, in the blocks program, the input signal is not multiplied by 100, since **ADC** expresses the voltage in hundredths.

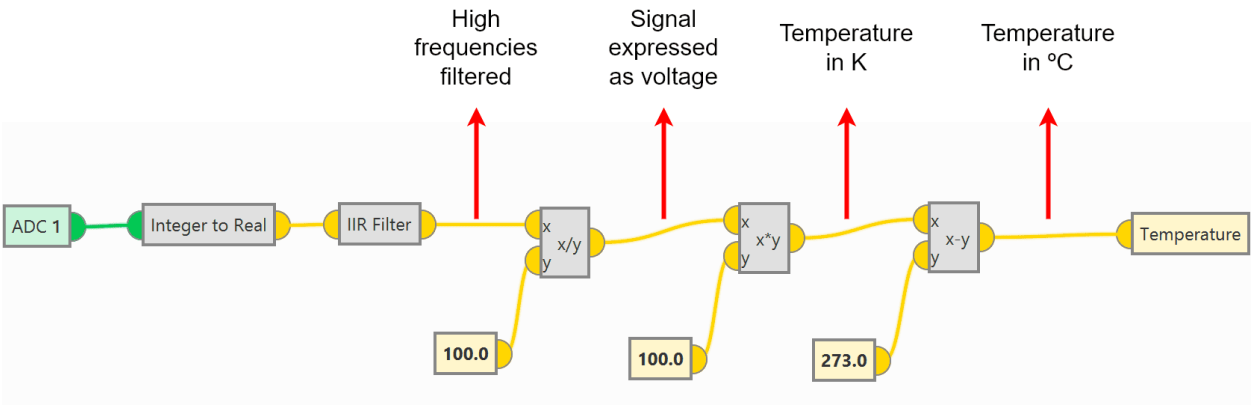


Fig. 62: **LM335 sensor - Block Programs operation detailed**

3. The **IIR Filter block** requires the following configuration, where the **column B** has **20 coefficients** (from 0 to 19) with **value 1**.

   Click on **Apply** to save the changes.

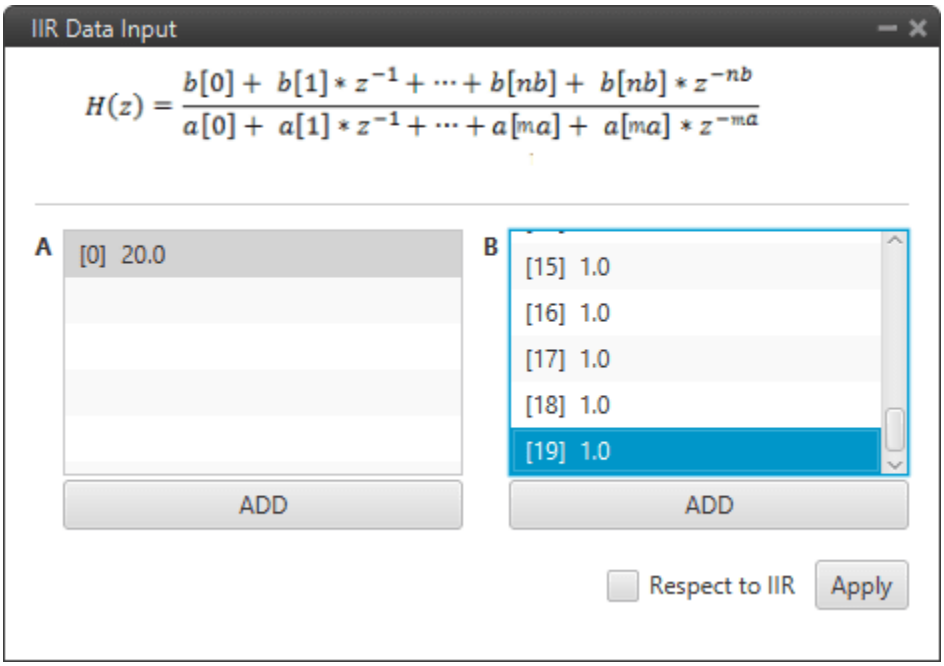


Fig. 63: **LM335 sensor - IIR Filter block configuration**

4. Save the configuation in the **Autopilot 1x**.

After implementing the operation, the variable *Temperature* (User variable renamed) will represent the temperature (in °C) measured by the **LM335** sensor.

Tip: With **Veronte Ops** it is possible to check that the sensor is working correctly.

Fig. 64: **LM335 sensor - Temperature value in Veronte Ops**

3.7.2.3 Magnetometer Honeywell HMR2300

3.7.2.3.1 RS-232

Magnetometer Honeywell HMR2300 can be connected via **RS-232** (serial interface) in accordance with the manufacturer's specifications and following the Pinout - Hardware Installation section of the **1x Hardware Manual**.

The following steps explain how to configure Veronte Autopilot 1x to integrate this external magnetometer:

1. Go to Input/Output menu → Serial panel → **RS232 tab**.

   Configure the serial port parameters:

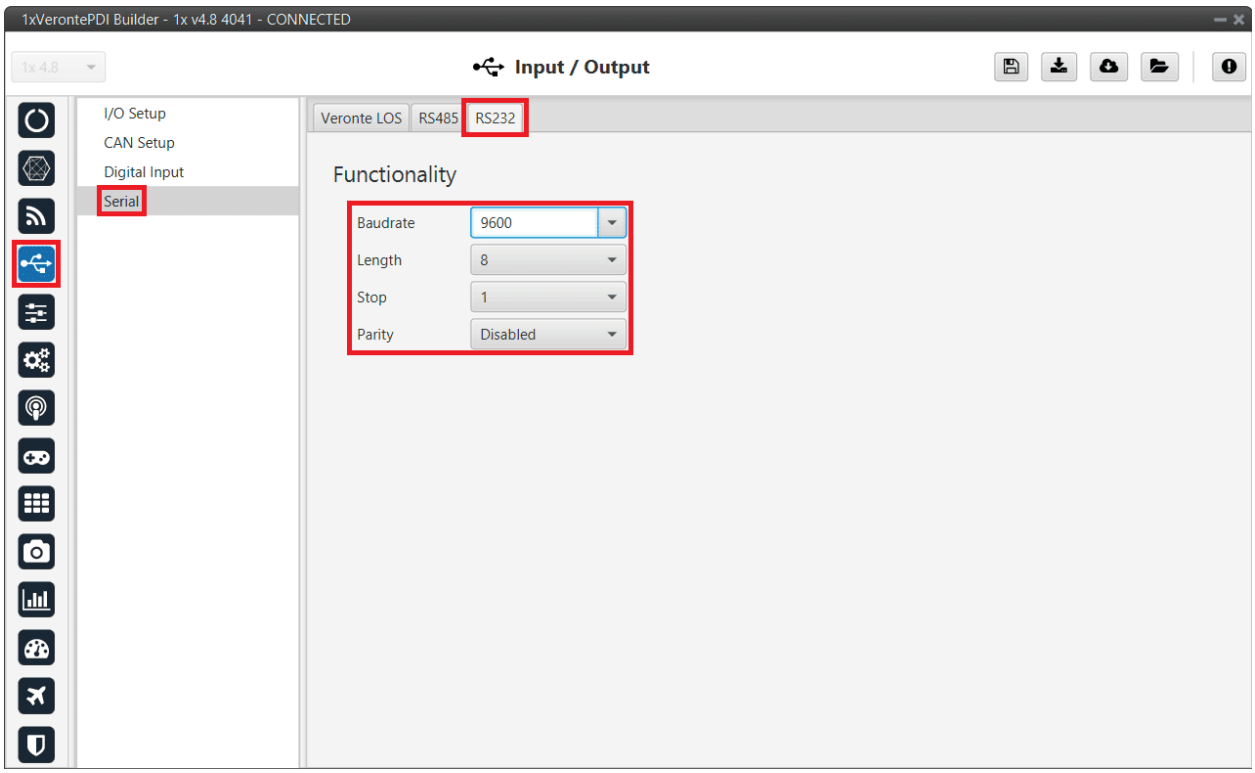


Fig. 65: **Magnetometer Honeywell HMR2300 - RS232 port configuration**

2. Go to **Block Programs menu**.

- Create a program to make the necessary connection to the sensor blocks.

  Usually the user has a **Navigation program** where the sensor blocks are implemented.

- Configure the *Magnetometer sensor* block selecting **External HMR2300**.

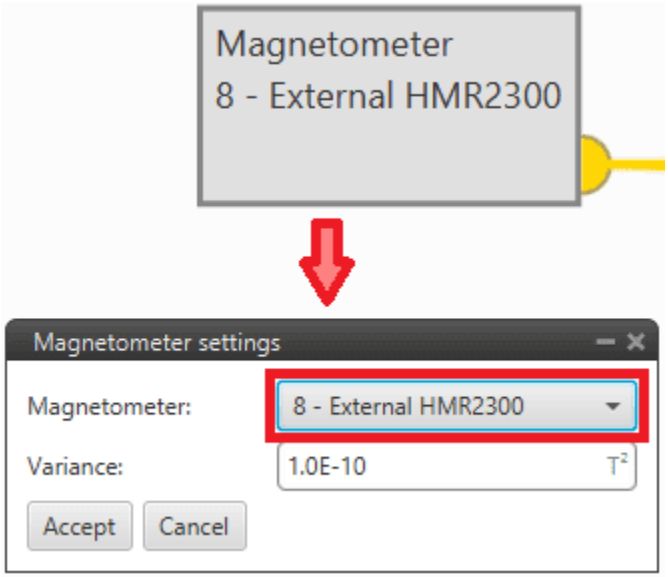


Fig. 66: **Magnetometer Honeywell HMR2300 - Magnetometer sensor block configuration**

For more information on this block, see *Magnetometer - Sensors blocks* of **Block Programs** section.

3. Go to Input/Output menu → **I/O Setup panel**.

   Bidirectionally connect the **RS232 Producer** to the **External HMR2300 magnetometer Consumer**:

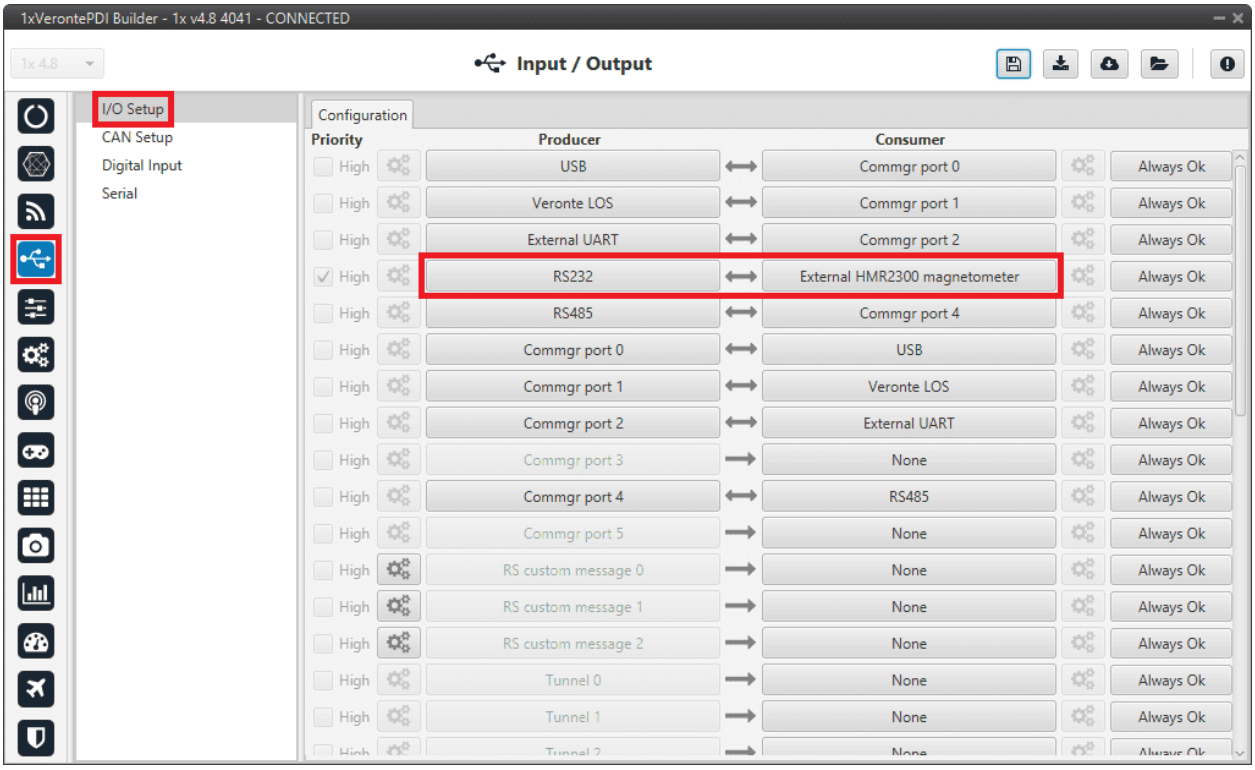


Fig. 67: **RS232 ↔ External HMR2300 magnetometer**

Then, the **External HMR2300 magnetometer Producer** should be automatically connected to the **RS232 Consumer**:

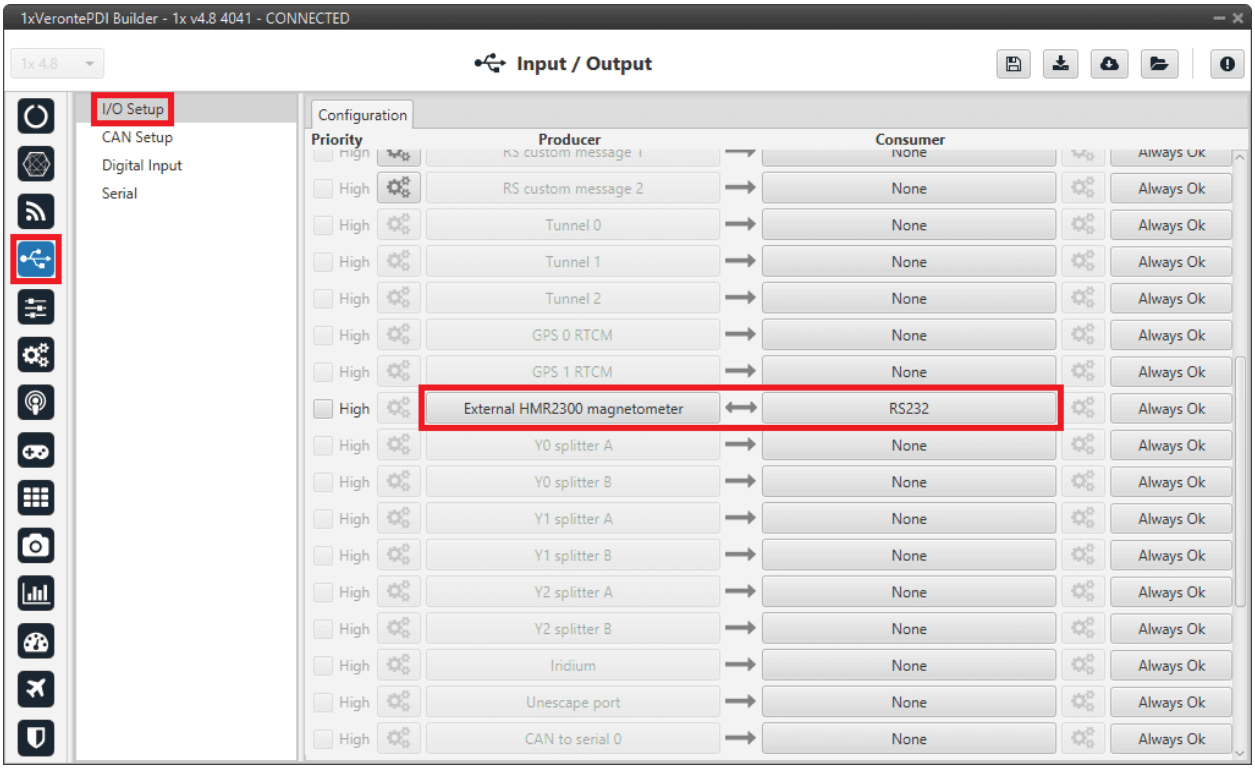


Fig. 68: **External HMR2300 magnetometer ↔ RS232**

For more information on the Magnetometer Honeywell HMR2300, check out the datasheet: Smart Digital Magnetometer HMR2300.

3.7.2.3.2 RS-485

Magnetometer Honeywell HMR2300 can be connected via **RS-485** (serial interface) in accordance with the manufacturer's specifications and following the Pinout - Hardware Installation section of the **1x Hardware Manual**.

Follow the next steps to establish a correctly communication between Honeywell HMR2300 magnetometer and Veronte Autopilot 1x via **RS-485**:

1. Connect the Honeywell HMR2300 magnetometer via USB to the PC (with a USB-RS485 conversor).
2. The configuration to establish communication must be:
   - In **binary** mode
   - With **continuous fowarding**
   - **ID = 00**

   To configure the magnetometer in this way, the following commands must be sent to it:
   - `*99Q`: This command reads the default values, including the device ID.
   - `*00WE`: This enables writing.
   - `*ddID=00`: Changes the ID to 00, where **dd** is the device ID obtained with the first command.
   - `*00WE`: Enables writing.

- `*00B`: Binary mode.
- `*00C`: Continuous send mode.
- `*00WE`: Enables writing.
- `*00SP`: Finally, this command saves the configuration in EEPROM.

3. Autopilot can now be configured to communicate via **RS485** with the magnetometer Honeywell HMR2300.

   The configuration to be carried out is very similar to that described above for communication with the magnetometer via *RS-232*:

   - Instead of configuring the 232 serial port, the **485 serial port is configured**.
   - And, the **bidirectional connection** must be made between the **RS485 port and External HMR2300 magnetometer** and not with the RS232 port.

For more information on the Magnetometer Honeywell HMR2300, check out the datasheet: Smart Digital Magnetometer HMR2300.

3.7.2.4 MEX as Magnetometer Honeywell HMR2300

MEX can be used as an **external magnetometer Honeywell HMR2300** connected to Veronte Autopilot 1x via serial or CAN interfaces.

Important: Both **MEX** and **Autopilot 1x** devices must be configured to conduct this integration, therefore, the following explanations are consequent to the MEX configuration detailed in the MEX as external magnetometer for Autopilot 1x - Integration examples section of **MEX PDI Builder** manual.

Communication via **serial** and **CAN** interfaces are explained separately.

3.7.2.4.1 Serial

For this serial connection, first check the Pinout - Hardware Installation section of the **1x Hardware Manual**.

The following steps explain how to configure **Veronte Autopilot 1x** to integrate MEX as an external magnetometer via serial:

1. Go to Input/Output menu → Serial panel → **RS232/RS485 tab**.

   Configure the serial port parameters:

   Note: This is an example of RS-232 connection, if MEX is connected via RS-485, configure the **RS485 serial tab**.

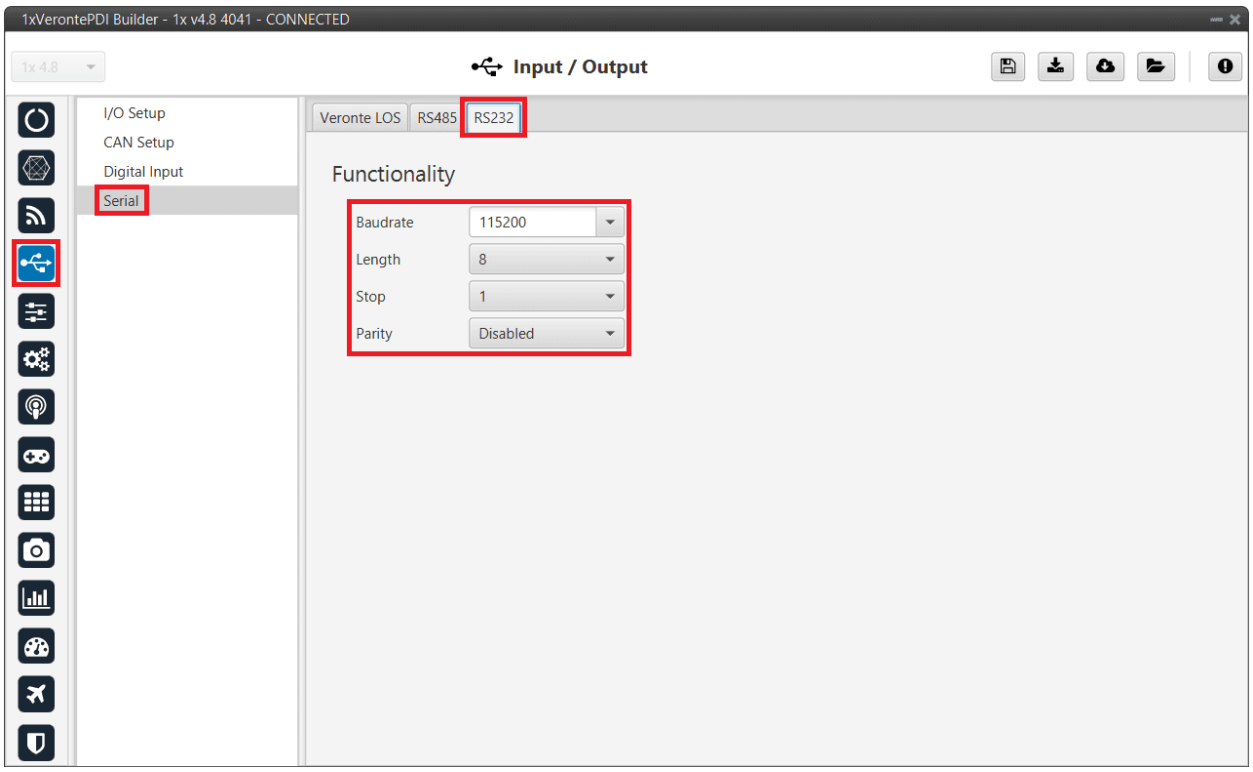


Fig. 69: **MEX as Magnetometer Honeywell HMR2300 - RS232 port configuration**

2. Go to **Block Programs menu**.
   - Create a program to make the necessary connection to the sensor blocks.

     Usually the user has a **Navigation program** where the sensor blocks are implemented.
   - Configure the *Magnetometer sensor* block selecting **External HMR2300**.

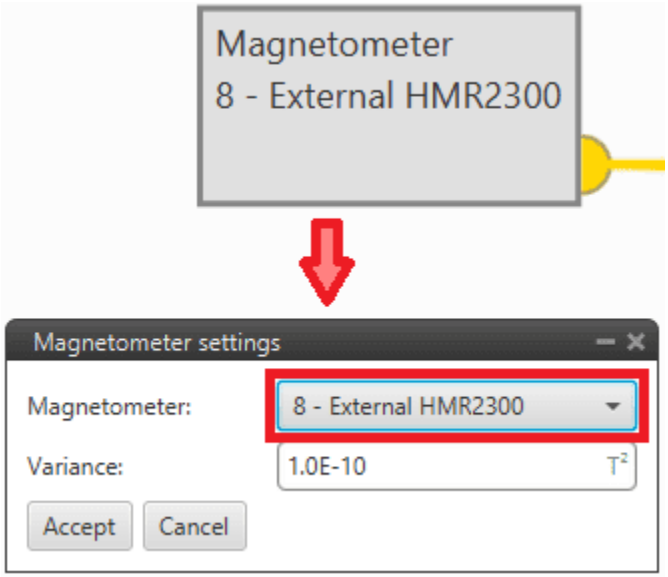


Fig. 70: **MEX as Magnetometer Honeywell HMR2300 - Magnetometer sensor block configuration**

For more information on this block, see *Magnetometer - Sensors blocks* of **Block Programs** section.

3. Go to Input/Output menu → **I/O Setup panel**.

   Connect the **RS232 Producer** to the **External HMR2300 magnetometer Consumer**:

Note: If the user connects the MEX via RS-485, connect the RS485 Producer instead of RS232 Producer.

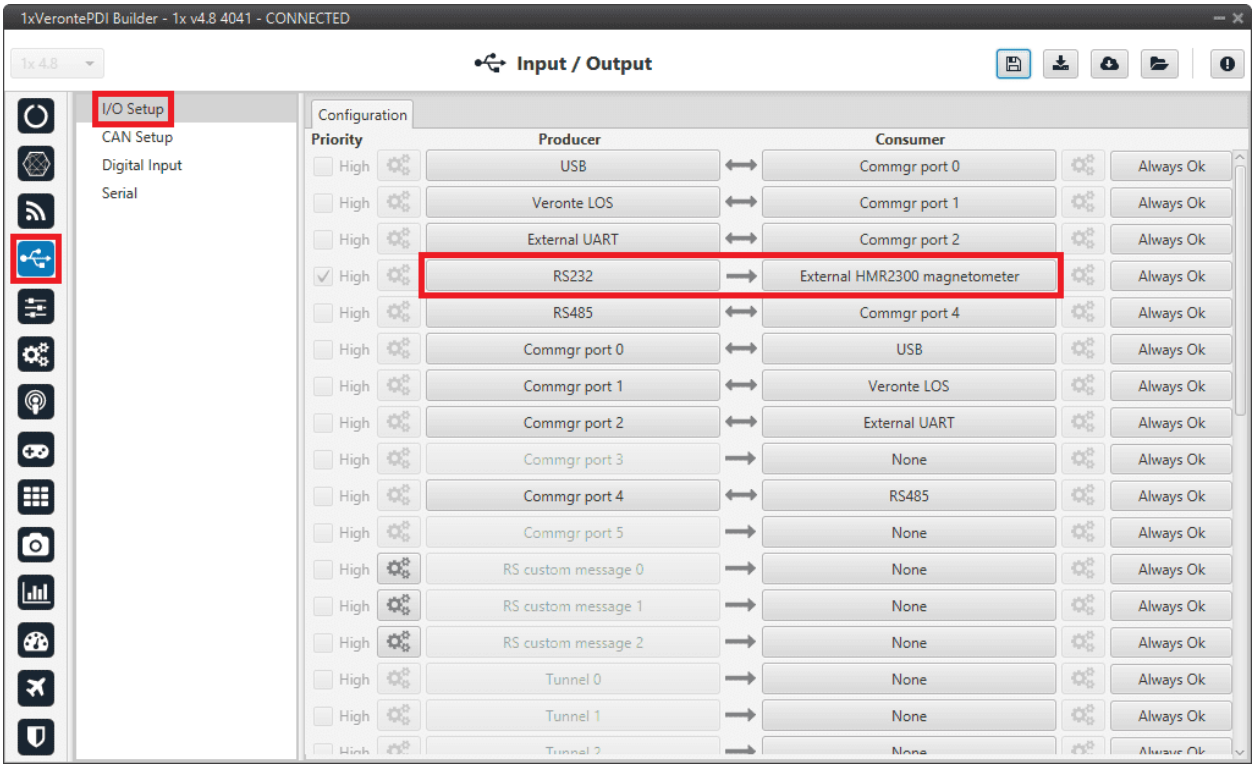


Fig. 71: **RS232 → External HMR2300 magnetometer**

3.7.2.4.2 CAN

The following steps explain how to configure **Veronte Autopilot 1x** to integrate MEX as an external magnetometer via CAN:

1. Go to **Block Programs menu**.
   - Create a program to make the necessary connection to the sensor blocks.

     Usually the user has a **Navigation program** where the sensor blocks are implemented.
   - Configure the *Magnetometer sensor* block selecting **External HMR2300**.

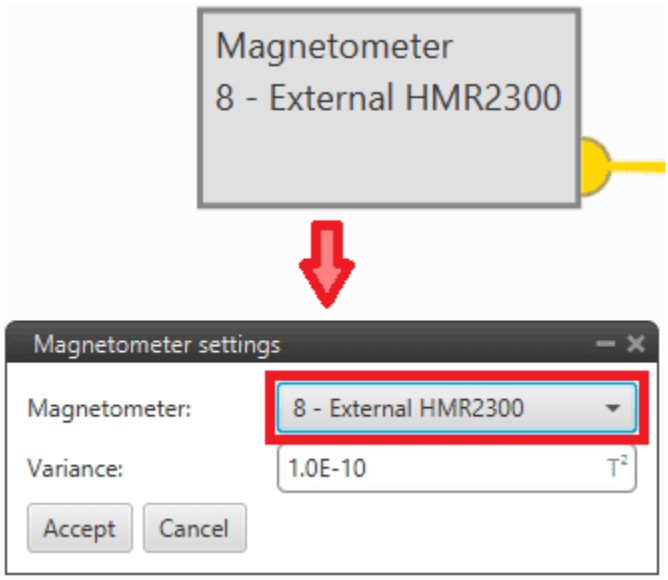


Fig. 72: **MEX as Magnetometer Honeywell HMR2300 - Magnetometer sensor block configuration**

For more information on this block, see *Magnetometer - Sensors blocks* of **Block Programs** section.

2. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Configure the required **mailboxes** to receive a message with the **CAN ID that matches the corresponding MEX configuration**.

Note: In this example, 4 mailboxes are added for **CAN A** with **ID 1301**, according to the MEX configuration carried out in CAN - MEX as external magnetometer for Autopilot 1x in the Integration examples section of **MEX PDI Builder** manual.

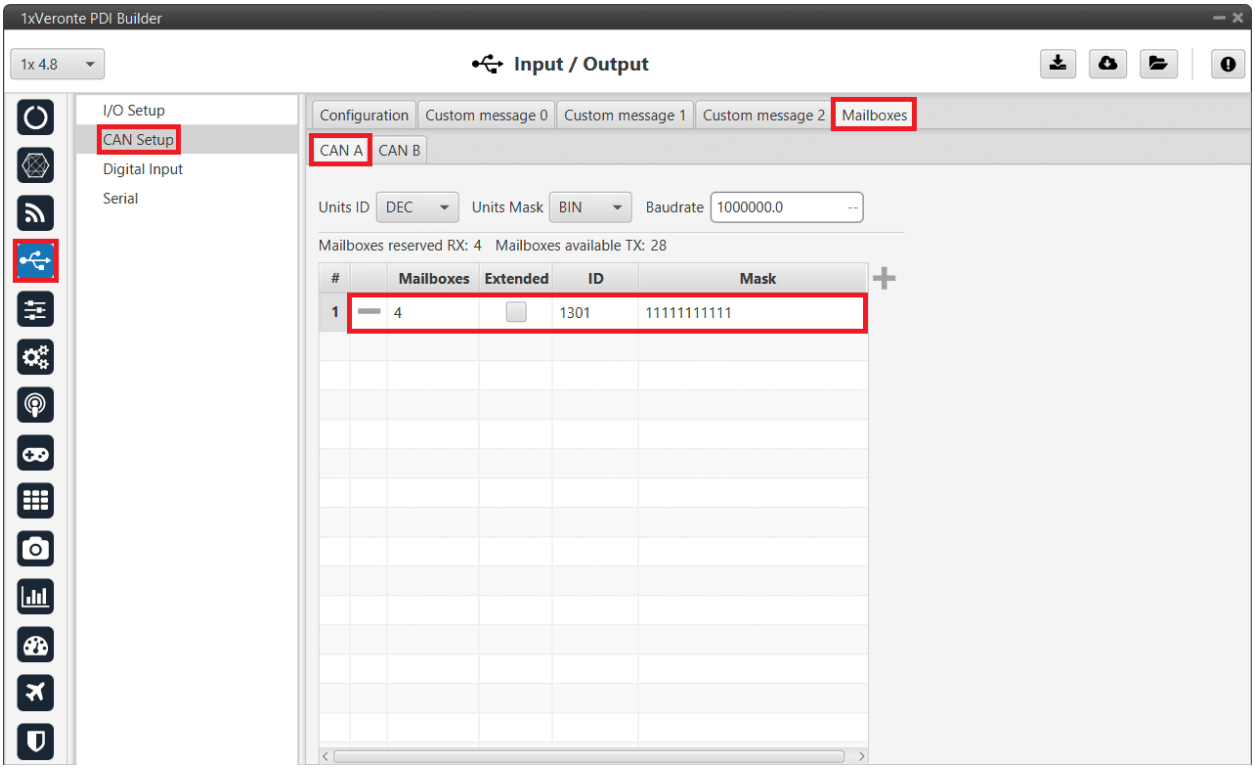


Fig. 73: **MEX as Magnetometer Honeywell HMR2300 - CAN Mailboxes configuration**

3. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect an **Input filter** Producer to a **CAN to serial** Consumer.

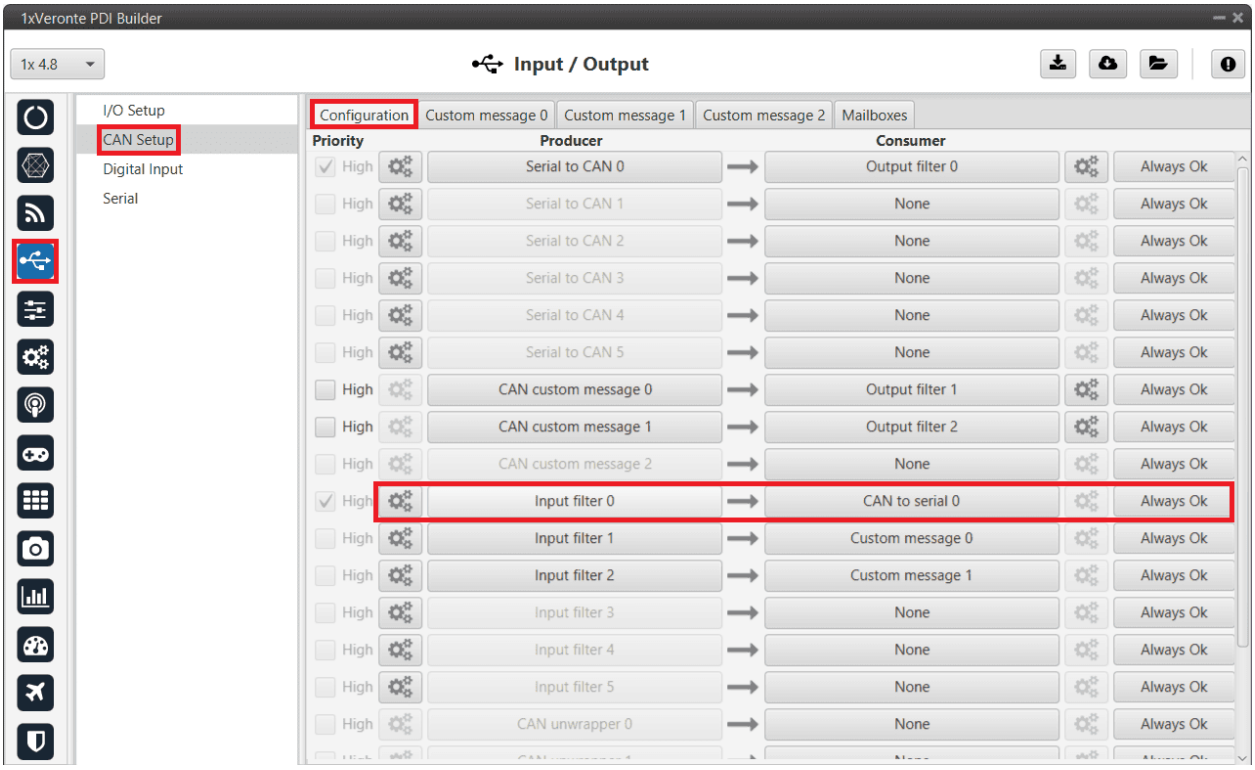


Fig. 74: **MEX as Magnetometer Honeywell HMR2300 - CAN Setup configuration**

Click on  of the selected Input filter and configure it with the **CAN port** and **CAN ID** that **matches those of the corresponding MEX configuration**.

Note: This example is configured according to the MEX configuration carried out in CAN - MEX as external magnetometer for Autopilot 1x in the Integration examples section of **MEX PDI Builder** manual.

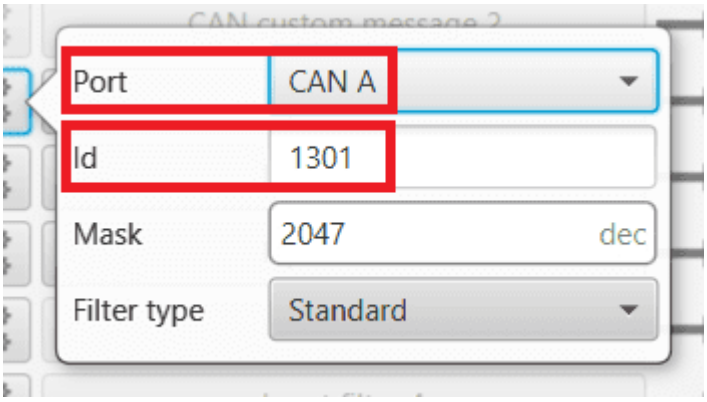


Fig. 75: **MEX as Magnetometer Honeywell HMR2300 - Input filter configuration**

4. Go to Input/Output menu → I/O Setup panel → **Configuration tab**.

   Connect a **CAN to serial** Producer to the **External HMR2300 magnetometer** Consumer.

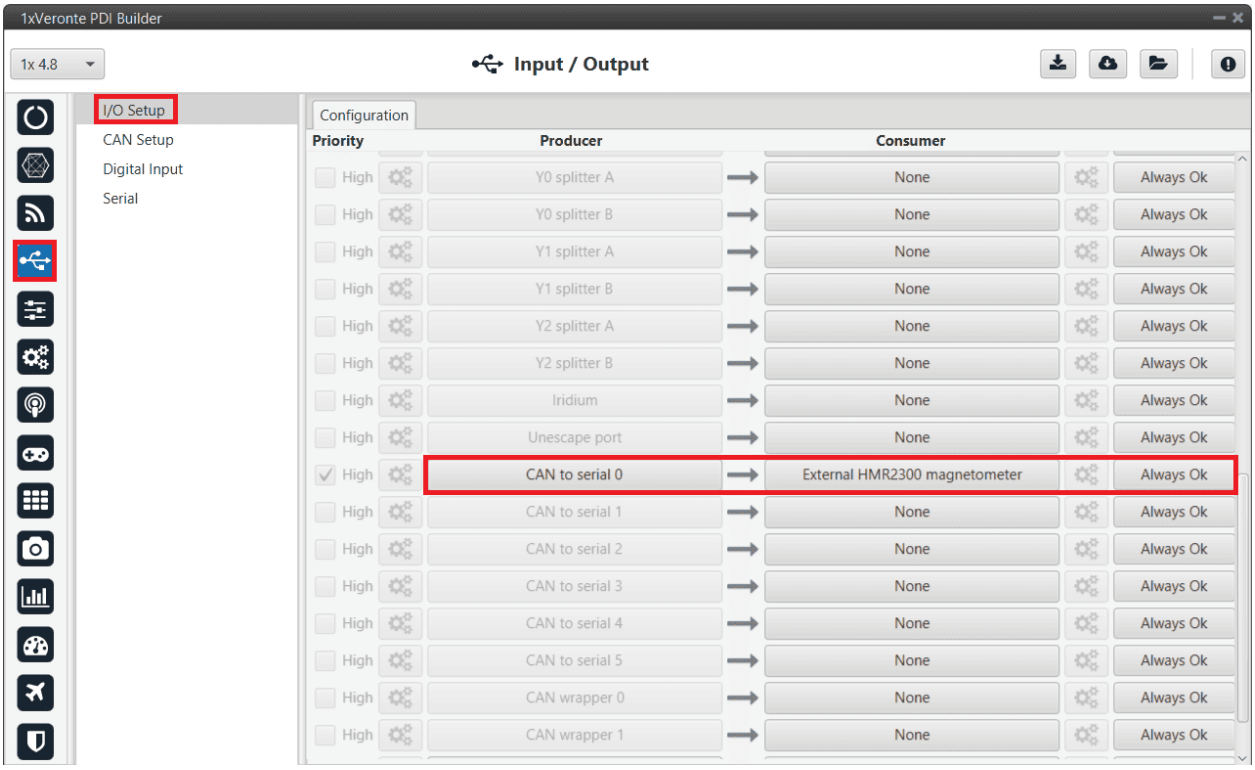


Fig. 76: **MEX as Magnetometer Honeywell HMR2300 - I/O Setup configuration**

3.7.2.5 OAT Sensor

Once OAT sensor is connected to Autopilot 1x, the value can be monitored in **1x PDI Builder** by using the variables ADC0 to ADC4.

Note: For pin *ANALOG_0* the correspondent ADC variable in **1x PDI Builder** is *ADC0*, for *ANALOG_1* is *ADC1* and so on.

Read the following steps to configure it:

1. Go to Connections menu → **ADC 0 panel** (This is only an example, the user must select the ADC pin where the signal is connected).

   Click on 'Create new program':

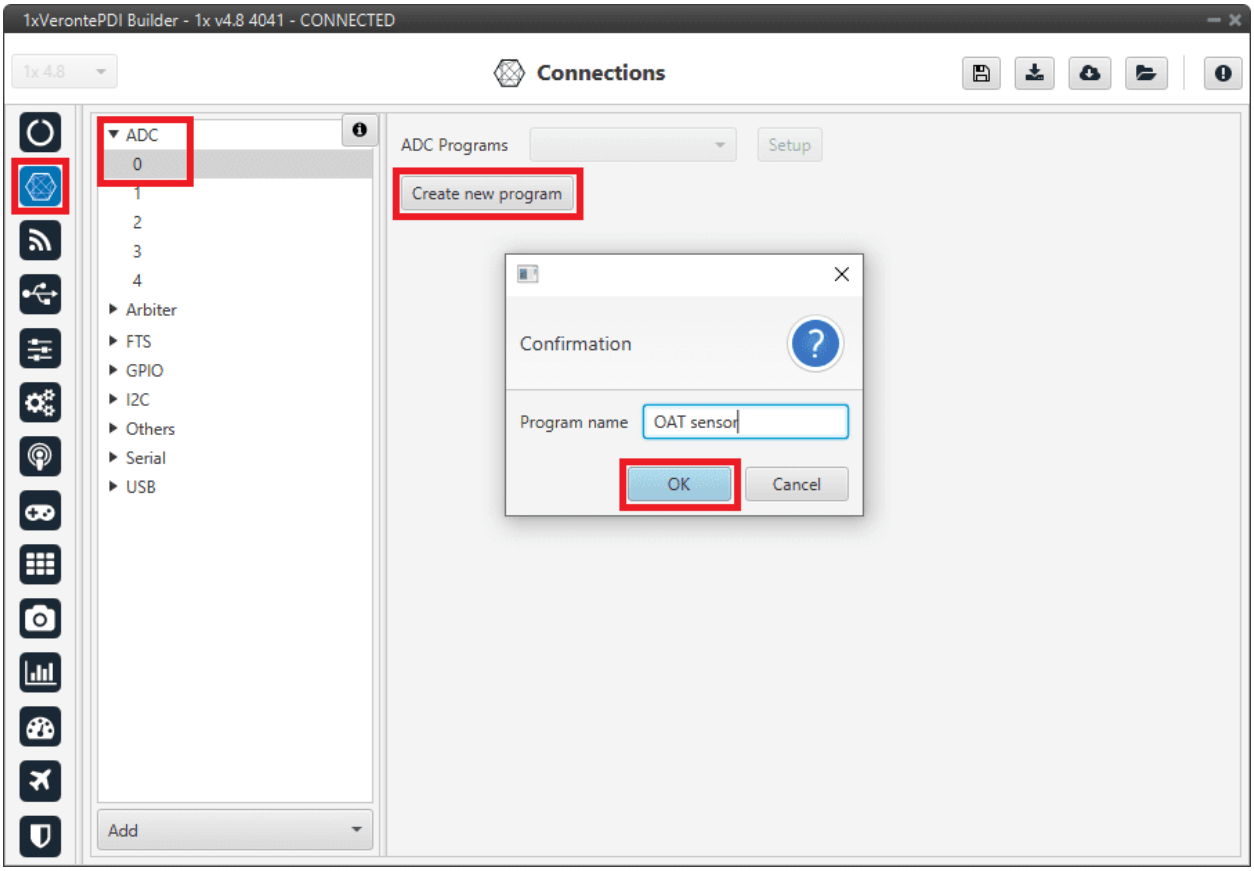


Fig. 77: **OAT Sensor - Create ADC program**

2. Go to **Block programs menu**.

   Configure the following operation (for more information about blocks, read *Block Programs* section of this manual):

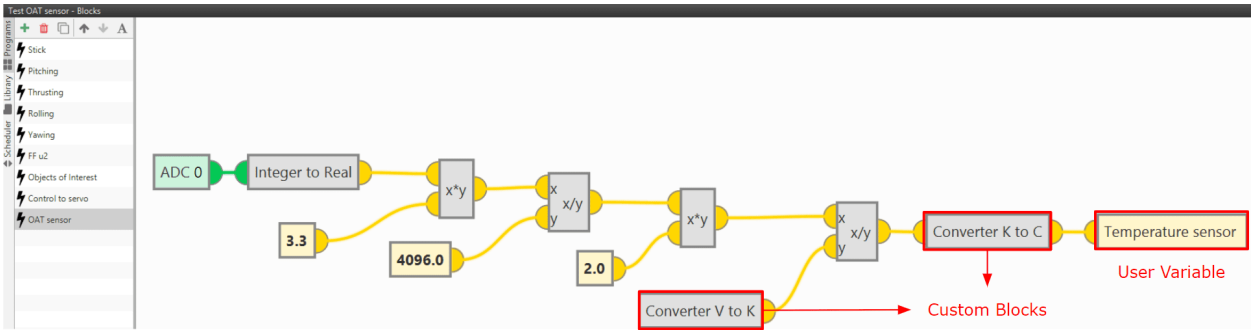


Fig. 78: **OAT sensor - Block Programs operation**

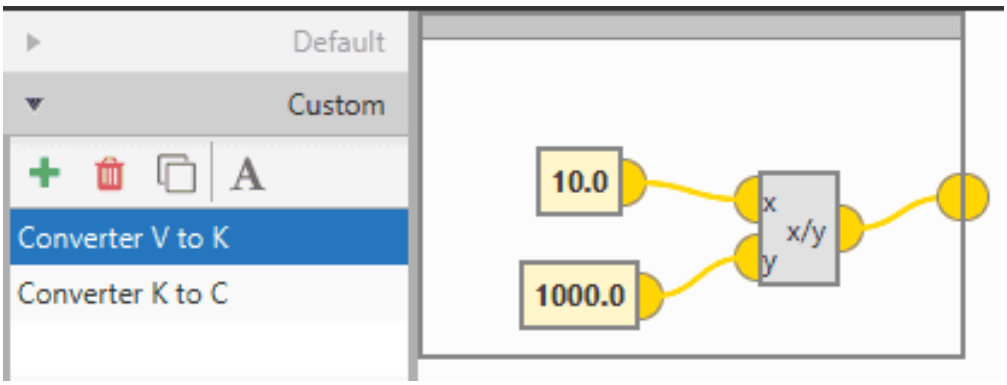


Fig. 79: **OAT sensor - Custom block: Converter from V to K**

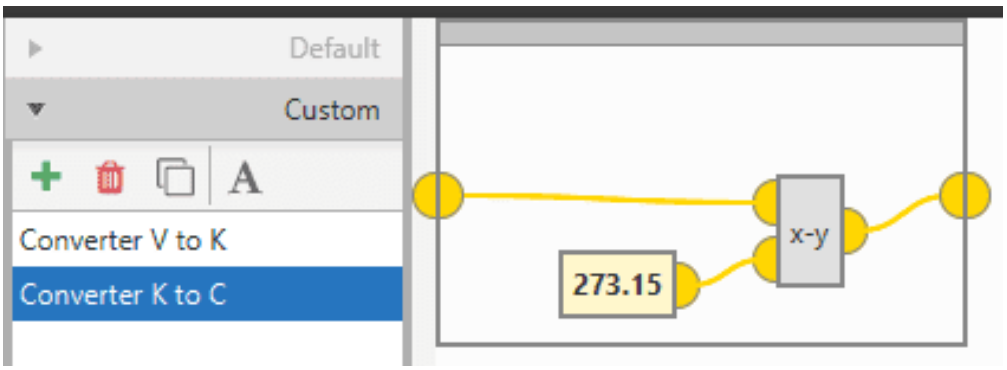


Fig. 80: **OAT sensor - Custom block: Converter from K to °C**

After implementing the operation, the variable *Temperature sensor* will represent the temperature (in °C) measured by the OAT sensor.

Note: If the temperature is needed in other units, the only thing necessary would be to modify the Custom block *Converter K to C*, or simply remove it.

Note: As Autopilot 4x can read up to 36 V per each ADC, the 3.3 value of the ADC program must be changed to 36 if applicable.

3.7.2.6 Vectornav VN-300

Vectornav VN-300 is an **external IMU** that can be connected via **RS-232** (serial interface) to Veronte Autopilot 1x.

The following steps explain how to configure Veronte Autopilot 1x to integrate this external IMU:

1. Go to Input/Output menu → Serial panel → **RS232 tab**.

   Configure the serial port parameters:

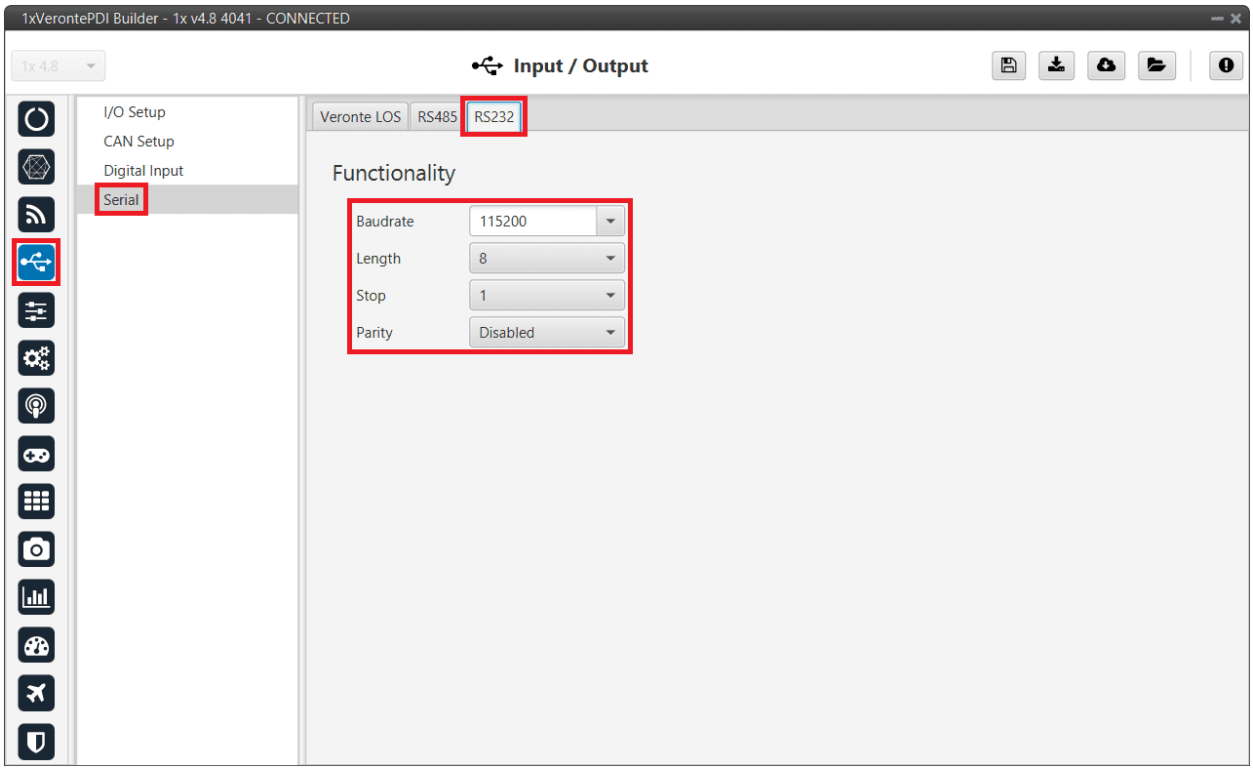


Fig. 81: **Vectornav VN-300 - RS232 port configuration**

2. Go to Input/Output menu → **I/O Setup panel**.

   Connect the **RS232 Producer** to the **Vectornav VN-300 Consumer**:

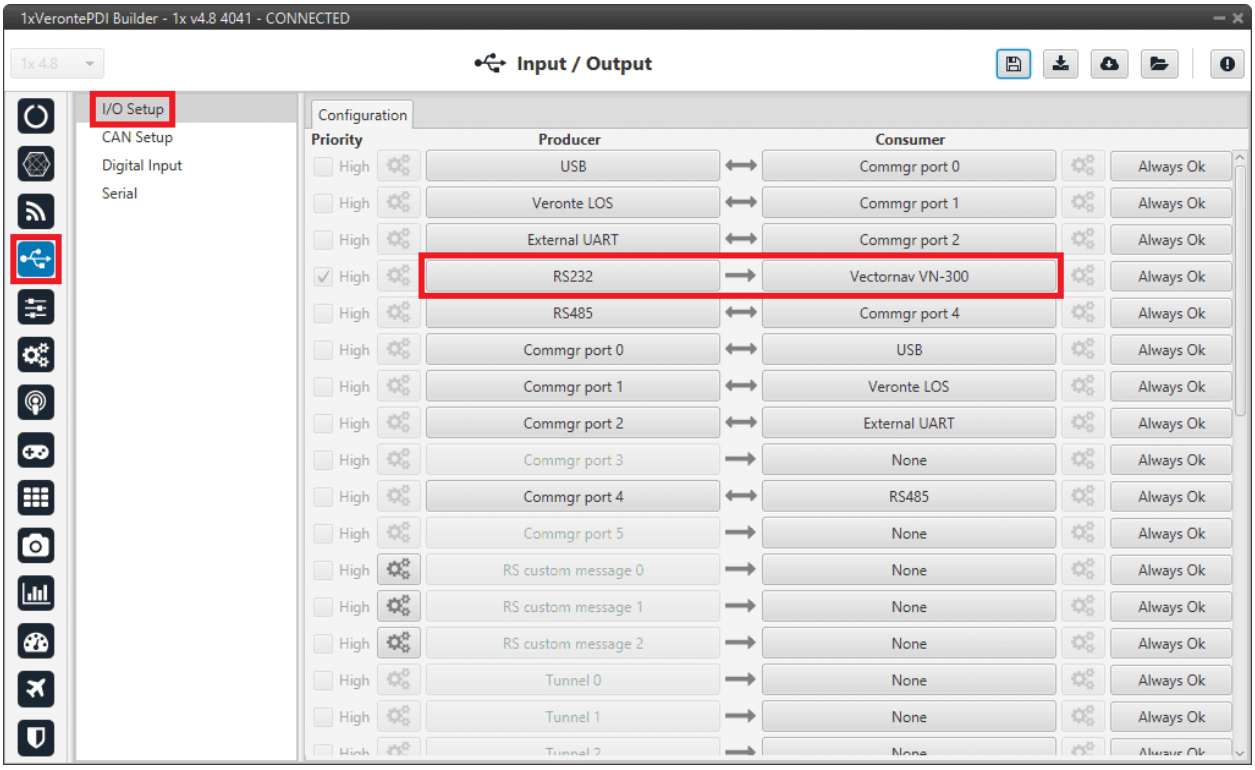


Fig. 82: **RS232 → Vectornav VN-300**

3. Go to **Block Programs menu**.
   - Create a program to configure the Vectornav VN-300 as the type of navigation.

     Usually the user has a **Navigation program** where the blocks that are related to the navegation are implemented.
   - Configure the *Navigation block* selecting **Vectornav VN-300**.

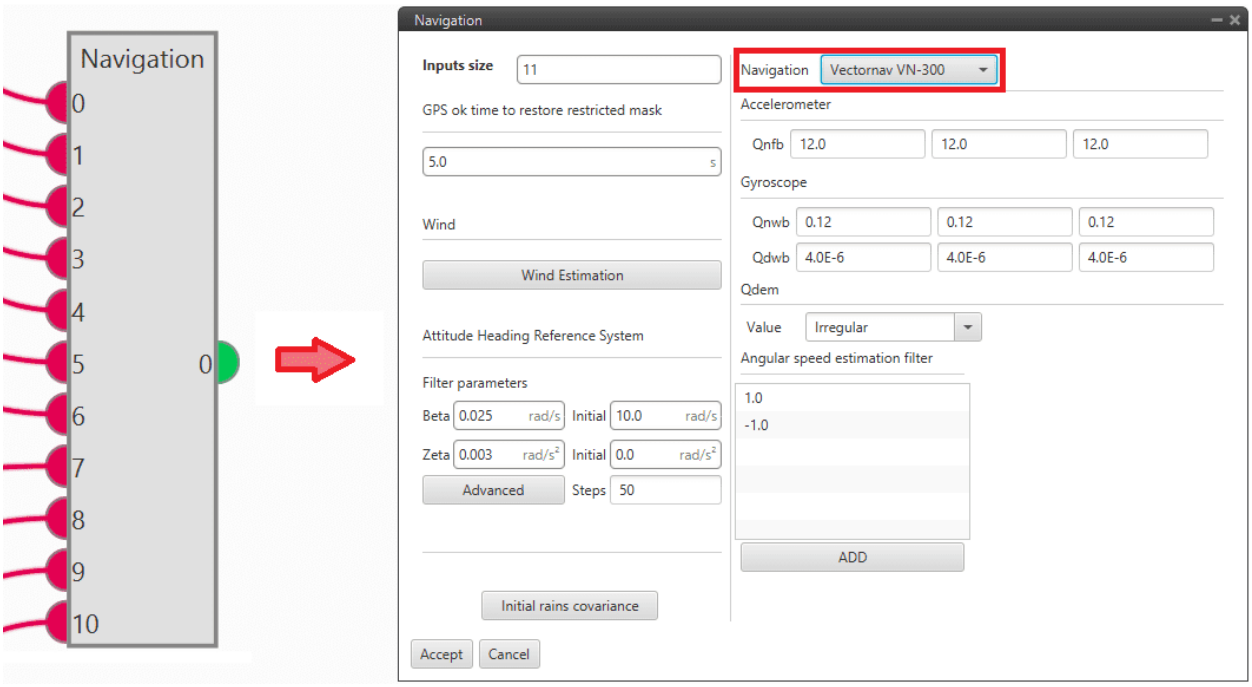


Fig. 83: **Vectornav VN-300 - Navigation sensor block configuration**

For more information on this block, see *Navigation blocks* of **Block Programs** section.

3.7.2.6.1 Vectornav VN-300 configuration

It is also required a configuration on the **Vectoran VN-300** IMU.

Follow the steps below to establish proper communication between **Vectornav VN-300** and **Veronte Autopilot 1x via RS-232**:

1. Connect the **Vectornav VN-300** IMU via USB to the PC (if necessary, use a USB-RS232 converter).
2. Connect one or both GPS antennas to it.
3. For its configuration, enable **periodic binary messages at 100 Hz**. Only the following outputs from the **Group 1 (Common group)** must be sent:

Outputs	Bit Offset
TimeGps	1
YawPitchRoll	3
AngularRate	5
Position	6
Velocity	7
Accel	8
InsStatus	12

Warning: Veronte Autopilot 1x is only capable to decode messages with **exactly that structure**, any **missing or added field** will cause Autopilot 1x to **reject the messages**.

In addition, the **baudrate** configured on the VN-300 must match that configured for Veronte Autopilot 1x. Therefore, it is recommended to configure the baudrate of the serial port to the standard **115200**.

3.7.3 Radios

Warning: The internal radio of Veronte Autopilots 1x depends on the hardware version, so the user should check the internal radio according to the hardware version of his Autopilot 1x:

- Veronte Autopilot 1x `v4.5`: To consult its internal radio, click here.
- Veronte Autopilot 1x `v4.8`: To consult its internal radio, click here.

3.7.3.1 Digi internal radio

3.7.3.1.1 Configuration

This section describes the necessary configuration for **1x PDI Builder** and the **Digi radio software** (XCTU) to allow a **correct communication between Veronte Autopilot 1x and its internal Digi radio**.

To configure the communication between Autopilots 1x and their internal Digi radios, apply the following steps to each one (air and BCS/PCS unit):

1. Connect the **Autopilot 1x** to a computer with **Veronte Link**, read its user manual to use it.

Configuration in 1x PDI Builder

2. Go to Input/Output menu → **I/O Setup panel**.

   The configuration of this panel is going to be **temporarily** modified in order to allow the setting up of a tunnel between the autopilot and the radio, i.e. the current configuration will have to be further re-established. For this reason, it is necessary that the user **first annotates the configuration** of **USB**, **Veronte LOS** and the ports to which they are connected. The following image shows an example.

   Note: It is recommended to take a screenshot for this step.

   Warning: Although the connection with Autopilot 1x will be lost via USB, users can still "see" the autopilot via serial (RS232 or RS485). For this purpose, the **bidirectional RS232 or RS485 connection must not be modified**.

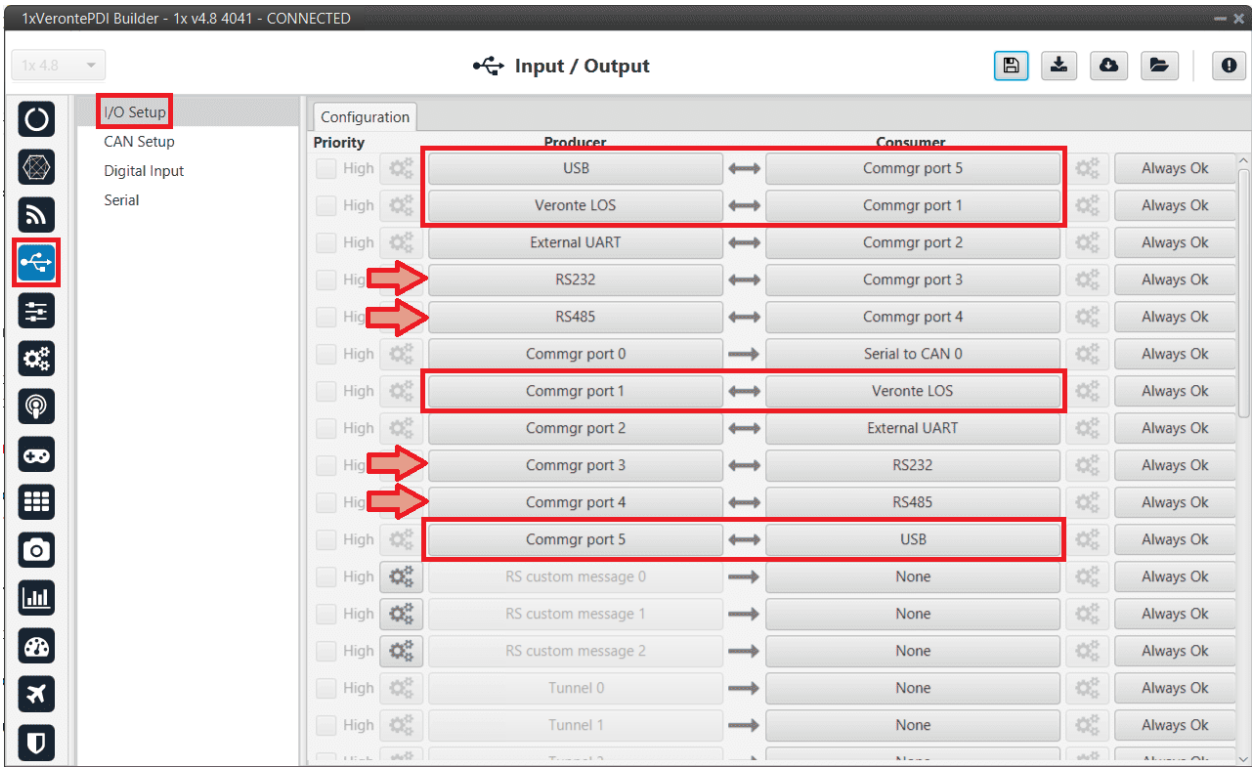


Fig. 84: **Digi internal radio - Example of configuration of USB and Veronte LOS ports**

3. Change the port which **USB producer** is connected to and select **Veronte LOS** as **consumer**.

   USB and **Veronte LOS** must have **bidirectional communication** $\longleftrightarrow$.

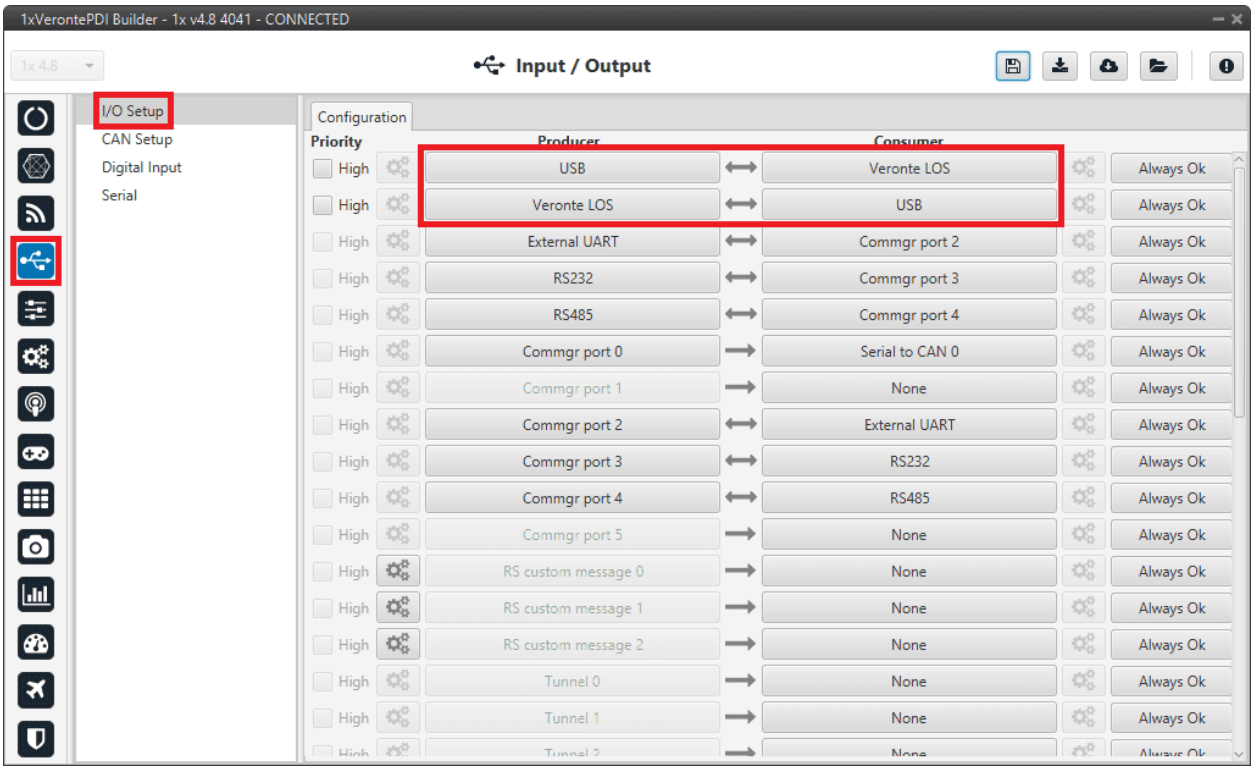


Fig. 85: **USB ⟷ Veronte LOS**

4. Go to Input/Output menu → Serial panel → **Veronte LOS tab**.

   It is important to know which **baudrate** is configured for the **Veronte LOS serial port** in order to **match** it with the one configured in the **Digi radio**.

   By default, the baudrate configured in **1x PDI Builder** is set to 115200.

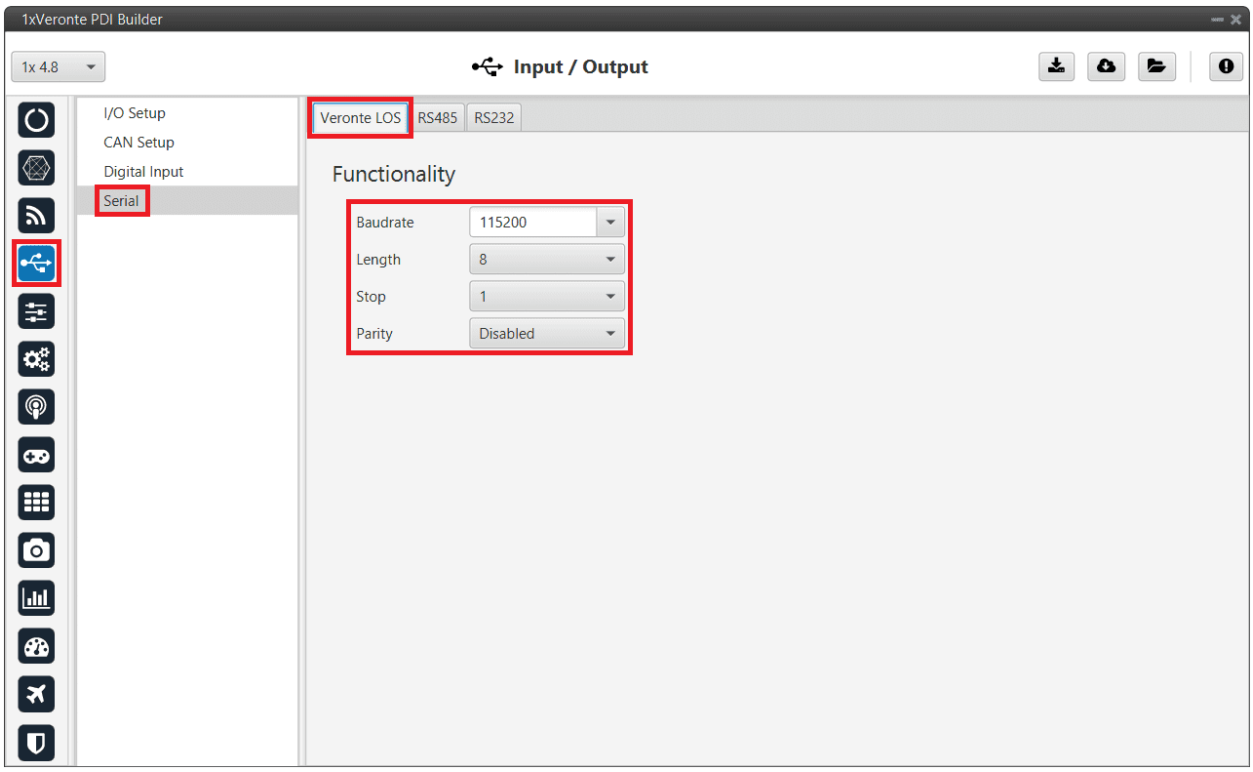


Fig. 86: **Digi internal radio - Veronte LOS baudrate**

5. Click on  to apply changes to the **Autopilot**.

Warning: The communication between computer and **Autopilot 1x** will be disconnected, since the autopilot is working as a tunnel between computer and radio. The computer will be communicating only with the Digi radio.

6. **Wait for the device to disconnect and close Veronte Link**. If the user does not close it, XCTU software will not be able to detect the radio as the COM is being managed by **Veronte Link**, and the following error message will appear:

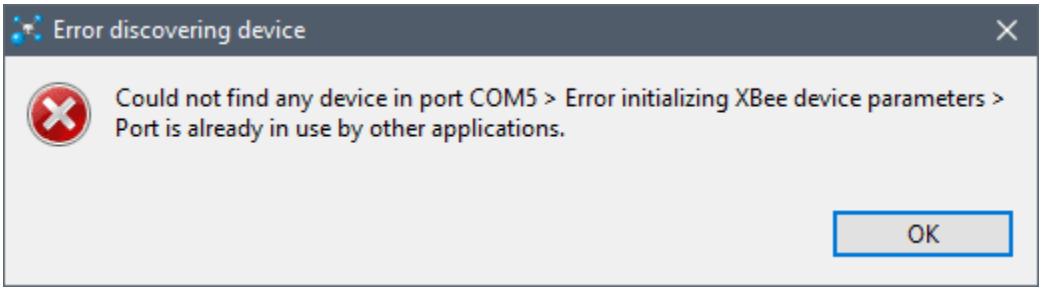


Fig. 87: **Digi internal radio - XCTU error message**

Important: Remember that to completely close the application the user must close it from the windows system tray.

Fig. 88: **Close Veronte Link**

Configuration in Digi radio software

7. Download and install XCTU (Digi radio software).
8. Build a configuration for 'air' or 'bcs' in **XCTU**:

   The integrated radio is the model **DIGI-XBEE3 XB3-24Z8UM**. For more information about how to configure it, read the XCTU User Guide.

   The following table shows which parameters can be configured. The rest of parameters should remain as default.

DIGI Parameter	Description
PL	Transmit power (100 mW)
ID	Network addres PAN ID
DD	Device type identifier
BD	UART baud rate (115200)
RR	Retries (minimum 5)
CH	2.4 GHz channel to send
MM	Mac mode, 802.15.4 with Digi header for discovery and packages duplicate
CA	Clear channel threshold as dBm
EA	Ack failures
EC	Failure to sent due to excess of energy in channel

Note: Radios to pair must have matching PAN IDs.

Warning: Check that the baudrate of the radio matches the baudrate configured in **1x PDI Builder**. If it is not the same, change one of them to match. Remember that **Veronte LOS baudrate must not exceed 115200**, as this may compromise proper communication.

9. **Only for Autopilots 1x implemented in a 4x**

   Digi radios are able to create a network and talk to each other, even if they are configured as endpoint.

   This is a problem, as it leads to radio channel overload. To prevent this problem, the destination addresses must be configured so that the ground station transmits in broadcast and each air unit transmits only to the ground.

   The following table shows how to configure air and ground units in **XCTU**:

Radio	Parameter	Configuration
Ground	**MY** 16-bit Source Address	`FFFF`
	DH Destination Address High	`0`
	DL Destination Address Low	`FFFF`
Air	**MY** 16-bit Source Address	`FFFF`
	DH Destination Address High	**SH** of the ground radio
	DL Destination Address Low	**SL** of the ground radio

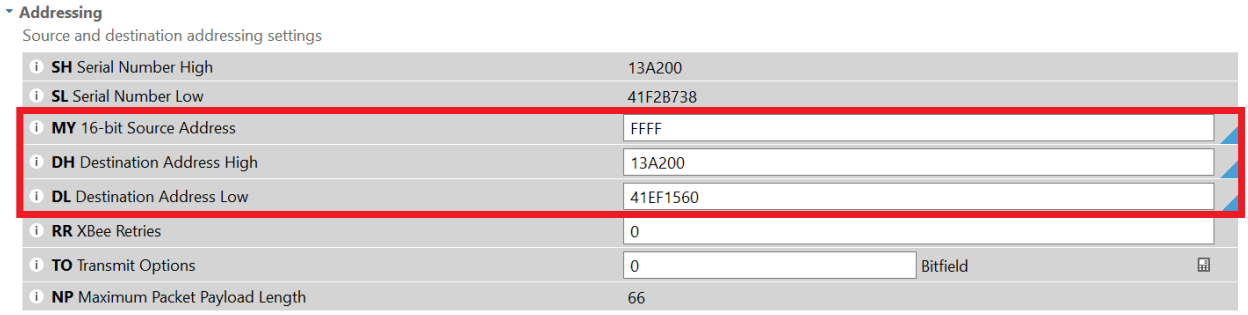


Fig. 89: **Digi internal radio - Air radio configuration example**

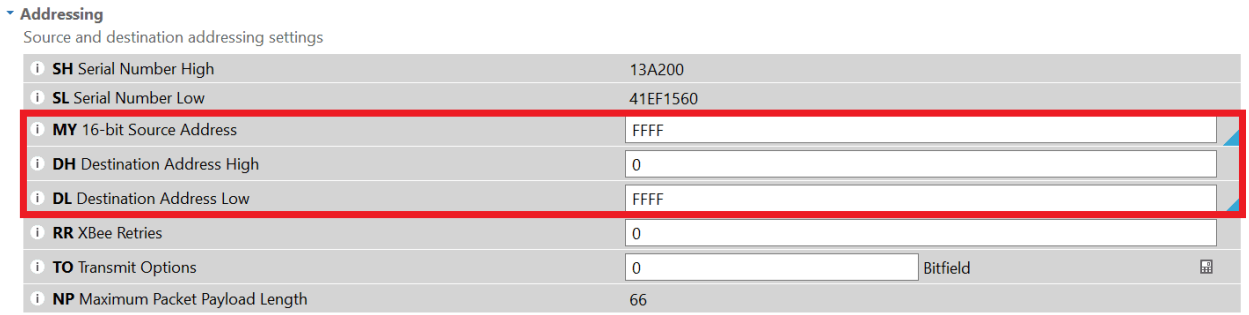


Fig. 90: **Digi internal radio - Ground radio configuration example**

10. After configuring the radio, the communication between computer and **Autopilot 1x** should be restored. To do it, force the maintenance mode.

Configuration in 1x PDI Builder

11. Go to Input/Output menu → **I/O Setup panel**.

    Finally, after configuring the Digi radio in its software, restore the annotated **USB** and **Veronte LOS** configuration (*step 2*).

If **communication between the Digi radio and the Veronte Autopilot 1x is lost** after the entire radio setup process described above, or later during operation, please refer to the *Communication lost with internal Digi radio - Troubleshooting* section of this manual.

3.7.3.1.2 Operational range

The following table is a **reference** of the functional range for each telemetry load (**it may be affected by enviromental conditions**):

	Frequency		
Load	**5 Hz**	**10 Hz**	**20 Hz**
Low (Half telemetry vector)	> 700 m	500 m	300 m
Medium (one telemetry vector)	> 700 m	100 m	80 m
High (two or more telemetry vectors)	300 m	80 m	X

Note: Telemetry vectors are structured messages with up to 255 bytes of data. To know more about them, read Message structure section of **VCP** user manual.

3.7.3.2 Microhard internal radio

This section describes the necessary configuration that must be performed in **1x PDI Builder** and 1x PDI Calibration to allow a **correct communication between Veronte Autopilot 1x and its internal Microhard radio**.

To configure the communication between Autopilots 1x and their internal Microhard radios, apply the following steps to each one (air and bcs unit):

1. Connect the **Autopilot 1x** to a computer with **Veronte Link**, read its user manual to use it.

Configuration in 1x PDI Builder

2. Go to Input/Output menu → **I/O Setup panel**.

   Since the configuration of this panel is going to be modified **temporarily**, i.e. the current configuration will have to be re-established, just to be able to set up a tunnel between the autopilot and the radio.

   It is necessary that the user **first annotates the configuration** of **Veronte LOS**, **Tunnel** and the ports to which they are connected. The following image shows an example.

   Note: It is recommended to take a screenshot for this step.

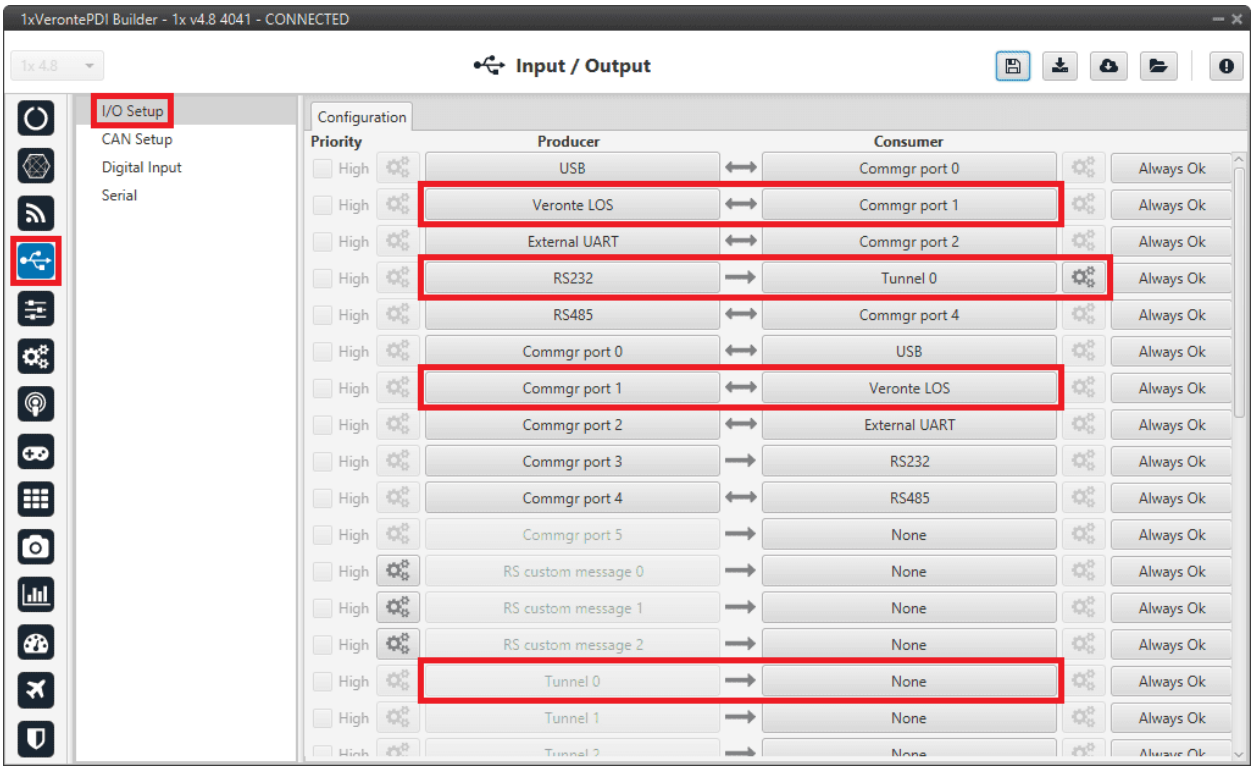


Fig. 91: **Microhard internal radio - Example of configuration of Veronte LOS port**

3. Change the port which **Veronte LOS producer** is connected to and select a **Tunnel** as **consumer**, in this example *Tunnel 0* has been selected.

   Veronte LOS and **Tunnel 0** must have **bidirectional communication** $\longleftrightarrow$.

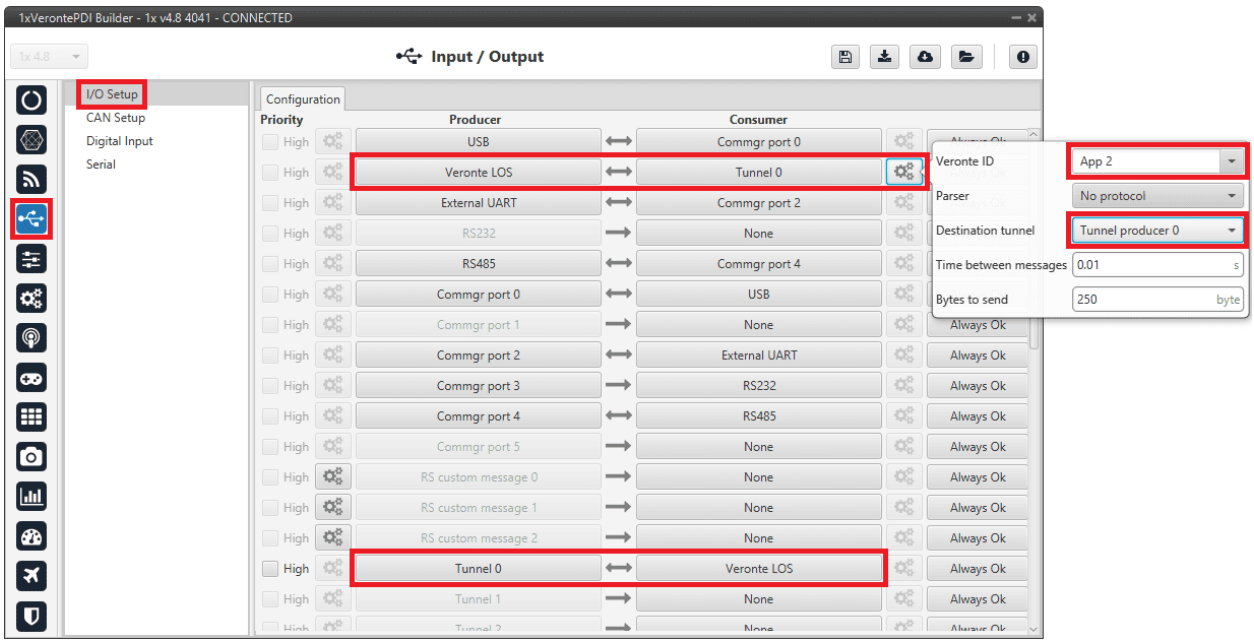


Fig. 92: **Veronte LOS $\longleftrightarrow$ Tunnel 0**

4. Click on  to apply the changes to the **Autopilot 1x**.

Configuration in 1x PDI Calibration

5. In **1x PDI Calibration**, go to **Terminal panel**.

   To set up the Microhard radio, the **1x PDI Calibration** software provides a **Microhard wizard** to assist the user in setting up the radio.

   Please refer to the Terminal and Microhard setup helper sections of the **1x PDI Calibration** user manual.

Configuration in 1x PDI Builder

6. Go to Input/Output menu → **I/O Setup panel**.

   Finally, after configuring the Microhard radio in **1x PDI Calibration**, restore the annotated **Veronte LOS** and **Tunnel 0** configuration (*step 2*).

7. Click on  to apply the changes to the **Autopilot 1x**.

3.7.3.3 External radios

This section describes the required configuration to be performed in **1x PDI Builder** to allow a **correct connection between Veronte Autopilot 1x and any external radio**.

External radios compatible with the Autopilot 1x, such as *Microhard*, *DTC*, *Digi*, *Silvus* and *Veronte Data Link* (Embention external radio, contact sales@embention.com for more information).

After configuring the external radio in the corresponding software, follow the steps below:

1. Go to Input/Output menu → Serial panel → **RS232 tab**.

   Check that these parameters are the same as the parameter values previously set in the external radio.

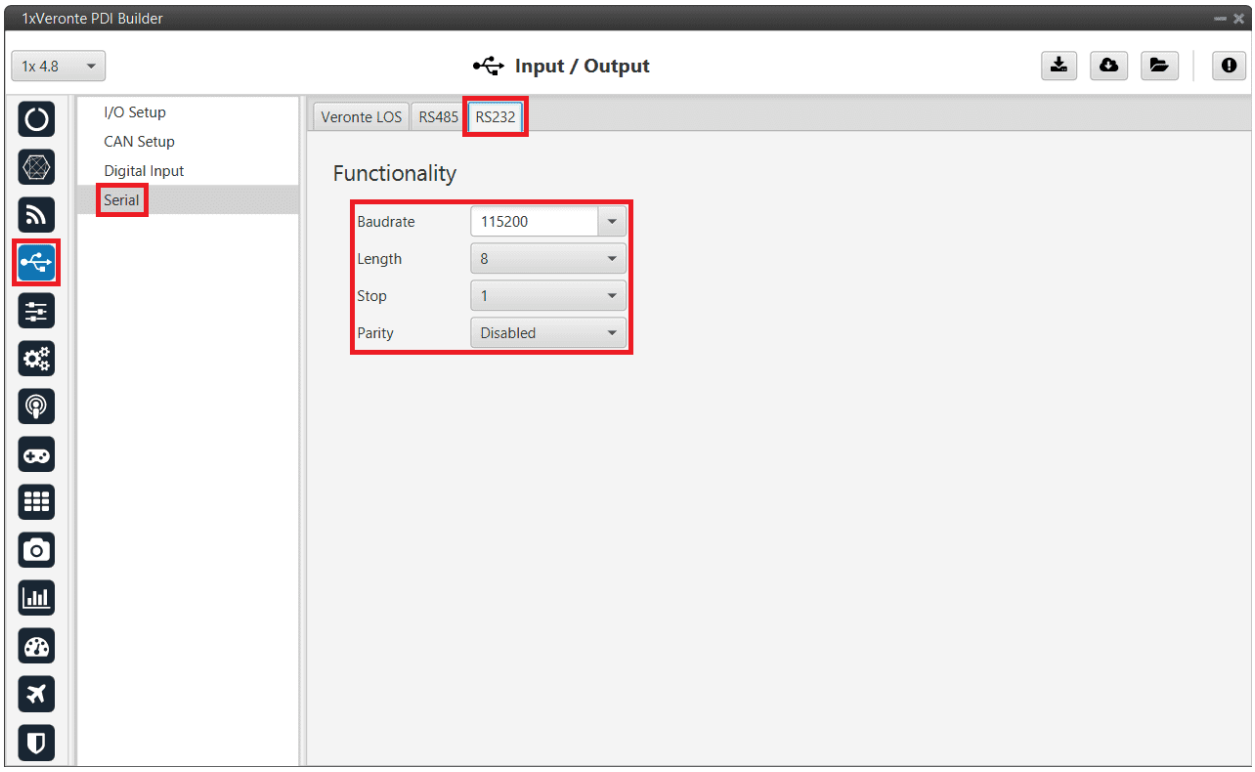


Fig. 93: **External radios - RS232 port configuration**

2. Go to Input/Output menu → **I/O Setup panel**.

   RS-232 has to be configured as a **bidirectional** commgr port.

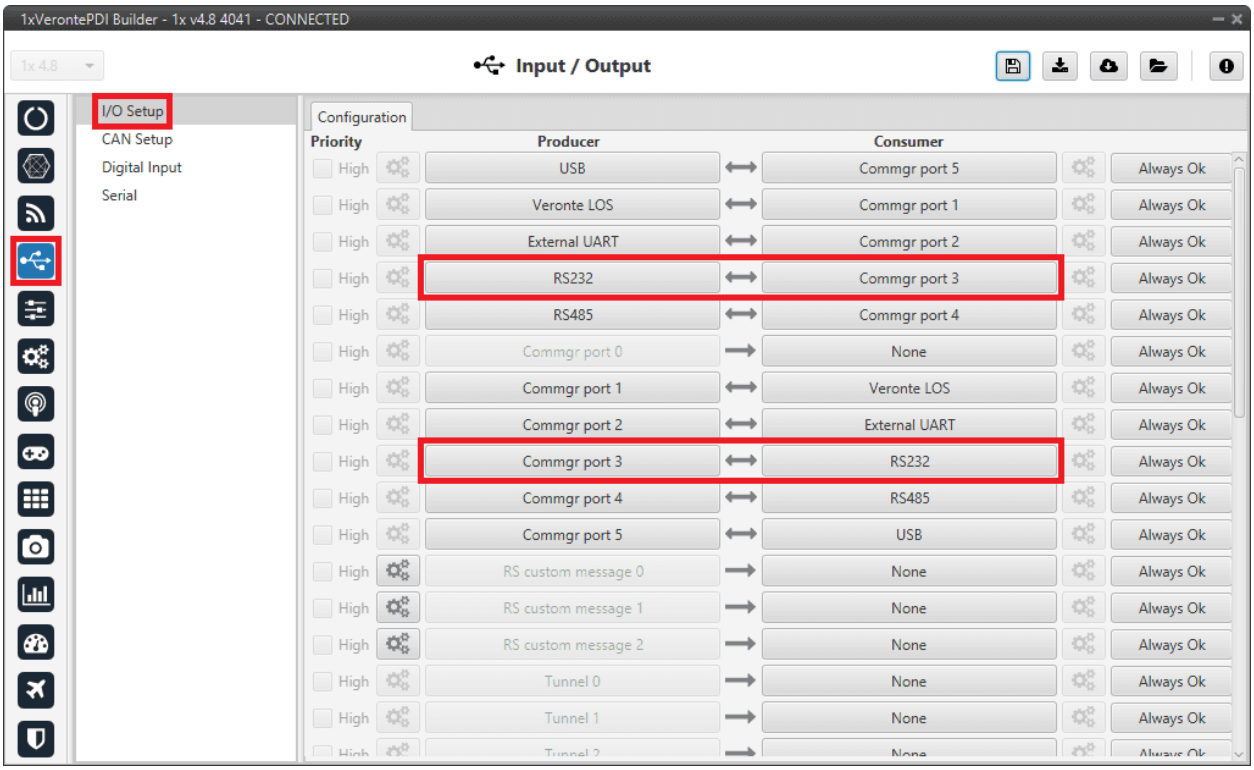


Fig. 94: **External radios - I/O Setup configuration**

Note: These settings have to be made in both Autopilot 1x units (GND and AIR).

3.7.4 Servos

The user can configure any actuator compatible with the communication interfaces.

3.7.4.1 PWM

The following steps explain how to configure a PWM servo in Veronte Autopilot 1x.

1. Connect the servo according to the manufacturer's documentation and follow the Pinout - Hardware Installation section of the **1x Hardware Manual** to connect it to the Autopilot 1x.
2. Go to Connections menu → **PWM panel**.
    - Select and configure the PWM pins where the servos are connected. Set the frequency according to the manufacturer's specifications.

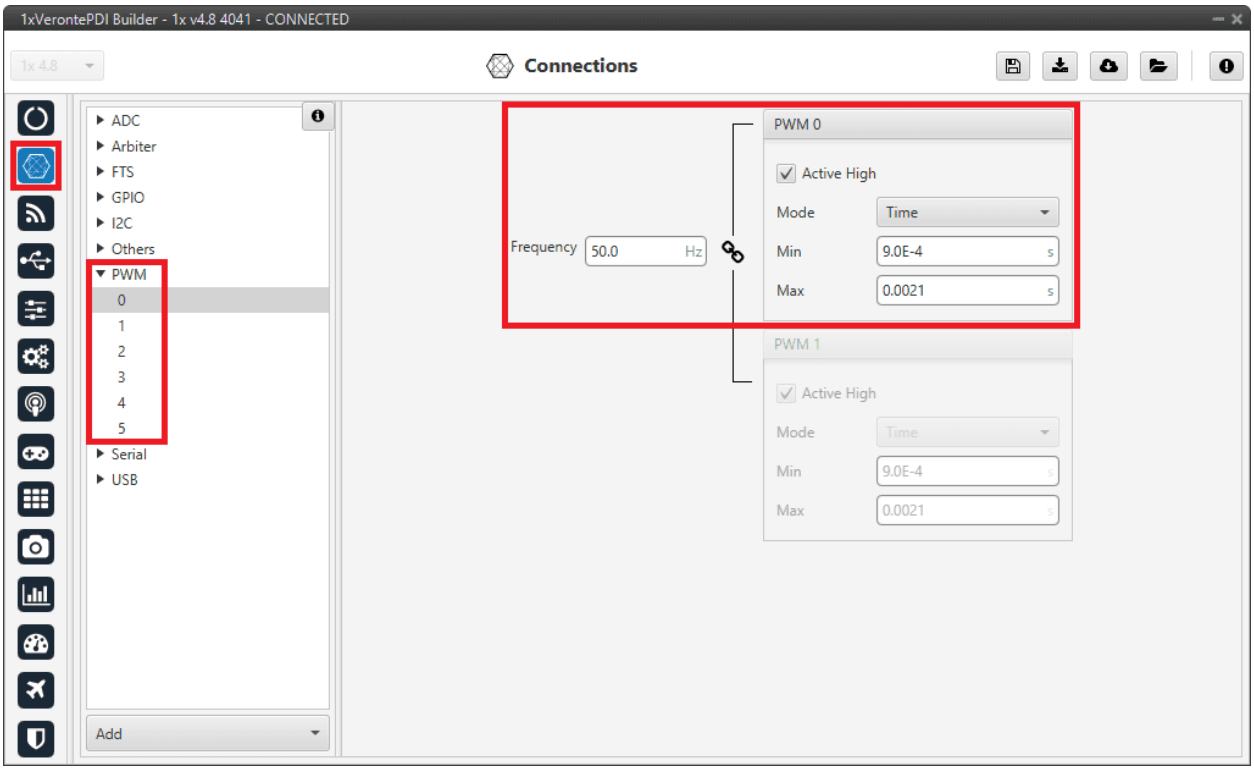


Fig. 95: **PWM - Connections configuration**

> **Caution:** If there is no PWM tab or the PWM pin where the servo is connected is not shown on the interface, it must be because it is configured as GPIO. For more information on this, refer to *PWM - Connections* section of this manual.

3. Go to **Block Programs menu**.
   - Create a program to make the necessary connection to the servo blocks.

     Usually the user has a **Control to servo program** where the servo blocks are implemented.
   - Configure the *Actuator block* connecting **PWM** block as *Pulse* and **Actuator Outputs** as *Servo*:

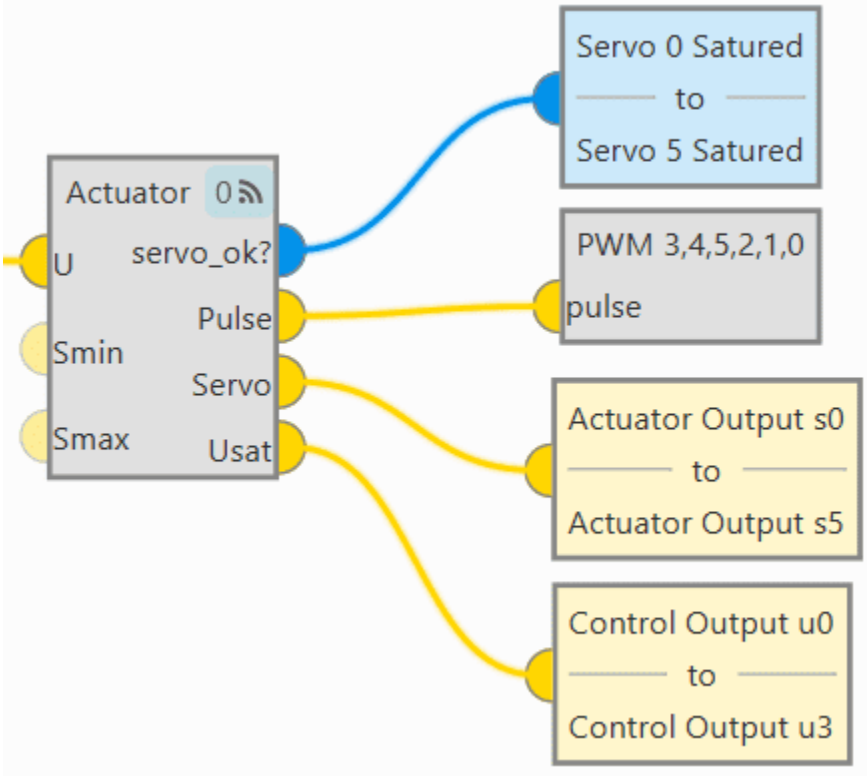


Fig. 96: **PWM - Block Programs connection**

- Assign a given PWM to a given actuator output.

  The assignment is done automatically in the order in which they are configured in the blocks. That is, the first PWM (the one with Id 0) will relate to the first actuator output, which does not necessarily mean that *PWM 0* is assigned to *Actuator Output s0*.

  In this example, the PWMs are assigned to the actuator outputs as shown in the following figure:

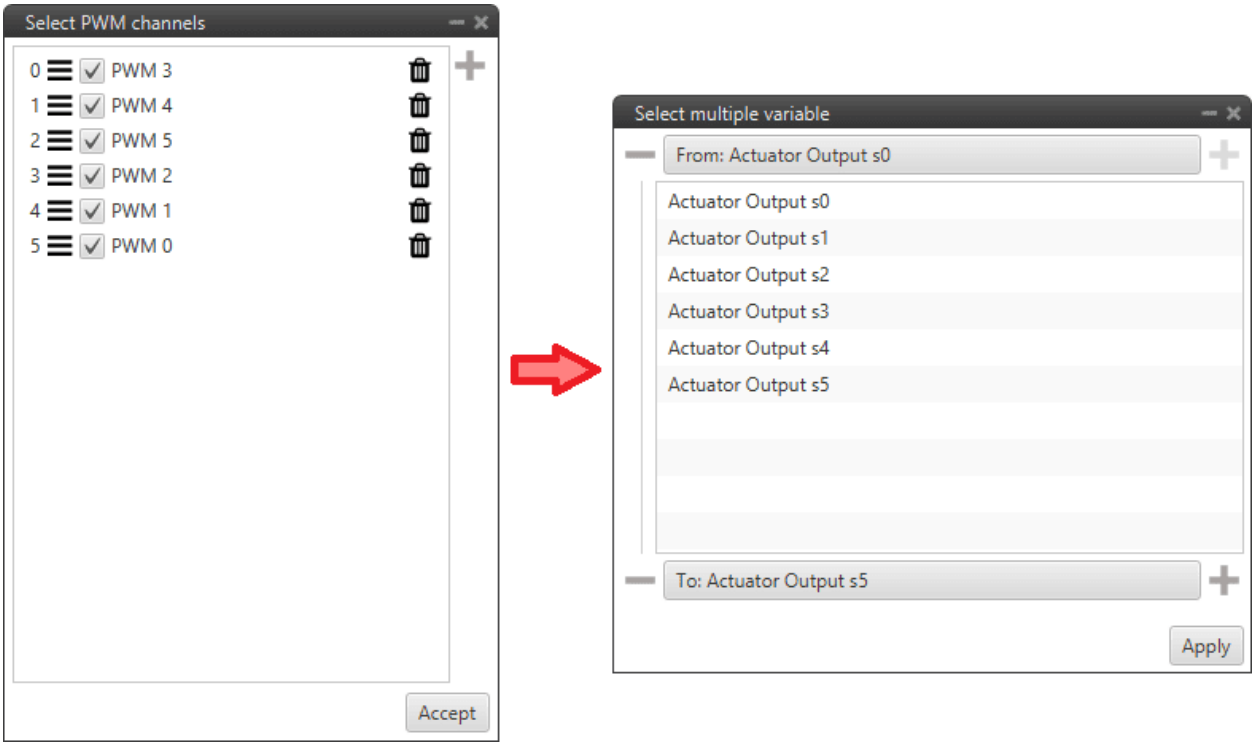


Fig. 97: **PWM - Block Programs configuration**

> **Note:** For instance, *PWM 5* (with id 2) is assigned to *Actuator Output s2*.

For more information on *Actuator* and *PWM* blocks, see *Actuator - Servos blocks* and *PWM - Servos blocks* of **Block Programs** section.

3.7.4.2 Serial

Serial servos are configured differently than PWM servos as the protocol of a serial device must be defined with *serial custom messages*.

In this case a **PWM** variable must be sent through a serial interface.

3.7.4.2.1 Volz DA26 - RS485

Firstly, the following wiring connection is recommended for a **RS485 connection between Volz DA26 servos and Veronte Autopilot 1x**:

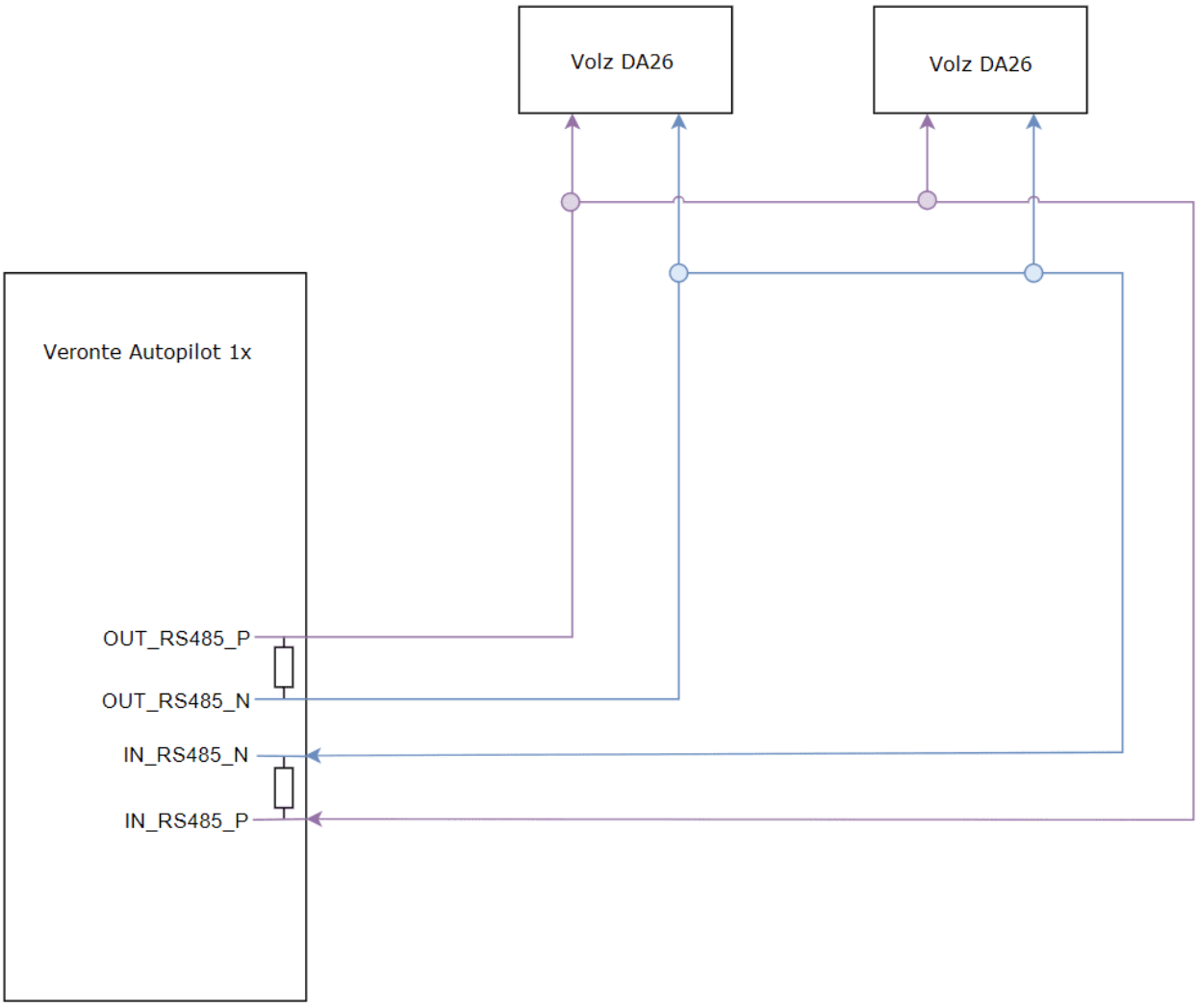


Fig. 98: **Volz DA26 - Veronte Autopilot 1x wiring connection**

The above diagram is made for the case where 2 Volz DA26 servos are connected, however, the connection is the same in case the user wants to connect only one or as many servos as the bus allows.

Follow the steps below to configure a **Volz DA26 servo via RS-485**.

1. Go to Input/Output menu → **I/O Setup panel**.

   Bidirectionally connect the **RS485** port to a **RS custom message**, in this example *RS custom message 0* is used:

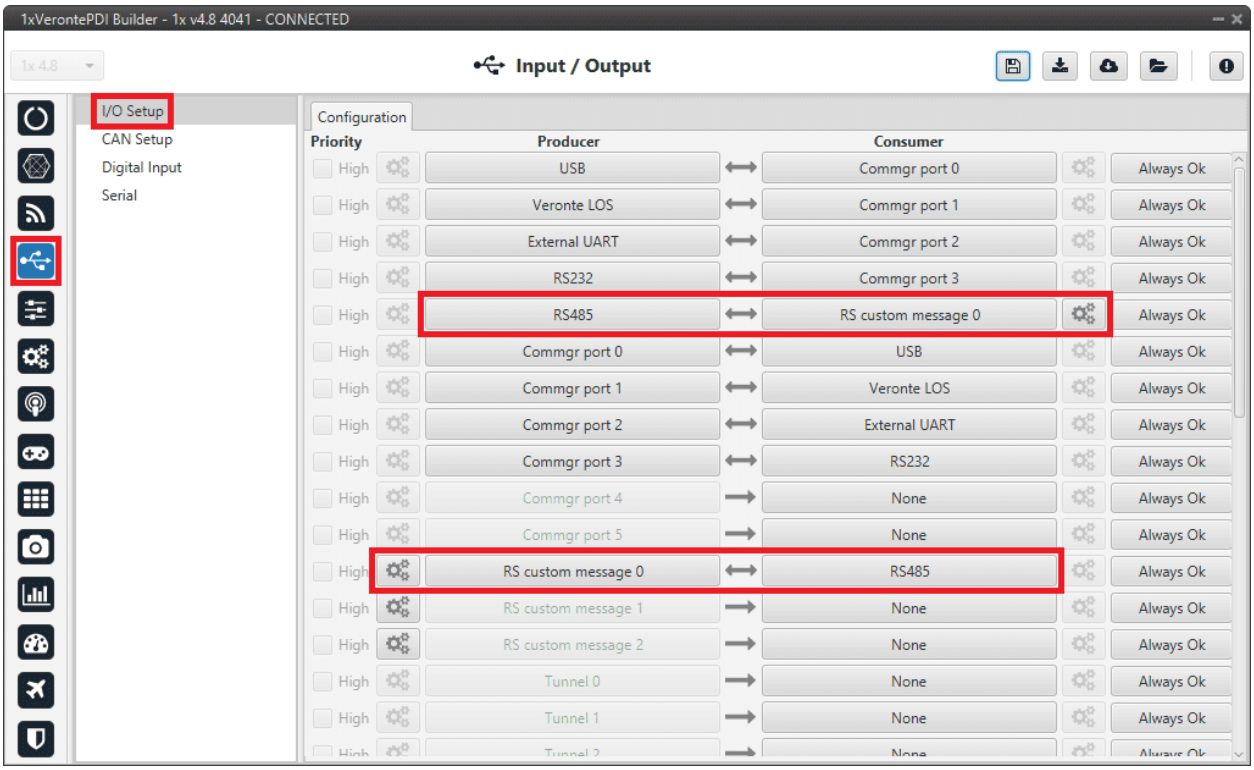


Fig. 99: **RS485 ↔ RS custom message 0**

2. Click on  to configure the *RS custom message 0* producer. Users must setup it by defining the protocol specified by the manufacturer:

Note: As the RS-485 is a **Half Full duplex** serial port, Veronte Autopilot 1x needs to leave this serial bus free for a certain time in order to receive the servo response. This is done by setting the **Delay** parameter.

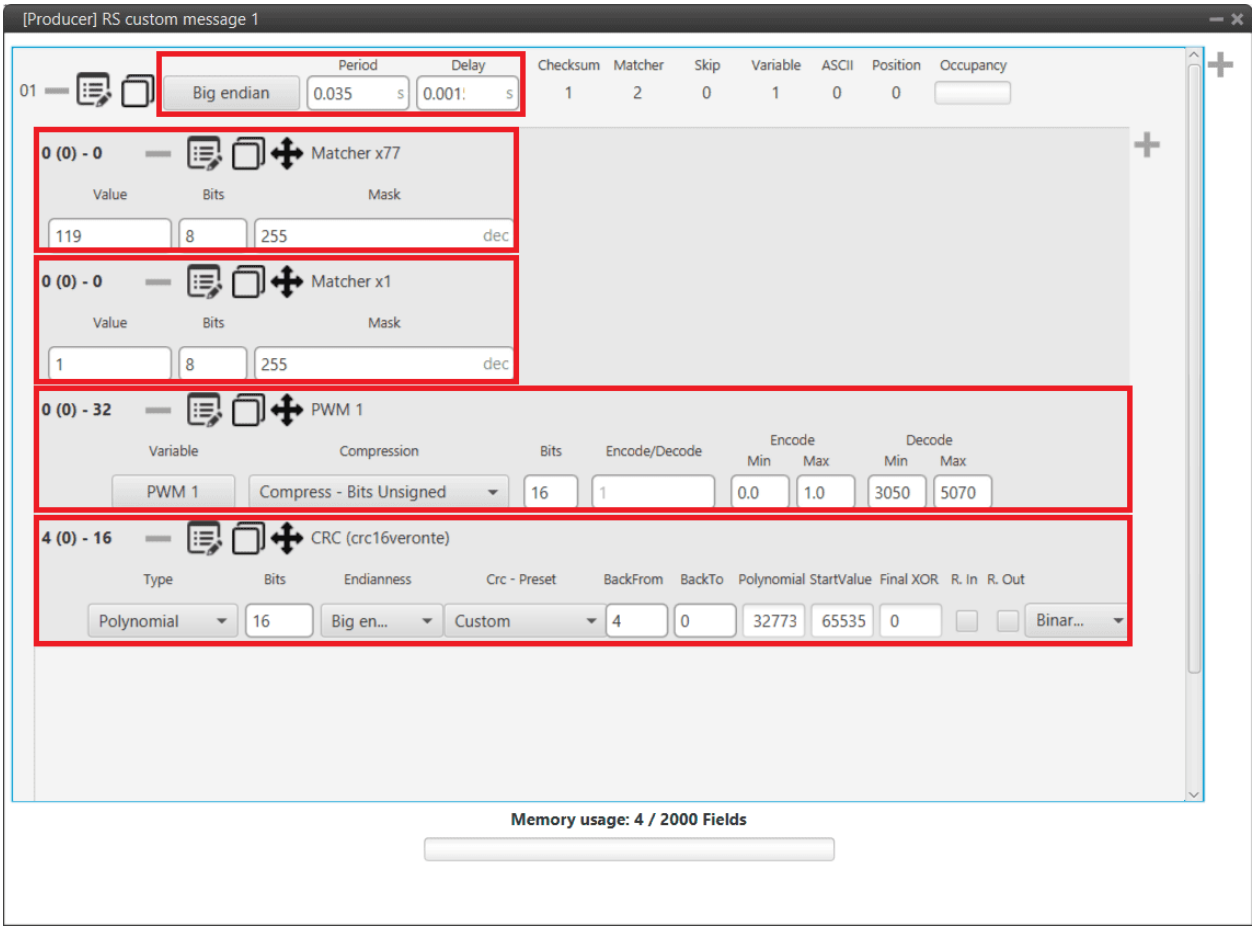


Fig. 100: **Volz DA26 - RS custom message 0 configuration**

- **Endianness**: Big endian
- **Period**: 0.035
- **Delay**: 0.0015
  - **Matcher x77**: Silent mode command (0x77).
    - **Value**: 119
    - **Bits**: 8
    - **Mask**: 255
  - **Matcher x1**: Servo interface Id = 1. The Id will be different for each servo and/or interface.
    - **Value**: 1
    - **Bits**: 8
    - **Mask**: 255
  - **PWM 1**: *PWM* is the variable that carries the information that has to be applied to the servo. Therefore, it must be included in the message.
    - **Variable**: PWM 1
    - **Compression**: Compress - Bits Unsigned

    - **Encode**: 0 / 1
    - **Decode**: 3050 / 5070
  - **CRC (Custom)**: A *Checksum* is needed to complete the communication protocol.
    - **Type**: Polynomial
    - **Bits**: 16
    - **Endianness**: Big endian
    - **CRC - Preset**: Custom
    - **BackFrom**: 4
    - **BackTo**: 0
    - **Polynomial**: 32773
    - **Start Value**: 65535
    - **Final XOR**: 0

Note: For more information on checksum, see *Checksum (CRC) explanation - Input/Output* section of this manual.

3.7.5 Stick

3.7.5.1 PPM Stick

3.7.5.1.1 General case

This is the case where the 1x ground unit (or BCS/PCS) sends commands directly to the 1x air unit.

Follow the steps below to perform a correct stick configuration on both units.

3.7.5.1.1.1 Ground unit

1. Go to Input/Output menu → **Digital Input panel**.

   Make sure that the following parameters have been configured:

   - Producer: **CAP 0**
     - Enabled
     - Select the pin to which the transmitter is connected (normally *EQEP_A*)
     - Edge detection: First rising edge
   - Consumer: **PPM 0**

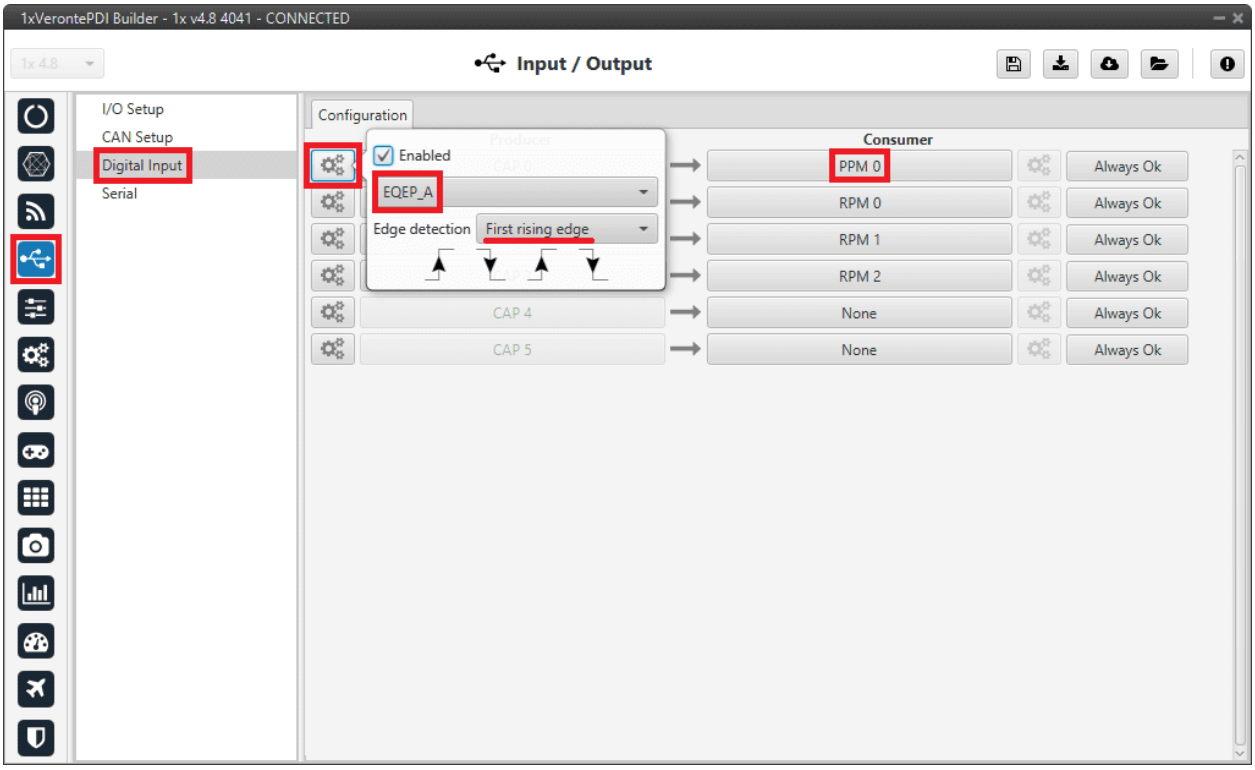


Fig. 101: **General case (Ground unit) - Digital Input configuration**

2. Go to Connections menu → **GPIO panel**.

   Verify that the pin to which the transmitter is connected, in this case *GPIO 16 (i.e., EQEP A)*, is set as input.

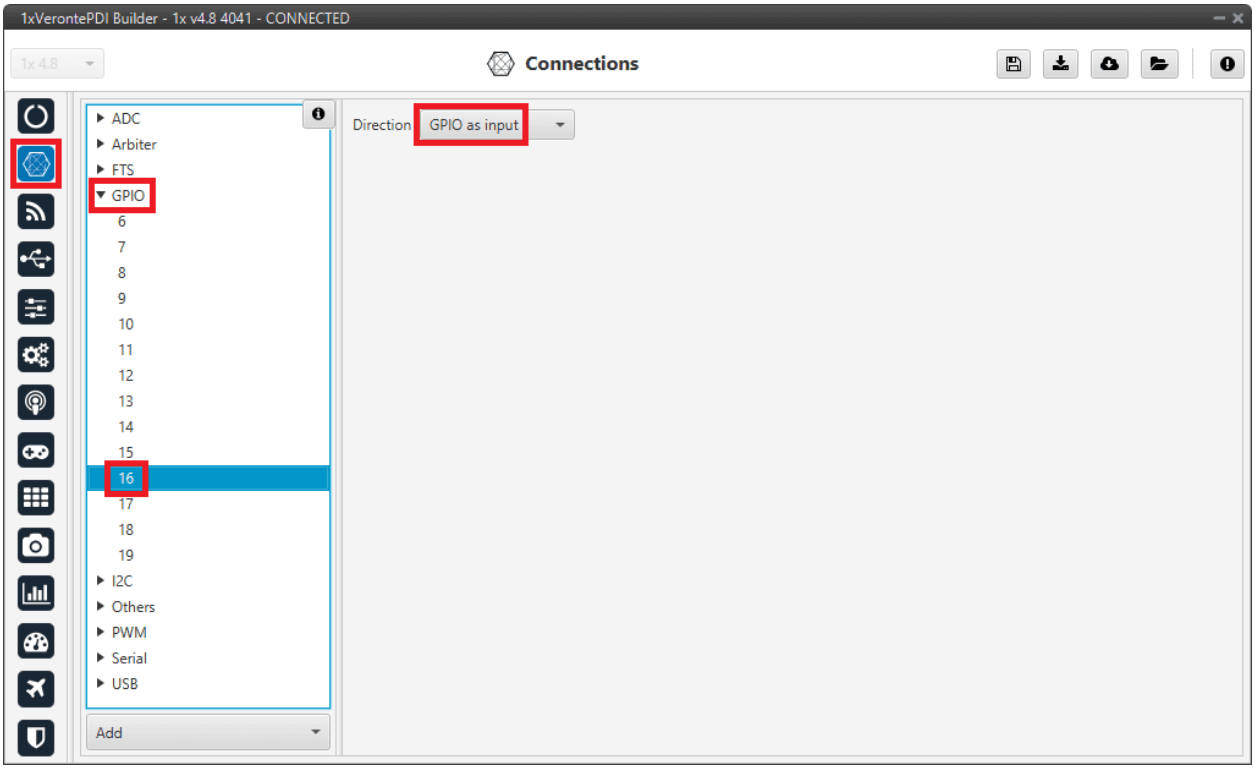


Fig. 102: **General case (Ground unit) - GPIO configuration**

3. Go to Stick menu → Transmitter 0 panel → **PPM tab**.

   Select the brand of transmitter that applies.

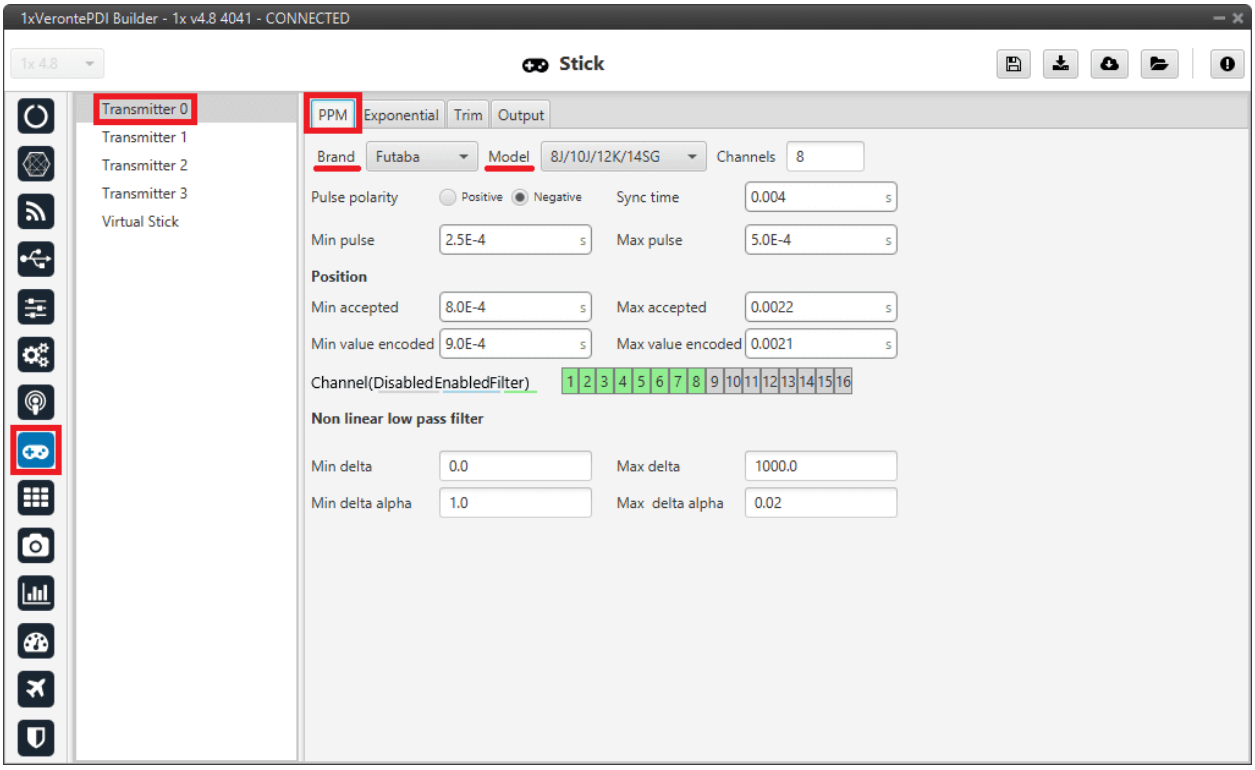


Fig. 103: **General case (Ground unit) - PPM configuration**

4. Go to Stick menu → Transmitter 0 panel → **Output tab**.

   Click on **Enable** and on **Remote** to send the stick information to the air unit. Please check the recommended values for the configurable parameters described in the *Ouput - Stick* section of this manual.

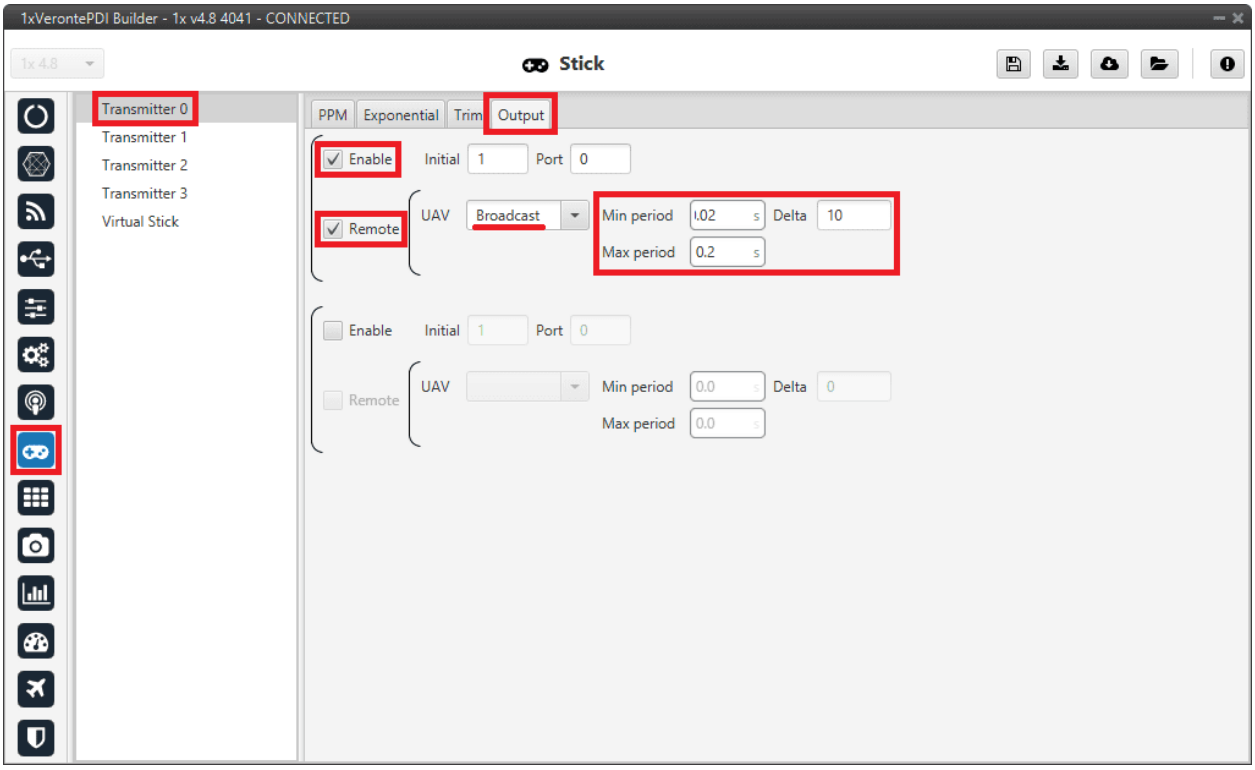


Fig. 104: **General case (Ground unit) - Output configuration**

If all these settings are correct, users can check that **'Stick PPM 0 not detected'** variable of the **GND unit** will be true.

Fig. 105: **Stick PPM 0 not detected variable - True**

3.7.5.1.1.2 Air unit

1. Go to Stick menu → Transmitter 0 panel → **PPM tab**.

   Select the brand of transmitter that applies (make the same configuration as the ground unit).

2. Go to Stick menu → Transmitter 0 panel → **Output tab**.

   Just click on **Enable**.

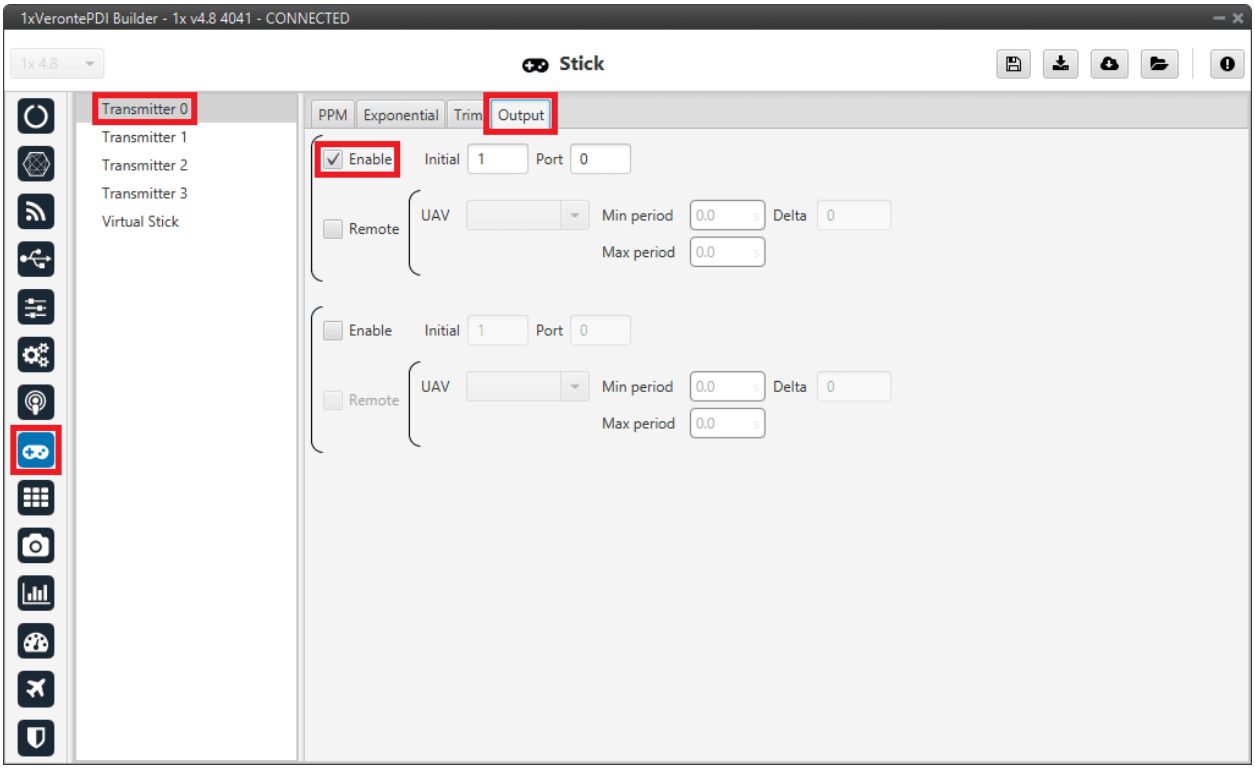


Fig. 106: **General case (Air unit) - Output configuration**

3. Go to Block Programs menu → **Stick program** → Double click on the **Stick block** → **Edit sources**.

Note: Normally the user has a **Stick Program** where the blocks that are related to the stick are implemented, however, the name of the user's program may be different.

Input the **ground unit address** to receive the stick information from that source and put it as the **highest priority** in the priority table. We recommend a Time Out of **0.4 s**.

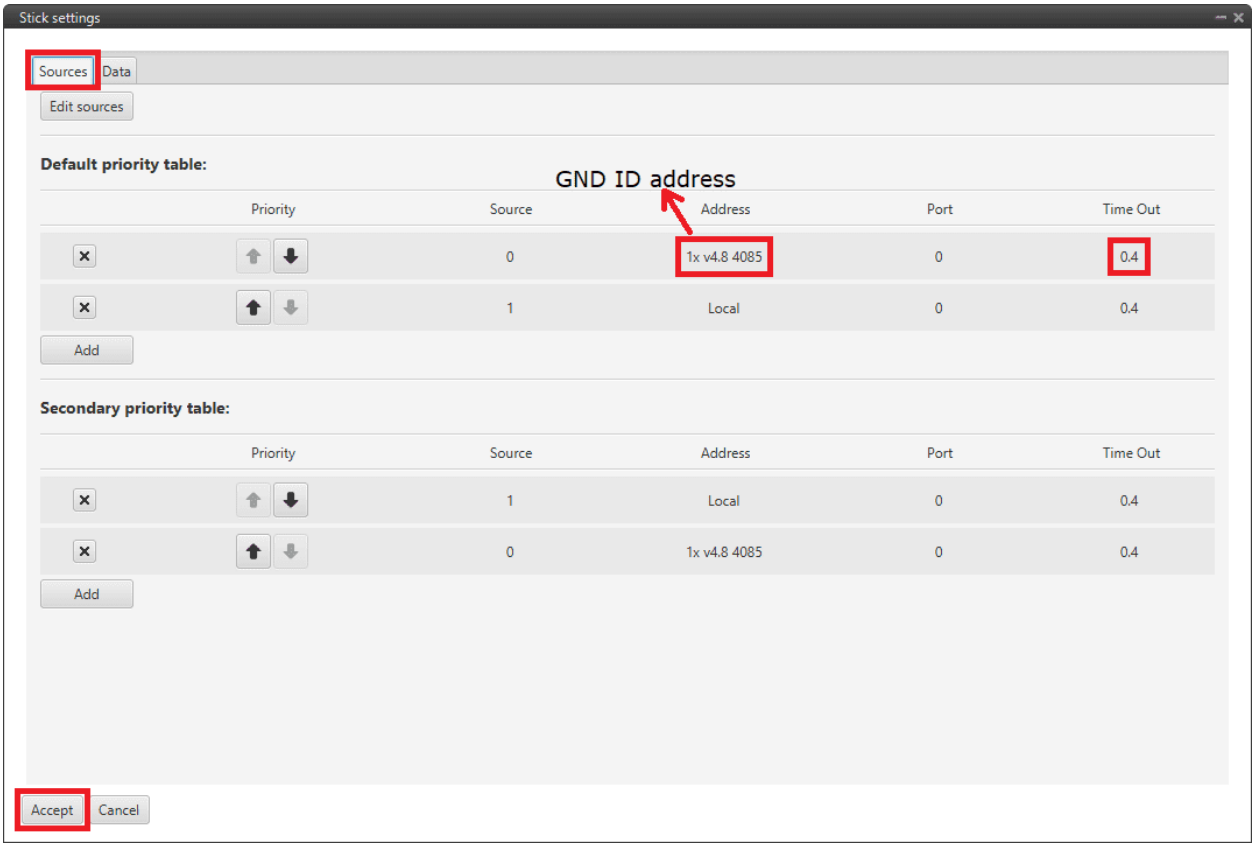


Fig. 107: **General case (Air unit) - Stick block configuration**

Then, if all is correct, users can check that **'Stick not detected'** variable of the **AIR unit** will be true.

Fig. 108: **Stick not detected variable - True**

And that means that the communication between the GND and the AIR unit is correctly configured.

3.7.5.1.2 Simulation case (HIL)

In this case, the user is only using one Autopilot 1x.

So users will have to follow *steps 1, 2 and 3* explained above for the **ground unit**, but also *steps 2 and 3* of the **air unit** configuration. However, instead of entering the **ground unit address**, select the **Local** option.

3.7.5.1.3 On-board PPM receiver case

In this case, follow the steps below to configure the **1x air unit**:

1. Go to Input/Output menu → **Digital Input panel**.

   Make sure that the following parameters have been configured:

   - Producer: **CAP 0**
     - Enabled
     - Select the pin to which the transmitter is connected (normally *EQEP_A*)
     - Edge detection: First rising edge
   - Consumer: **PPM 0**

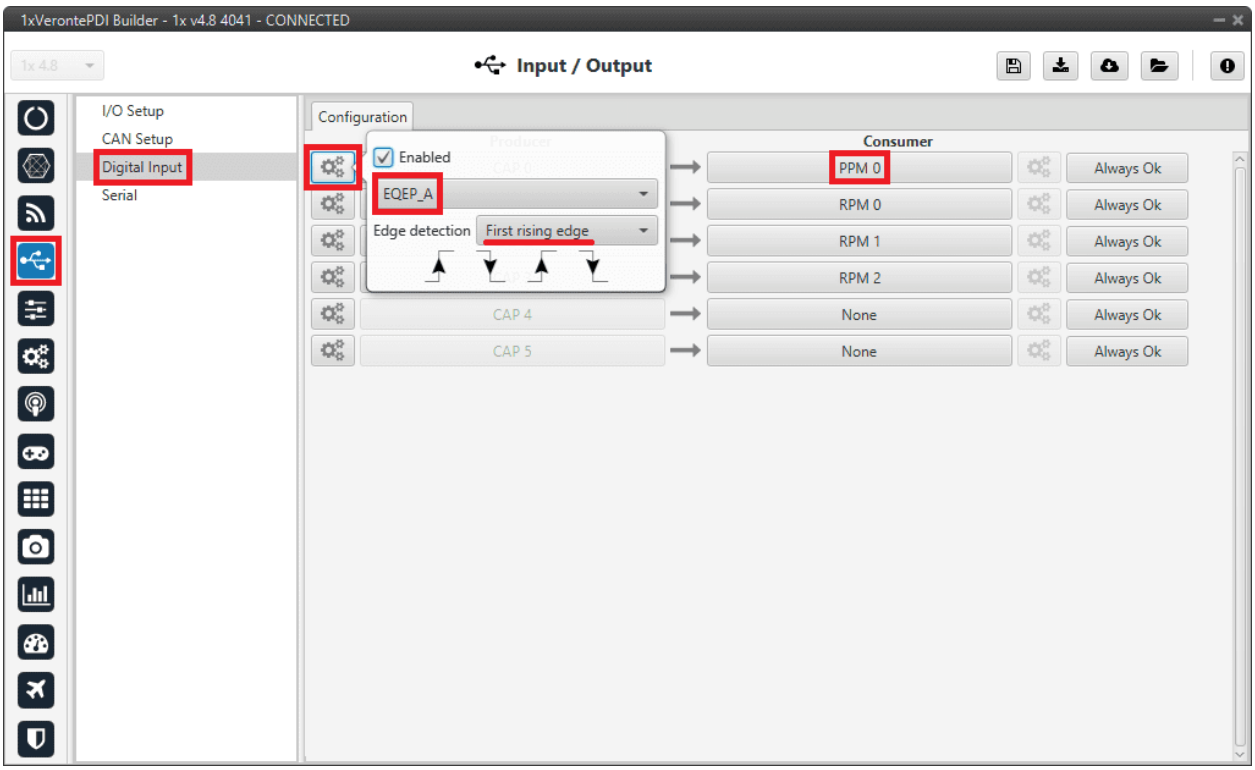


Fig. 109: **On-board PPM receiver case - Digital Input configuration**

2. Go to Connections menu → **GPIO panel**.

   Verify that the pin to which the transmitter is connected, in this case *GPIO 16 (i.e., EQEP A)*, is set as input.

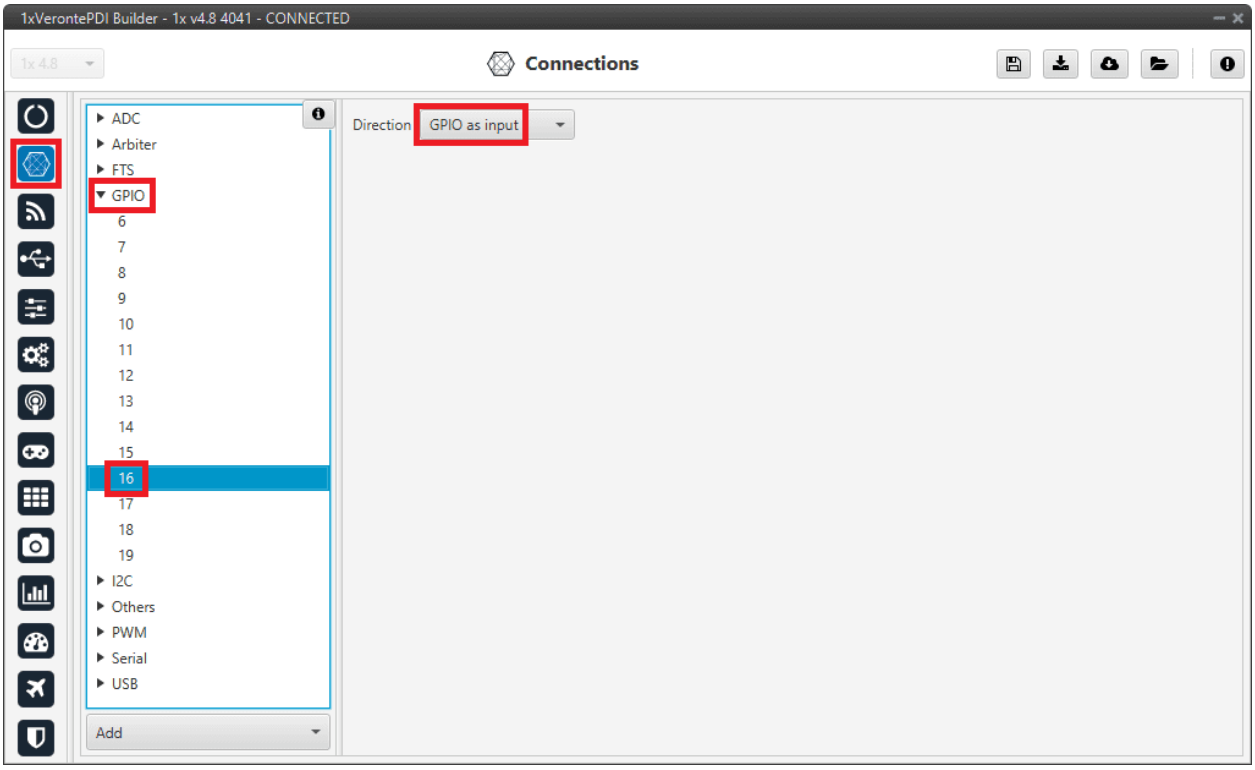


Fig. 110: **On-board PPM receiver case - GPIO configuration**

3. Go to Stick menu → Transmitter 0 panel → **PPM tab**.

   Select the brand of transmitter that applies.

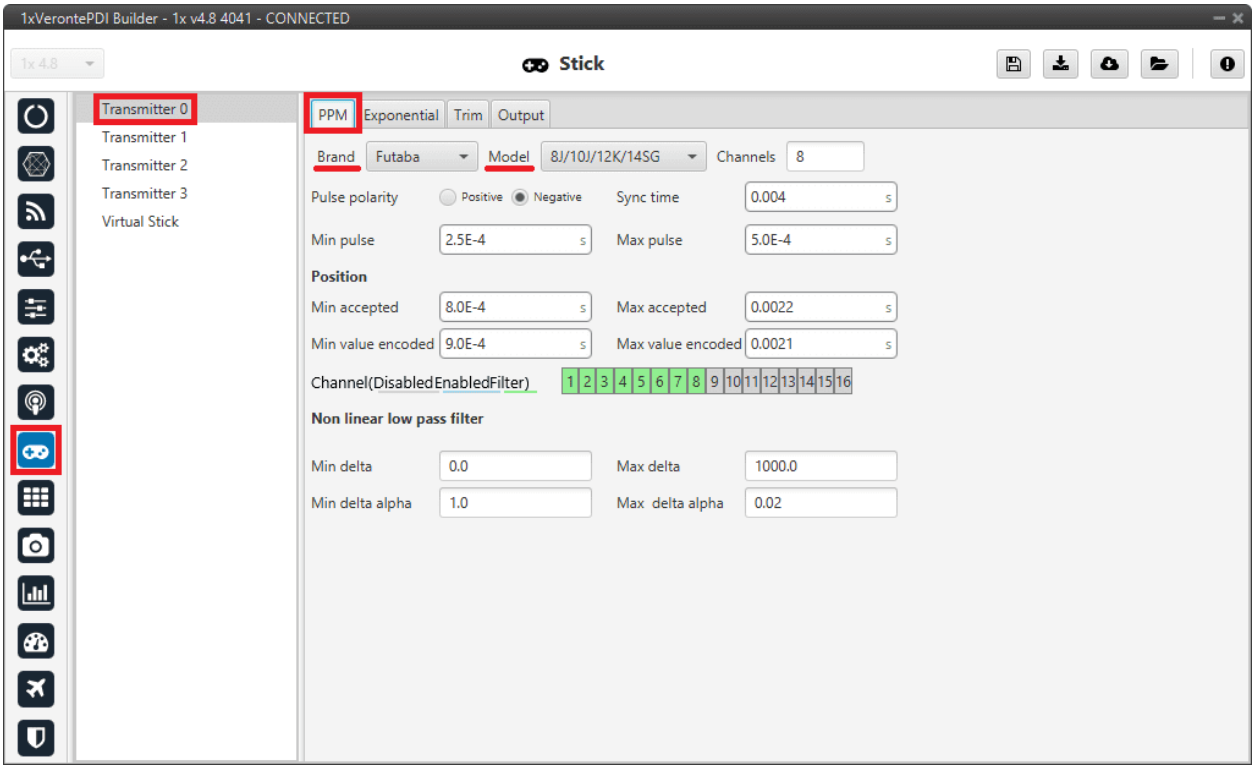


Fig. 111: **On-board PPM receiver case - PPM configuration**

4. Go to Stick menu → Transmitter 0 panel → **Output tab**.

   Just click on **Enable**.

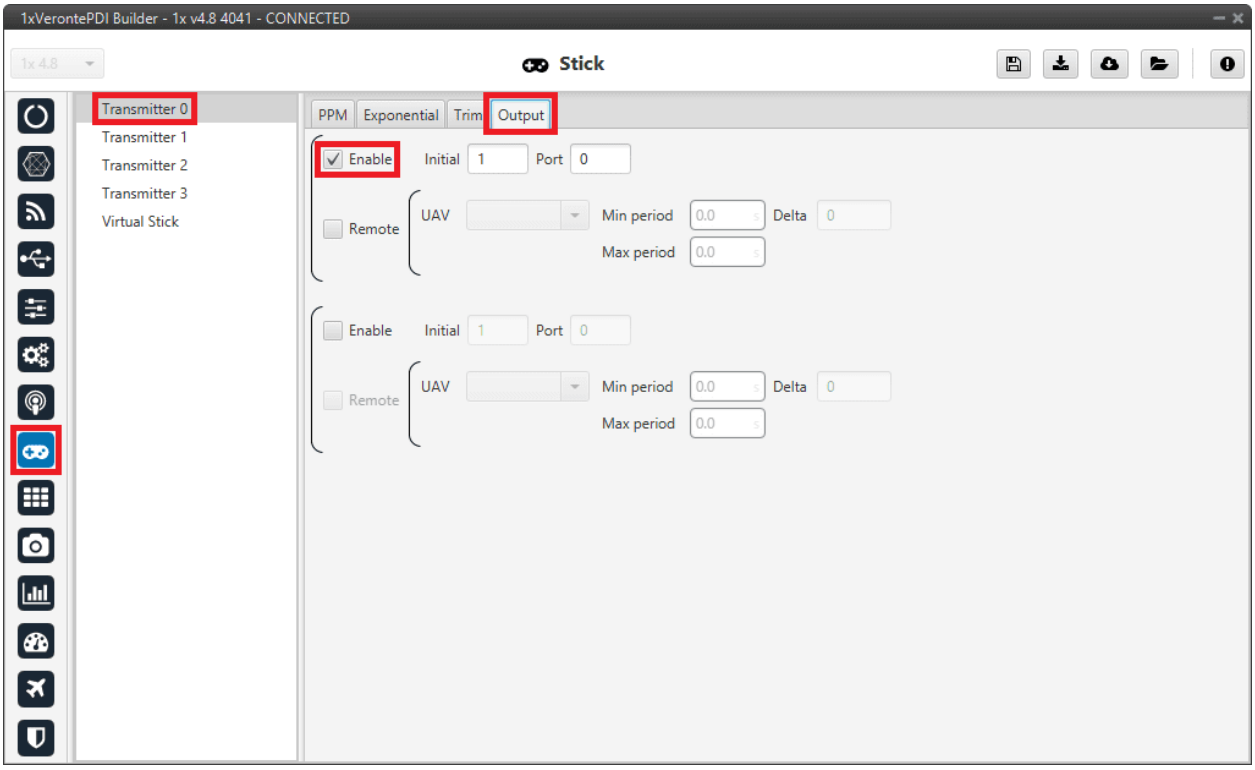


Fig. 112: **On-board PPM receiver case - Output configuration**

If all these settings are correct, users can check that **'Stick PPM 0 not detected'** variable of the **AIR unit** will be true.

Fig. 113: **Stick PPM 0 not detected variable - True**

5. Go to Block Programs menu → **Stick program** → Double click on the **Stick block** → **Edit sources**.

Note: Normally the user has a **Stick Program** where the blocks that are related to the stick are implemented, however, the name of the user's program may be different.

Input the address as **Local** to receive the stick information from that source and put it as the **highest priority** in the priority table. We recommend a Time Out of **0.4 s**.

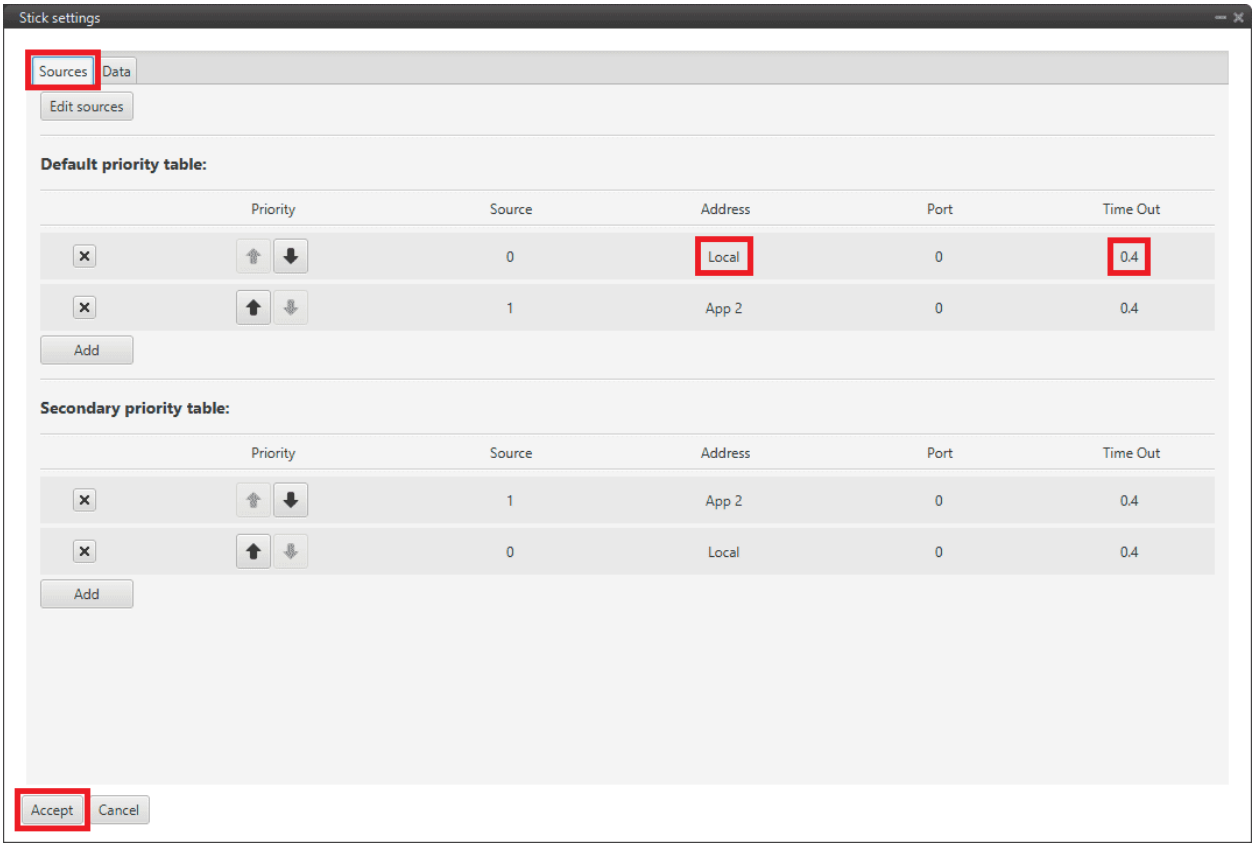


Fig. 114: **On-board PPM receiver case - Stick block configuration**

Then, if all is correct, users can check that **'Stick not detected'** variable of the **AIR unit** will be true.

Fig. 115: **Stick not detected variable - True**

And that means that the communication with the AIR unit is correctly configured.

3.7.5.2 Stick widget

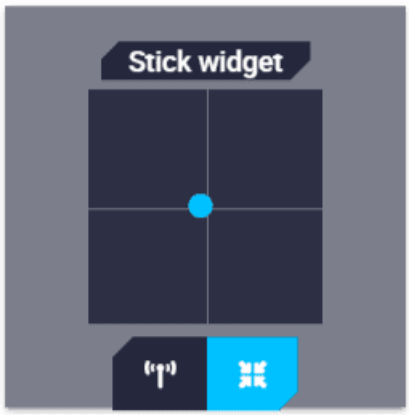


Fig. 116: **Veronte Ops - Stick Widget**

To configure the **Stick widget** of **Veronte Ops** as a control input of **1x**, follow the next steps:

1. Go to Block Programs menu → Stick program → Double click on the **Stick block** → **Edit sources**.

   Note: Normally the user has a Stick Program where the blocks that are related to the stick are implemented, however, the name of the user's program may be different.

   Add a source with **Address 2** (App 2) to receive the stick information from the stick widget (of Veronte Ops) and put it as the highest priority in the priority table. A *Time Out* of 0.4 s is recommended.

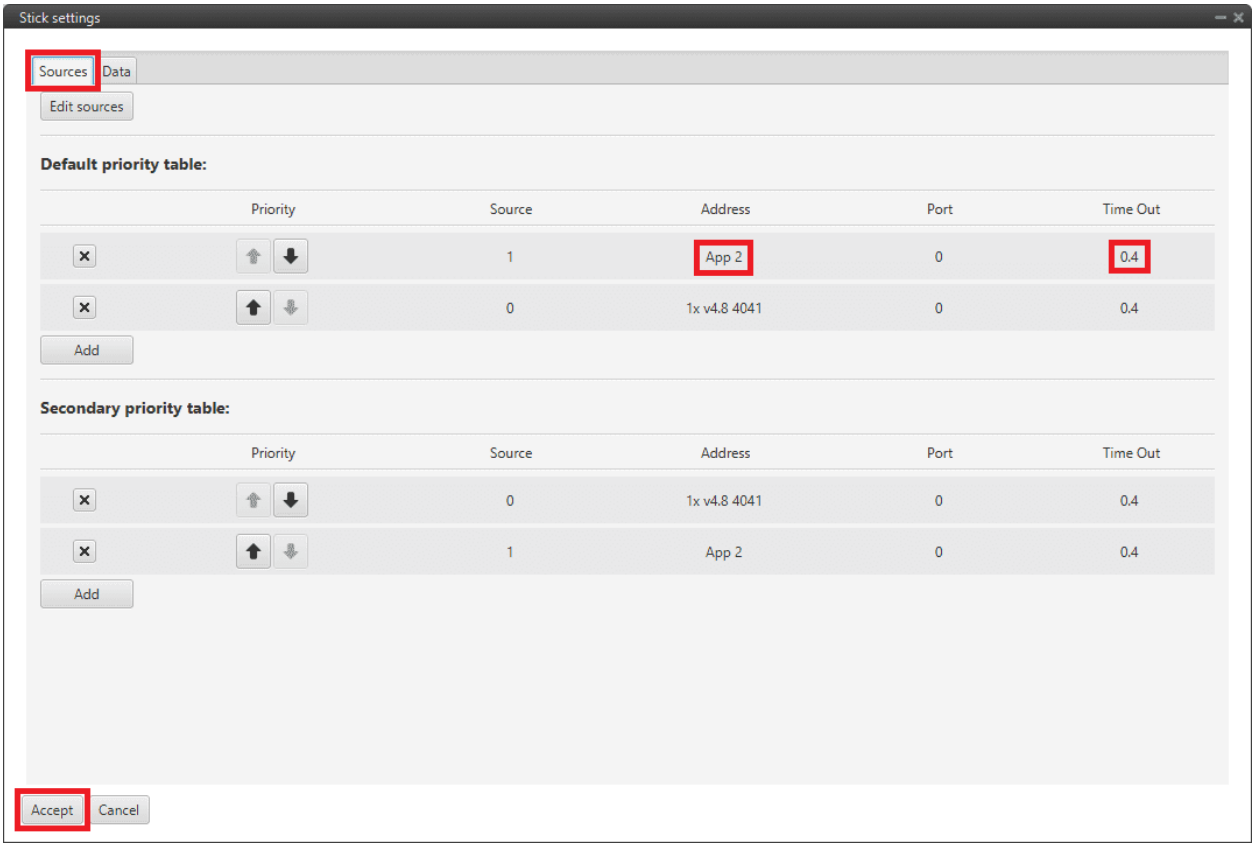


Fig. 117: **Stick block - Stick settings**

2. Configure the **Stick Widget** in **Veronte Ops**. Please find an example of how to configure it in the Stick widget - Integration examples section of the **Veronte Ops** user manual.

3.7.5.3 USB joystick

Veronte software is able to detect USB devices such as joystick. The signals of these devices can be read and configured to send stick information to **Veronte Autopilot 1x** through the **Stick widget** of **Veronte Ops**.

To configure them:

1. Connect the USB joystick to the computer.
2. Go to Block Programs menu → Stick program → Double click on the **Stick block** → **Edit sources**.

Note: Normally the user has a Stick Program where the blocks that are related to the stick are implemented, however, the name of the user's program may be different.

Add a source with **Address 2** (App 2) to receive the stick information from the stick widget (of Veronte Ops) and put it as the highest priority in the priority table. A *Time Out* of 0.4 s is recommended.

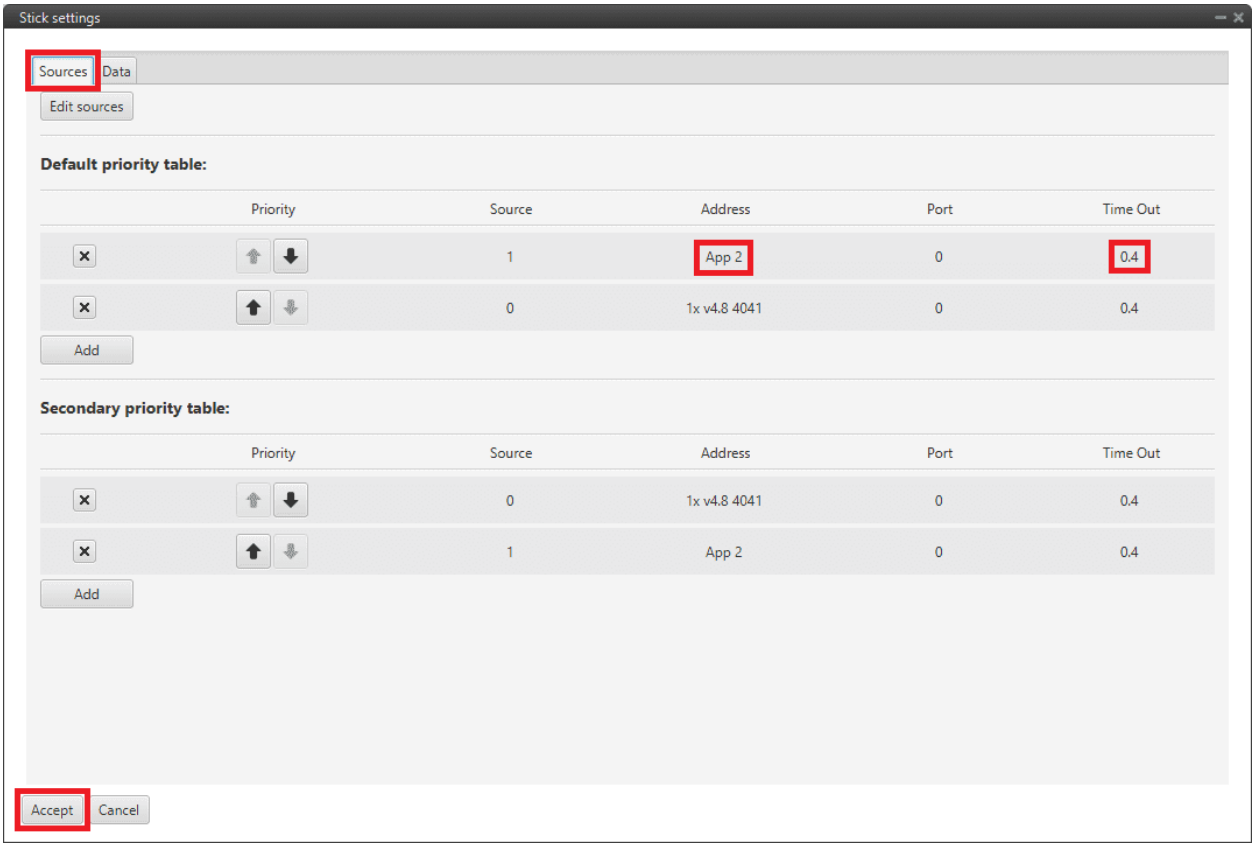


Fig. 118: **Stick block - Stick settings**

3. In **Veronte Ops**, configure the **Stick Widget** to be used with a USB joystick. Please find an example of how to configure it in the USB joystick - Integration examples section of the **Veronte Ops** user manual.

Note: It is also possible to convert USB joystick commands into PPM signals using the Veronte Stick Expander.

For more information about this product, please visit the Stick Hardware Manual or contact sales@embention.com.

3.7.5.4 Virtual Stick

It is necessary to configure a **Virtual Stick** to process the stick information received from sources other than PPM (by CAN, Serial, ADC, etc.). This stick data must be stored in user variables, and once the virtual stick is configured, these will be processed as stick control signals.

To configure a virtual stick, follow the next steps:

1. Go to Stick menu → Virtual Stick panel → **Input Variable tab**.
   - **Enable** the virtual stick
   - Enter an **update period** (**0.02 s** is recommended).
   - Add the variables containing the stick information in **Input Variable** field.
   - Select as **Stick ok bit** a user bit variable to indicate if the Virtual Stick configuration has been properly set.

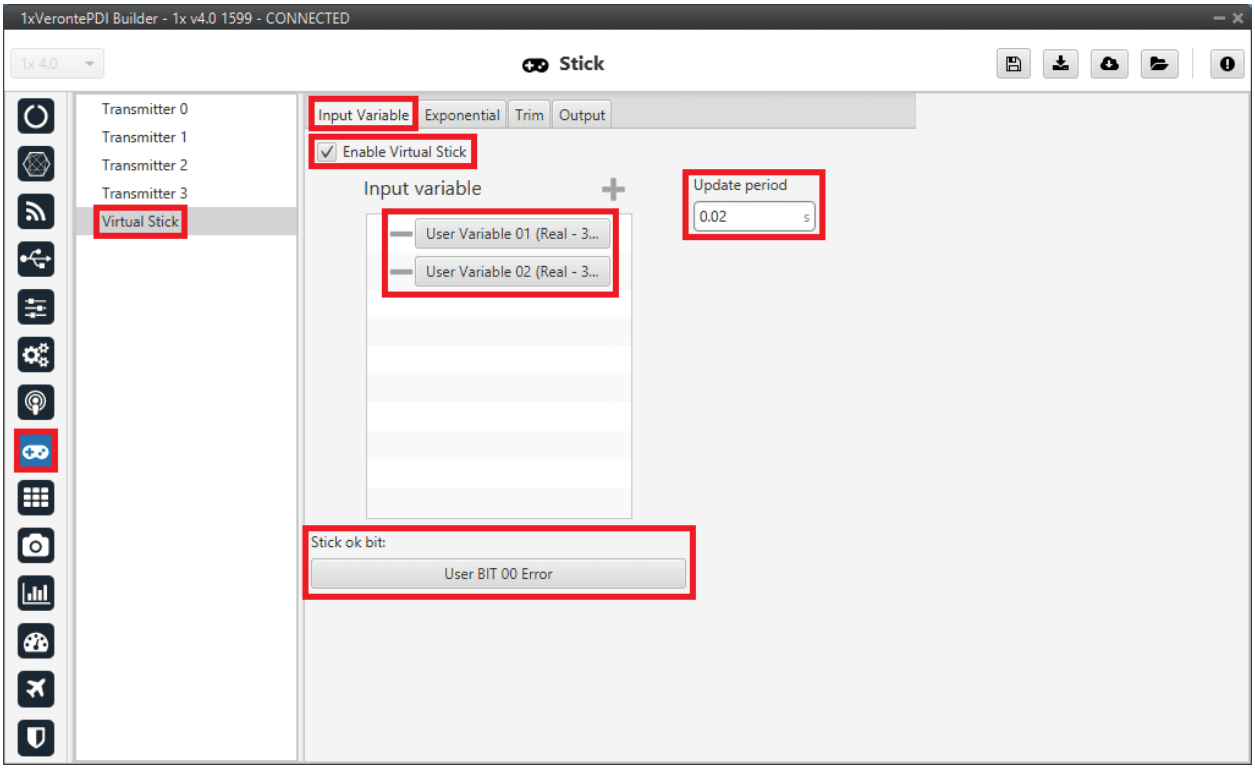


Fig. 119: **Virtual Stick - Input Variable configuration**

2. Go to Stick menu → Virtual Stick panel → **Output tab**.

   Just click on **Enable**.

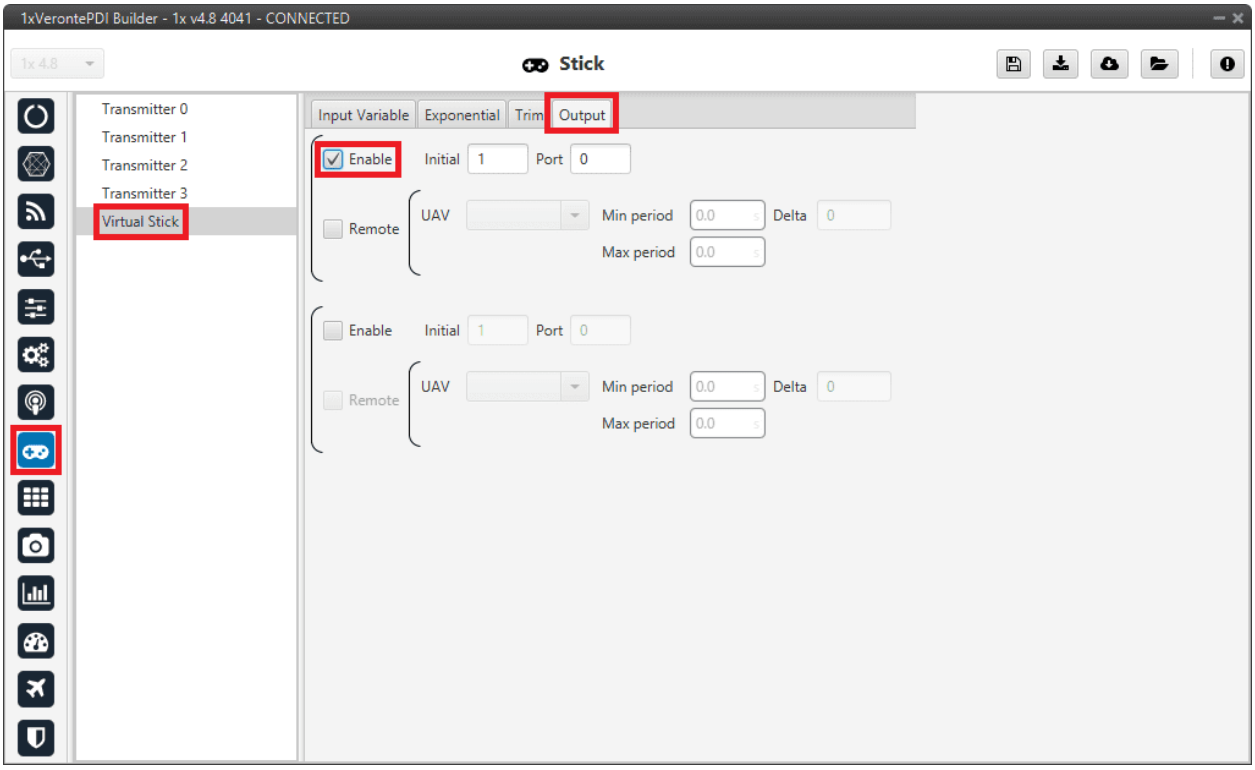


Fig. 120: **Virtual Stick - Output configuration**

Note: If all these settings are correct, users can check the variable previously set as *Stick ok bit* is true.

---

3. Go to Block Programs menu → Stick program → Double click on the **Stick block** → **Edit sources**.

Note: Normally the user has a **Stick Program** where the blocks that are related to the stick are implemented, however, the name of the user's program may be different.

Add a source with address **Local** to receive the stick information from the configured variables and put it as the **highest priority** in the priority table. A *Time Out* of **0.4 s** is recommended.

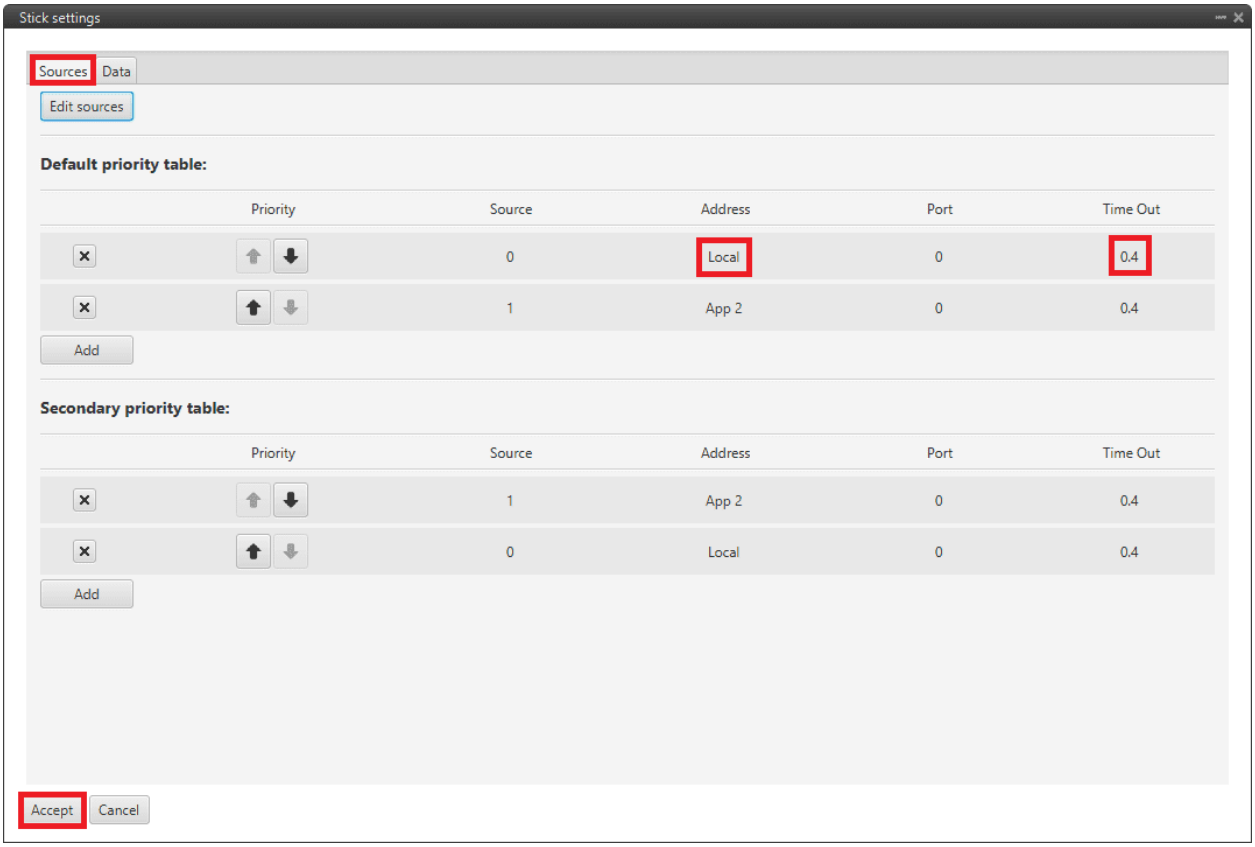


Fig. 121: **Virtual Stick - Stick block configuration**

Then, if all is correct, users can check that *Stick not detected* variable is true, which means that the communication is correctly configured.

3.7.6 Veronte products

3.7.6.1 Autopilot 4x

Veronte Autopilot 4x is a redundant system that includes 3 complete **Veronte Autopilots 1x** modules fully integrated with dissimilar arbiters.

Normally, the configuration development is done for one **Autopilot 1x** unit, however, to integrate it into a **Veronte Autopilot 4x**, several modifications must be made to the 3 units for a correct operation of the **4x** system.

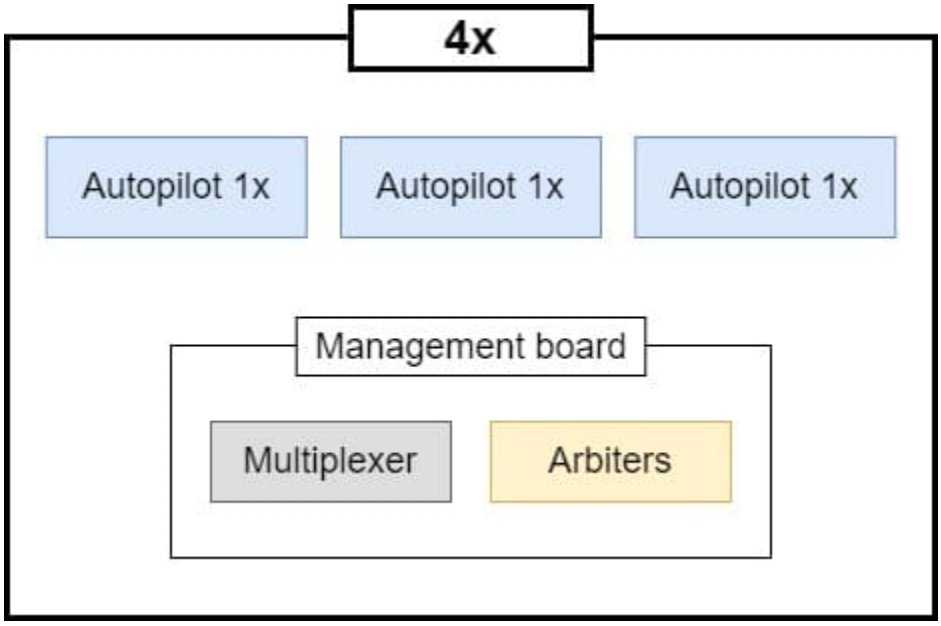


Fig. 122: **Autopilot 4x basic diagram**

This section presents:

- *Autopilots 1x configuration* ⇒ Configuration required on an **Autopilot 1x** as far as being part of an **Autopilot 4x** is concerned:
  - *Definition of the 4x group*
  - *Control and communication between autopilots within the 4x group*
  - *Communication between Autopilots 1x and Arbiter*

Caution: Arbitration configuration is conducted in Arbiters using the **4x PDI Builder** software. For more information on this, please refer to the Arbitration configuration - Integration examples section of the **4x PDI Builder** user manual.

- *Configuration for external radio communication through RS232* ⇒ Configuration to establish communication between **Autopilot 4x** and an external radio connected to its multiplexed RS232, as well as CAN communication between the **Autopilots 1x**.
- *Arbiters communication* ⇒ Configuration necessary to establish the connection between the PC and Arbiters using one of the **Autopilots 1x** as a "tunnel".

Important: These examples are valid for **Veronte Autopilot 4x** `hwv 1.8` and higher.

This is because the **4X Selected** bit variable used throughout this explanation has been introduced from **Autopilot 4x** `hwv 1.8`.

However, if users have a **4x** with a **lower hwv**, they can follow all the steps by simply replacing this bit with the corresponding bit coming from the arbiter status message. For more information on the status message, see Status Message - CAN Bus protocol section of the **4x Software Manual**

3.7.6.1.1 Autopilots 1x configuration

This example describes the steps necessary to adapt a complete and functional configuration of an **Autopilot 1x** to that of an **Autopilot 1x** within an **Autopilot 4x**.

The following schema broadly summarizes the configuration that will be explained:

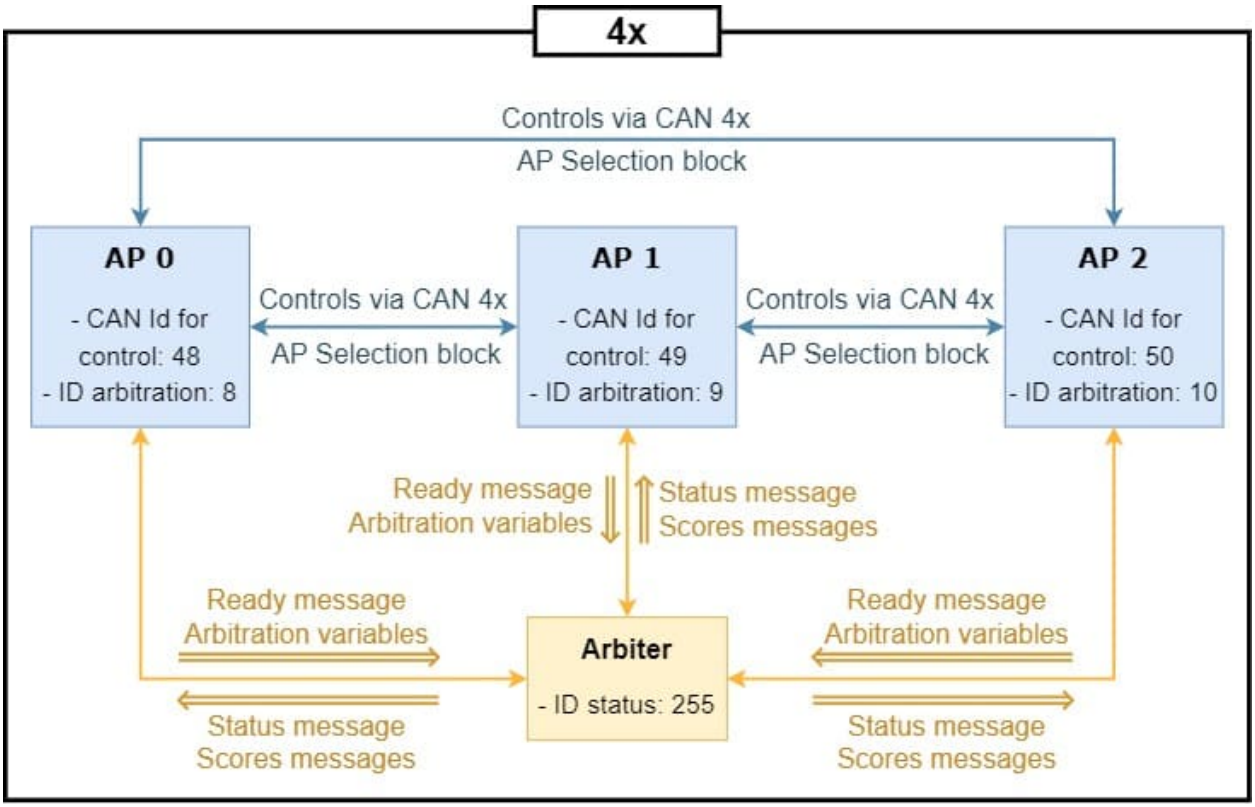


Fig. 123: **Autopilot 4x - Autopilots 1x configuration diagram**

Important: All the **Ids** represented in the schema are the **default** ones and those that will be used throughout this example, but users can change them as they wish.

Nonetheless, the Ids designated for arbitration in this configuration must **match** those entered in the **arbitration configuration** of the **4x PDI Builder** software.

	AP 0	AP 1	AP 2
Ids for arbitration	8	9	10
CAN Ids for control	48	49	50

In this example, **AP 0** is configured, so the **Ids** associated with it are **8** for arbitration and **48** for control.

3.7.6.1.1.1 Definition of the 4x group

Firstly it is necessary to indicate that this **Veronte Autopilot 1x** is part of a **Veronte Autopilot 4x**.

For this purpose, go to Control menu → Modes panel → **4x Veronte tab**:

1. Enter the **Arbiter address**:

   ⇒ **Arbiter A address** = 50000 + Serial number of **4x**

   ⇒ **Arbiter B address** = 54000 + Serial number of **4x**

   Note: Arbiter A is configured in this example.

2. Activate the **Enable output overwrite** so that the autopilots that are not in command (the ones not selected) give the same control output as the selected autopilot. This way, when the selected autopilot changes (due to arbitration), the transition in control commands is smooth.
3. Add the 3 **Autopilots 1x** that are part of the **Autopilot 4x** by clicking on **Add UAV** and specifying their **addresses** (address = Serial number).
4. For each autopilot set the **CAN Id** to be used for the control messages.

   Note: *Control Ids* described above have been entered here.

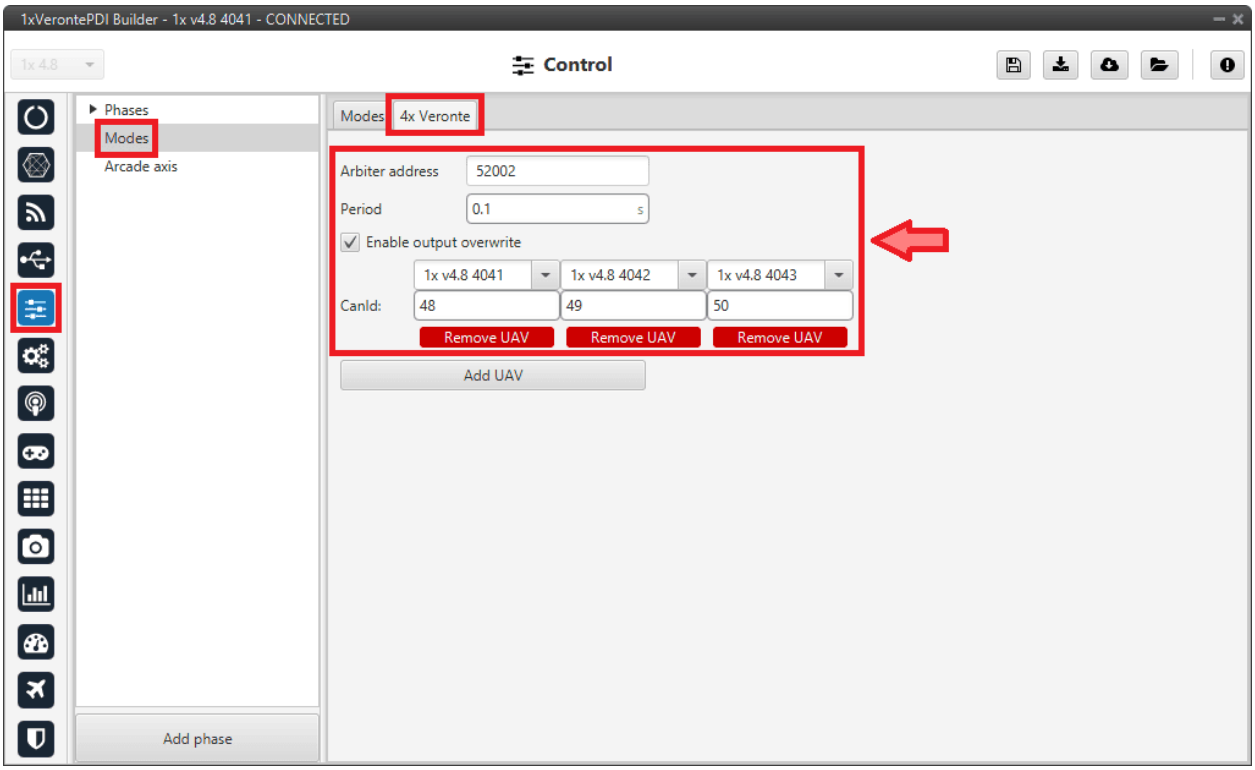


Fig. 124: **Autopilots 1x configuration - 4x group**

3.7.6.1.1.2 Control and communication between autopilots within the 4x group

Then, in order for the **Autopilots 1x** to send control commands to each other, it is necessary to configure the CAN communication. To do this:

Note: Remember that this is an example of the configuration for **AP 0**.

---

5. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   - For **sending commands** from this autopilot, connect the **CAN 4x** producer to an **Output filter** consumer configured to **CAN A**, in this case *Output filter 5* has been selected.

   - For **receiving commands** from the other autopilots, connect an **Input filter** producer to the **CAN 4x** consumer, in this case *Input filter 0* has been selected.

     Therefore, in order to read control command messages from all autopilots, it is necessary to correctly configure the Id and mask in this Input filter. That is, by setting the Id to **48** and the mask to **2044** (decimal format), command messages from **48** to **50** will be read.

     Note: The following table shows the Ids and mask in binary and decimal format used in this example.

	Decimal format	Binary format
AP 0 control Id	48	000 0011 0000
AP 1 control Id	49	000 0011 0001
AP 2 control Id	50	000 0011 0010
Mask	2044	111 1111 1100

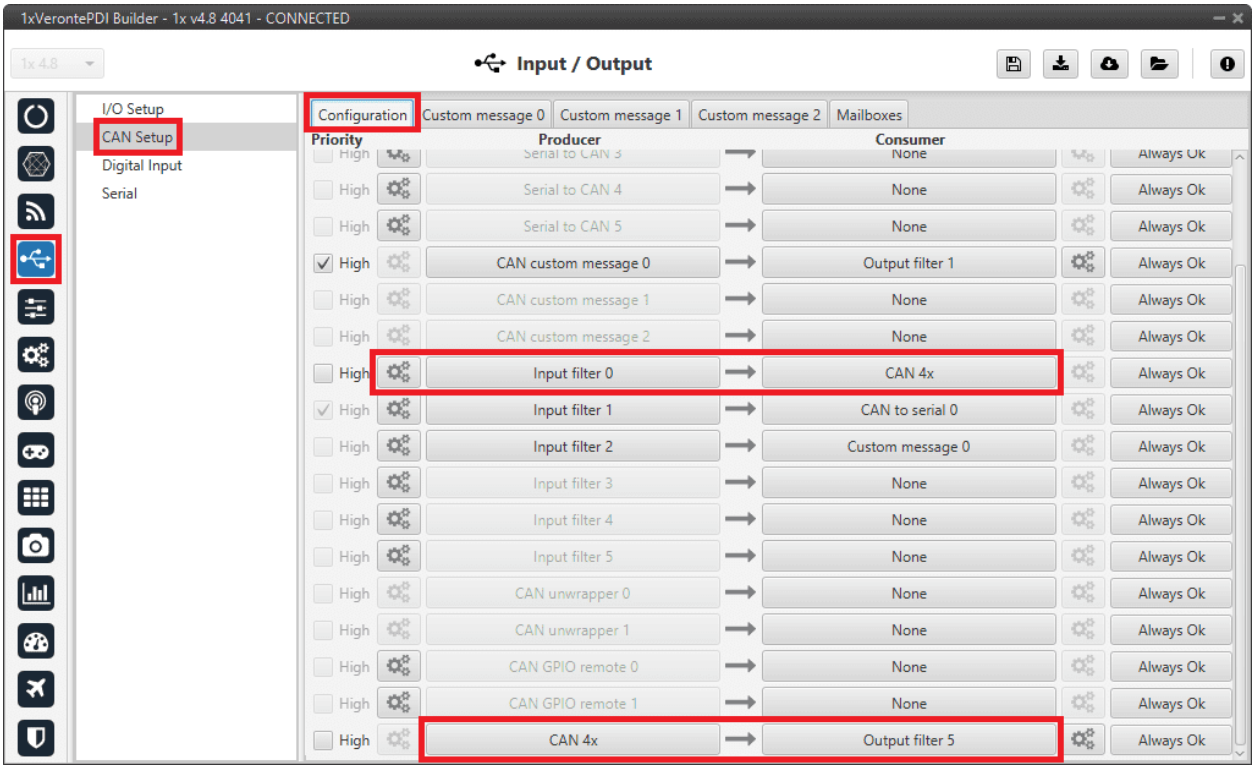


Fig. 125: **Autopilots 1x configuration - CAN Setup configuration**

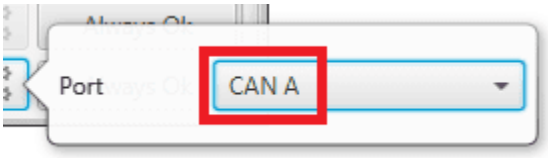


Fig. 126: **Autopilots 1x configuration - Output filter configuration**

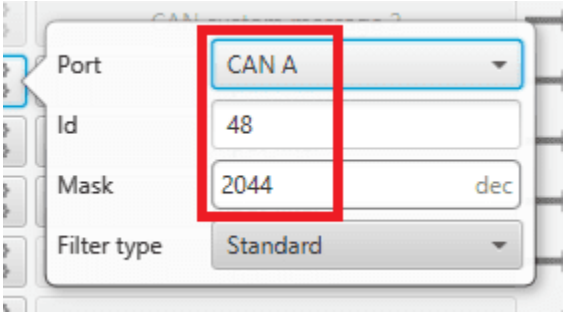


Fig. 127: **Autopilots 1x configuration - Input filter configuration**

6. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Configure at least **4** mailboxes in **CAN A** bus with the same configuration as the Input filter to correctly receive the control command messages:

⇒ **ID: 48 DEC**

⇒ **Mask: 111 1111 1100 BIN**

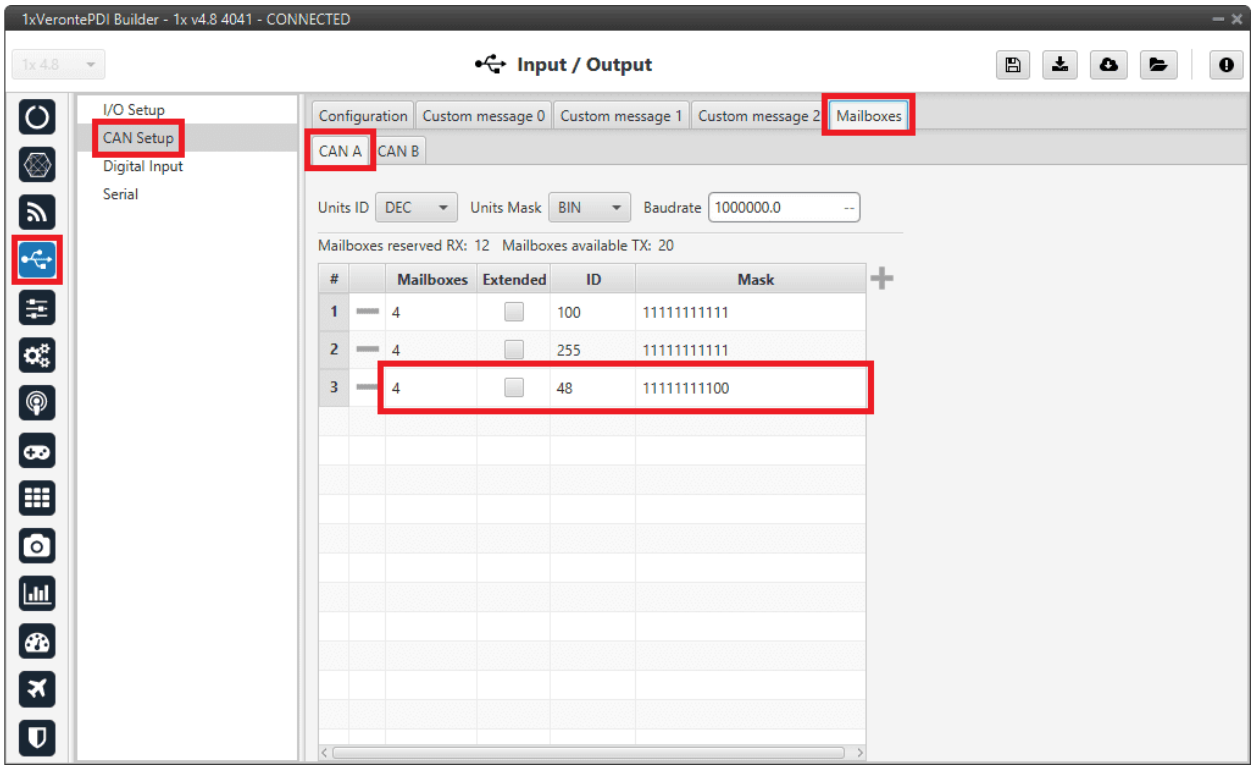


Fig. 128: **Autopilots 1x configuration - Mailboxes configuration**

7. Go to **Block Programs menu**.

   - In all programs using "**Control Output uX**" variables, it is necessary to add the **AP Selection** block between these variables and the corresponding control block, usually the **PID Static** block. For example:

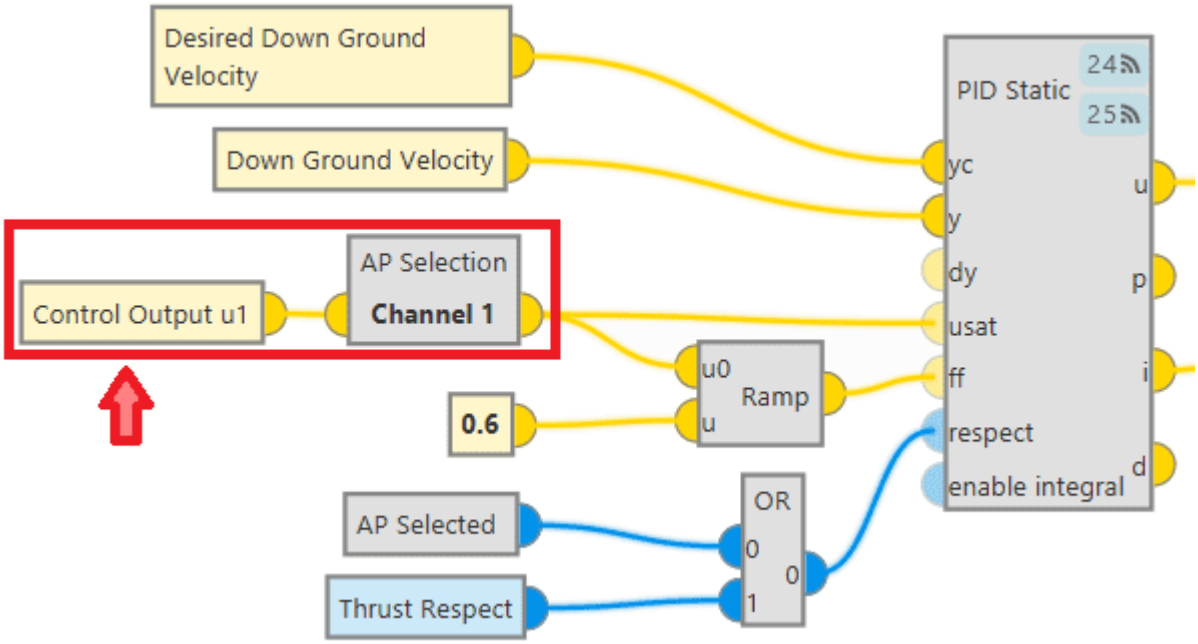


Fig. 129: **Example of AP Selection block connection**

This connection is performed to **always** use the **control output of the selected AP as input** to the corresponding control block, usually the **PID Static** block. Therefore, the input of the **AP Selection** block is the control output of the AP in the configuration and the output is the control output of the selected AP.

In the configuration of this block, users only have to select the channel through which the information related to this control output is being shared. For example, if **Control Output u1** is connected to this block, **channel 1** must be selected. For more information on this block, see the *AP Selection - Mode/AP Selection blocks* in the **Block Programs** section of this manual.

Warning: This block only allows to select the same channel once in all programs, i.e. this block can only be used once for each control output.

Consequently, if a control output is used more than once in the programs, users must create an auxiliary program in which they connect each **Control Output uX** to the **AP Selection** block and save this "transformation" in user variables. Then, they will be able to use these user variables throughout the programs as many times as they need.

Below is an example in which all the control outputs used in the configuration have been associated to renamed user variables:

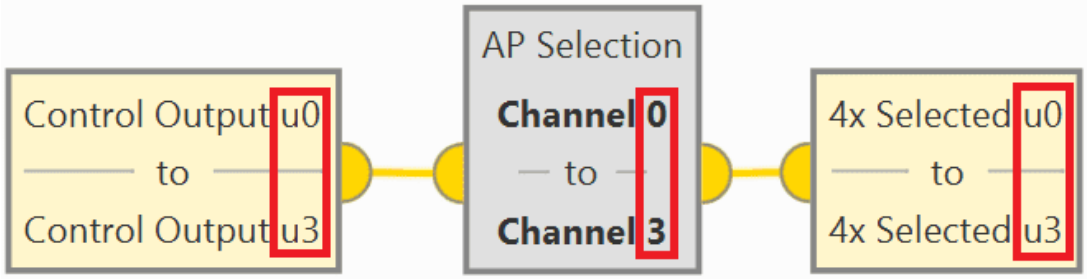


Fig. 130: **Example of AP Selection block connection to several Control Outputs**

These users variables have been previously renamed in the UI menu → Variables panel → **Real Vars tab**:

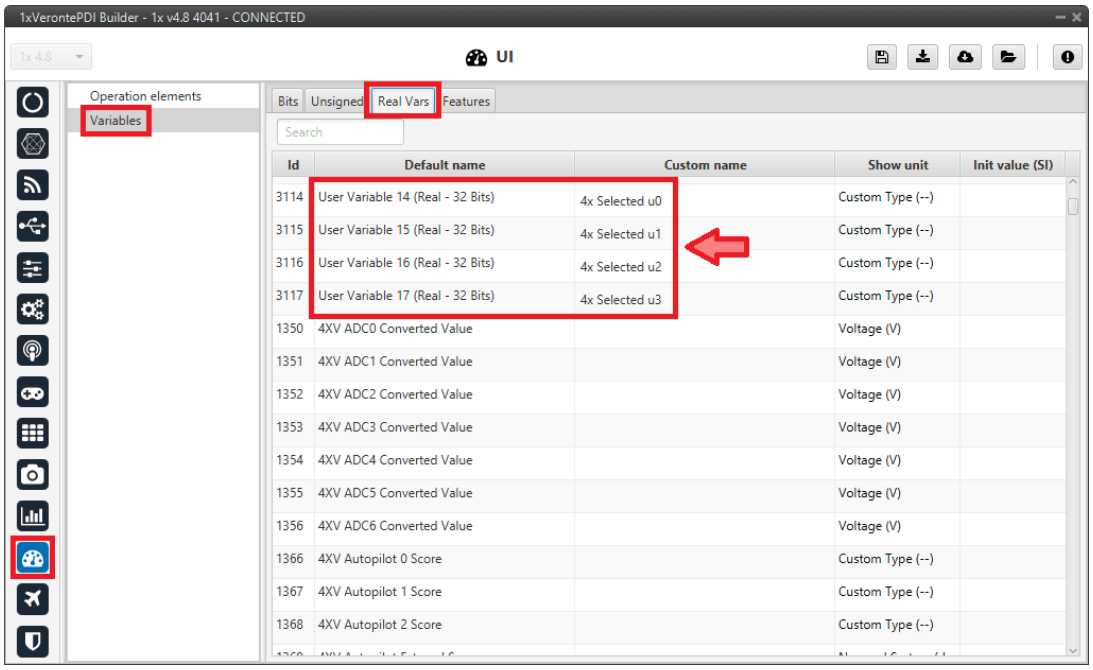


Fig. 131: **User variables renamed**

- Furthermore, build the following **library block**, "*AP Selected*", to be used as the **respect** input of the corresponding control block, usually the **PID Static** block.
  1. Go to the **Library tab**.
  2. Add a new block by clicking on ✚ and rename it as desired, in this example *AP Selected*.
  3. Insert in this custom block the "**NOT**" and "**Read Bit**" blocks.
  4. Right click on the custom block and add an **output** to it.
  5. Connect the "**Read Bit**" block as **input** of the "NOT" logic block and as iy¡ts **output**, the output that

has been added to the **custom block**.

6. Finally, select the **4X Selected** variable for the "Read Bit".

For more information on library blocks, see *Library blocks - Block Programs* section of this manual.

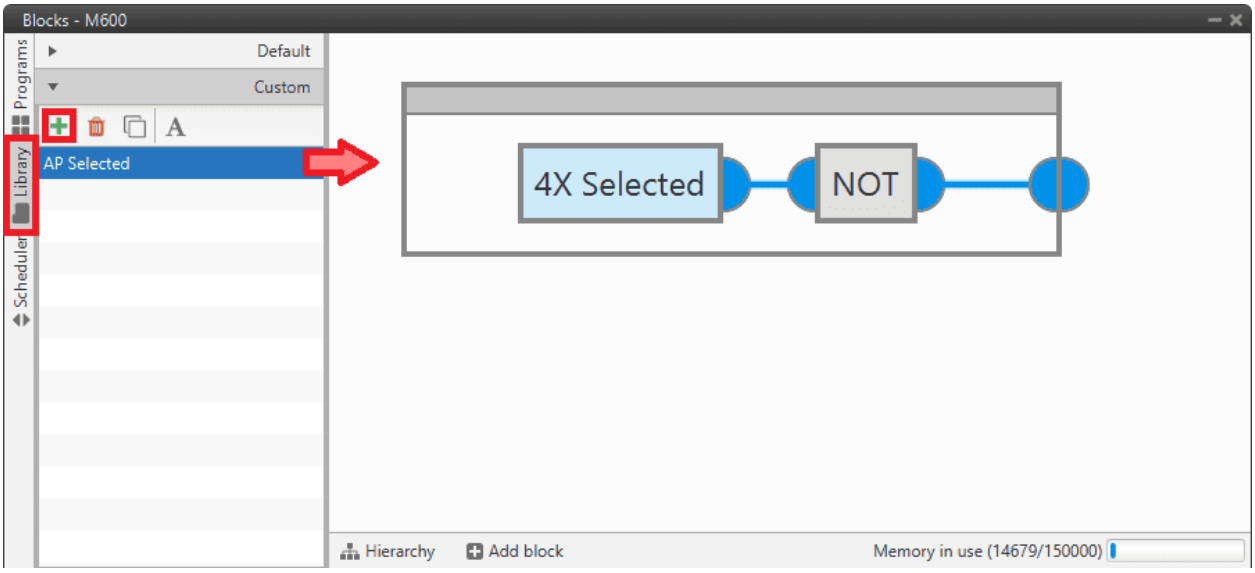


Fig. 132: **Autopilots 1x configuration - Library block**

The overall result of these blocks related to **4x** should look like this:

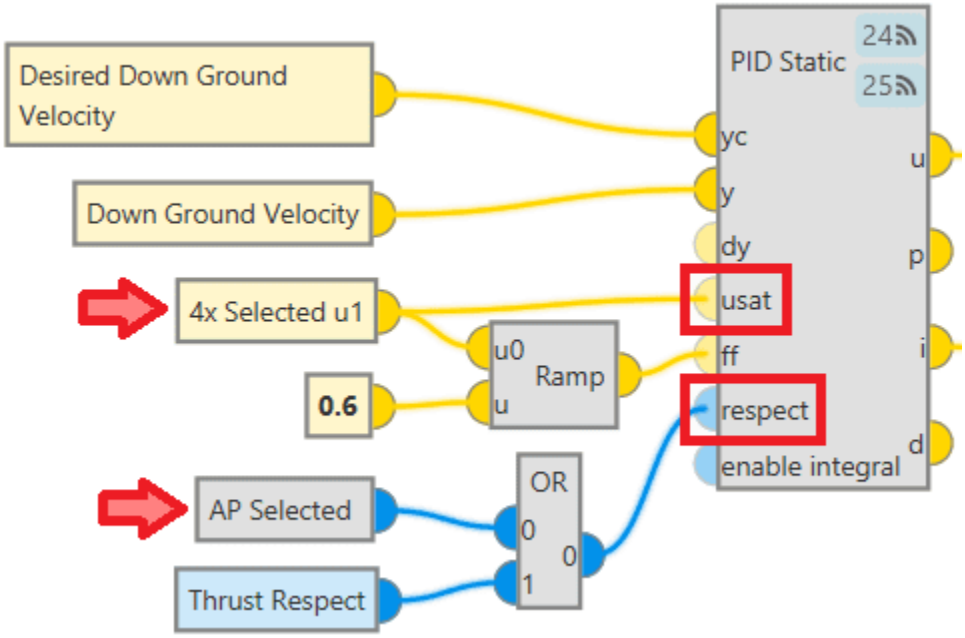


Fig. 133: **Autopilots 1x configuration - Blocks**

With all this block configuration, the **expected behavior** is to avoid jumps in the control commands when the selected autopilots change (due to arbitration). This is achieved by always having as **usat** input of the **PID Static** block the control output of the selected AP, since the integral term of the 3 **Autopilots 1x** of **group 4x** always remains the same.

And the logic that follows is this:

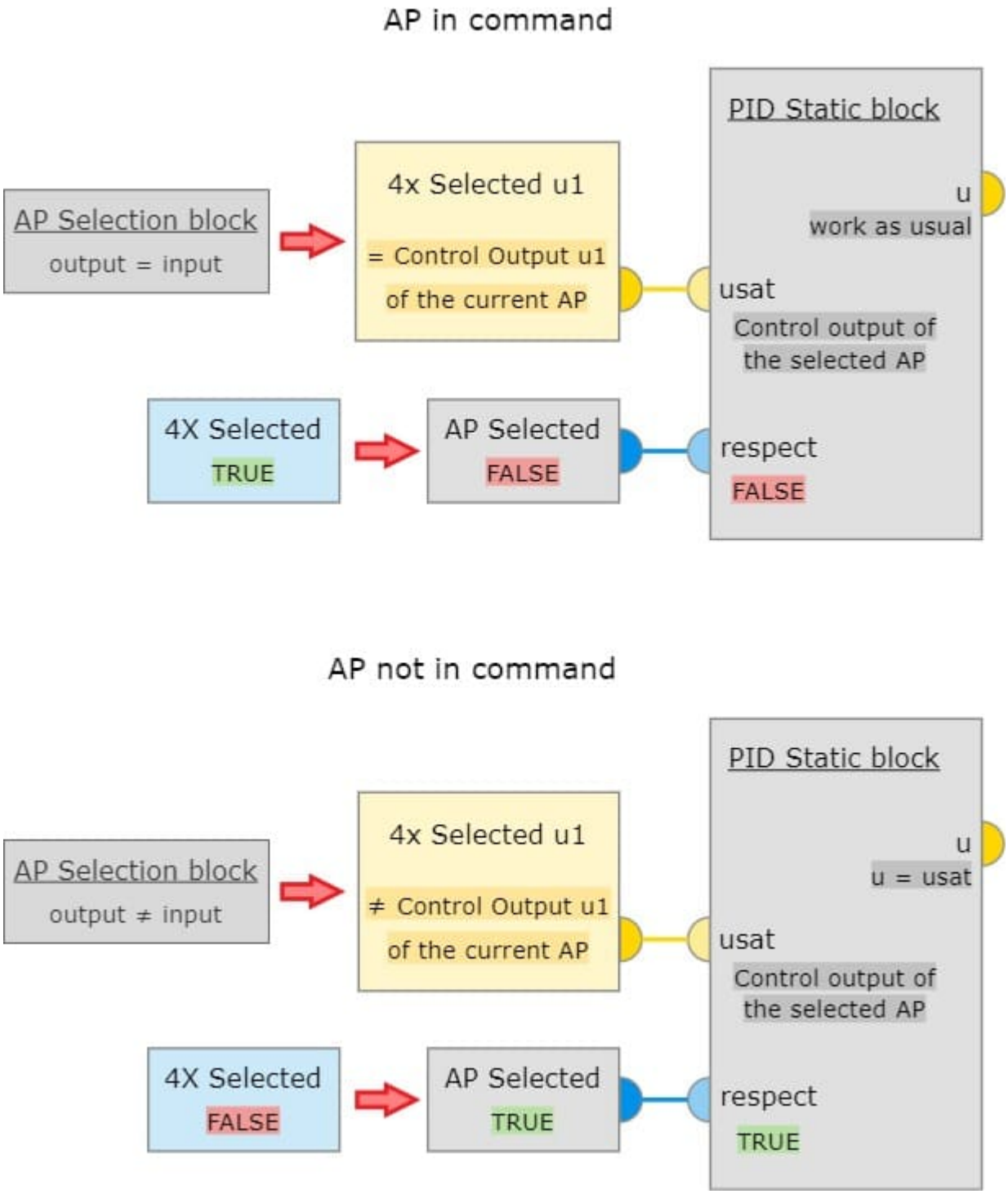


Fig. 134: **Autopilots 1x configuration - Blocks logic**

3.7.6.1.1.3 Communication between Autopilots 1x and Arbiter

Finally, the CAN communication must be correctly configured to be able **to send** the Ready and Arbitration messages from the autopilots to the arbiter and **to read** the Status and Score messages from the arbiter:

Note: The structure/configuration of these messages must be done following the protocol defined in the CAN Bus protocol section of the **4x Software Manual**.

Remember that this is an example of the configuration for **AP 0**.

---

8. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.
    - For **sending** messages **to the arbiter**, connect a **CAN custom message** producer to an **Output filter** consumer, in this case *CAN custom message 0* and *Output filter 1* have been selected.
    - For **receiving** messages **from the arbiter**, connect an **Input filter** producer to a **Custom message** consumer, in this case *Input filter 2* and *Custom message 0* have been selected.

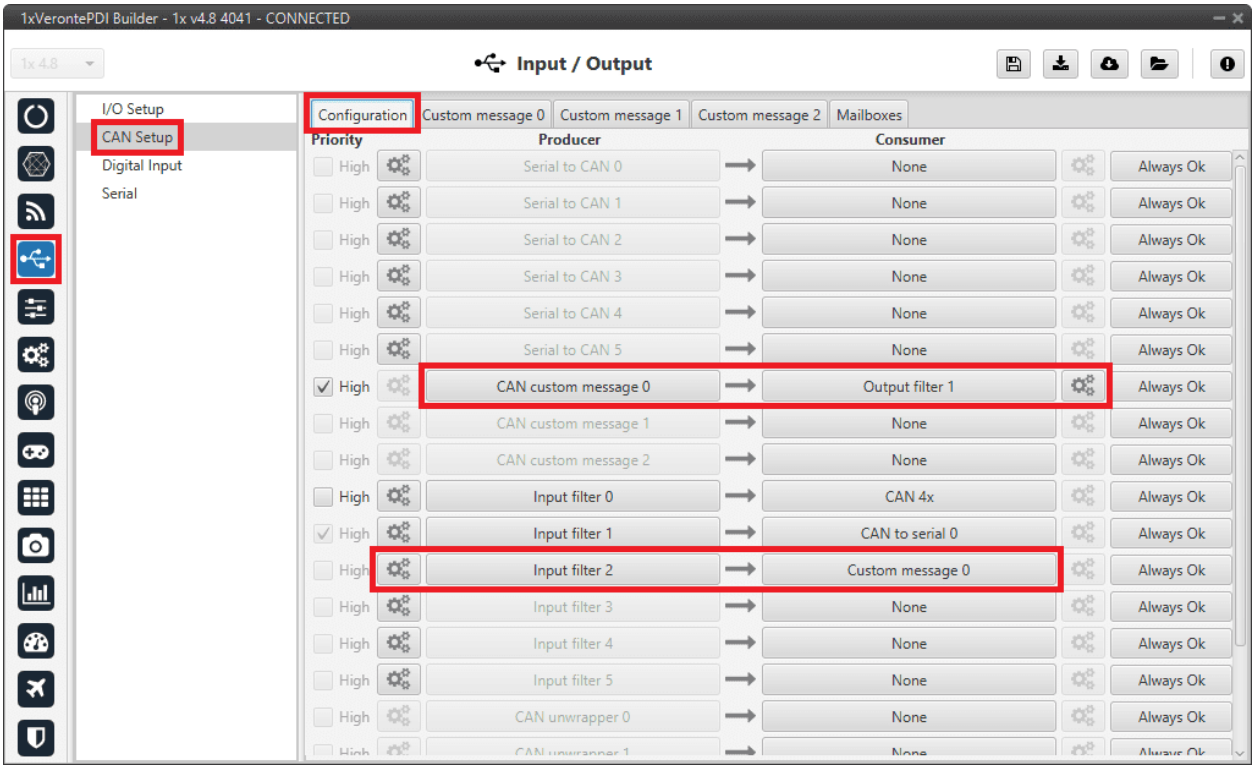


Fig. 135: **Autopilots 1x configuration - CAN Setup configuration**

- The **Output filter** is configured to **both** CAN ports:

Fig. 136: **Autopilots 1x configuration - Output filter configuration**

- The Id set in this **Input filter** must match the *status Id* set in the arbiter configuration. In this example **Id 255** has been chosen:

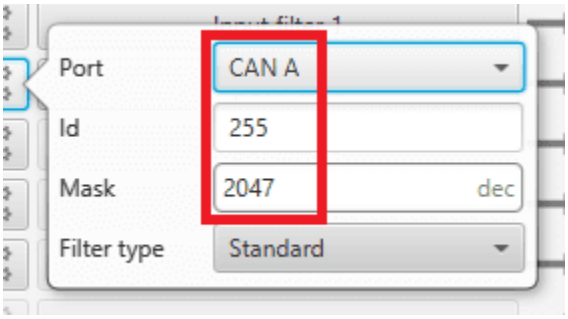


Fig. 137: **Autopilots 1x configuration - Input filter configuration**

9. Go to Input/Output menu → CAN Setup panel → **Custom Message 0 tab** (since *CAN custom Message 0* has been connected to the output filter) → **TX** to configure **the messages to be sent to the arbiter**.

- In this example, 5 messages with **CAN ID 8** (*arbitration Id* of the AP 0), **Little endian** and **Period 0.05 s** have been added. More information on CAN messages configuration can be found in the *TX/TX Ini Messages (Custom Messages) - Input/Output* section of this manual.

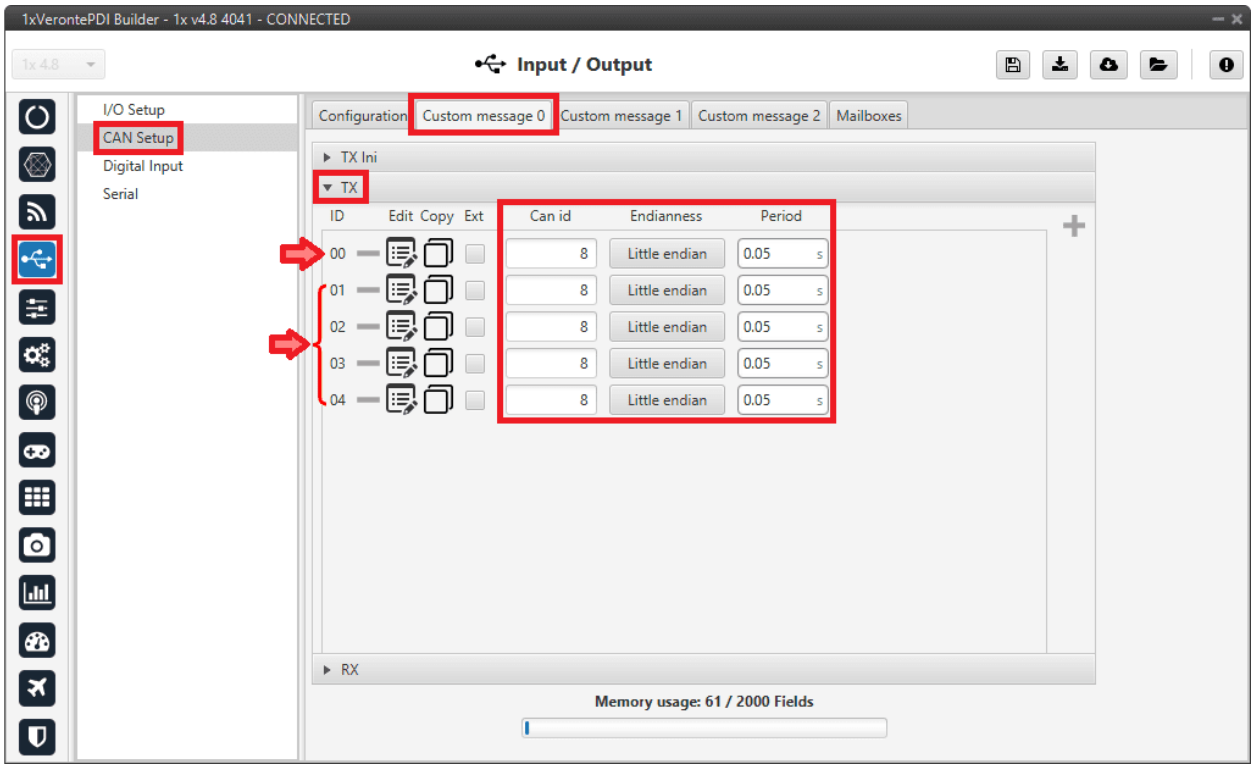


Fig. 138: **Autopilots 1x configuration - CAN custom message 0 (TX) configuration**

- Then, to create each message, click on its corresponding button:

– **ID 00** ⇒ **Ready message**. It must be built following the Ready Message - CAN Bus protocol section of the **4x Software Manual**.

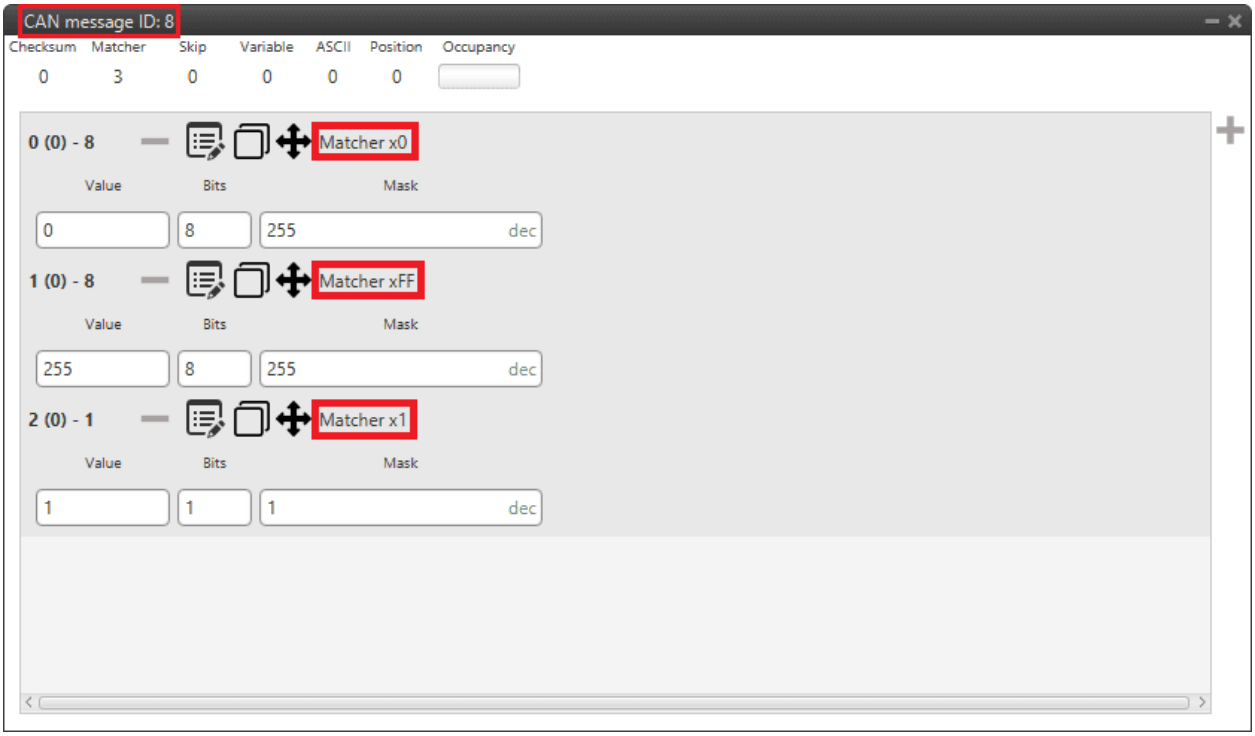


Fig. 139: **Autopilots 1x configuration - Ready message**

– **ID 01-04** ⇒ **Arbitration variables messages**. They must be built following the Arbitration Message - CAN Bus protocol section of the **4x Software Manual**.

  In this example, **Roll**, **Pitch**, **Position not fixed** and a **Custom Arbitration Variable** have been configured as arbitration messages:

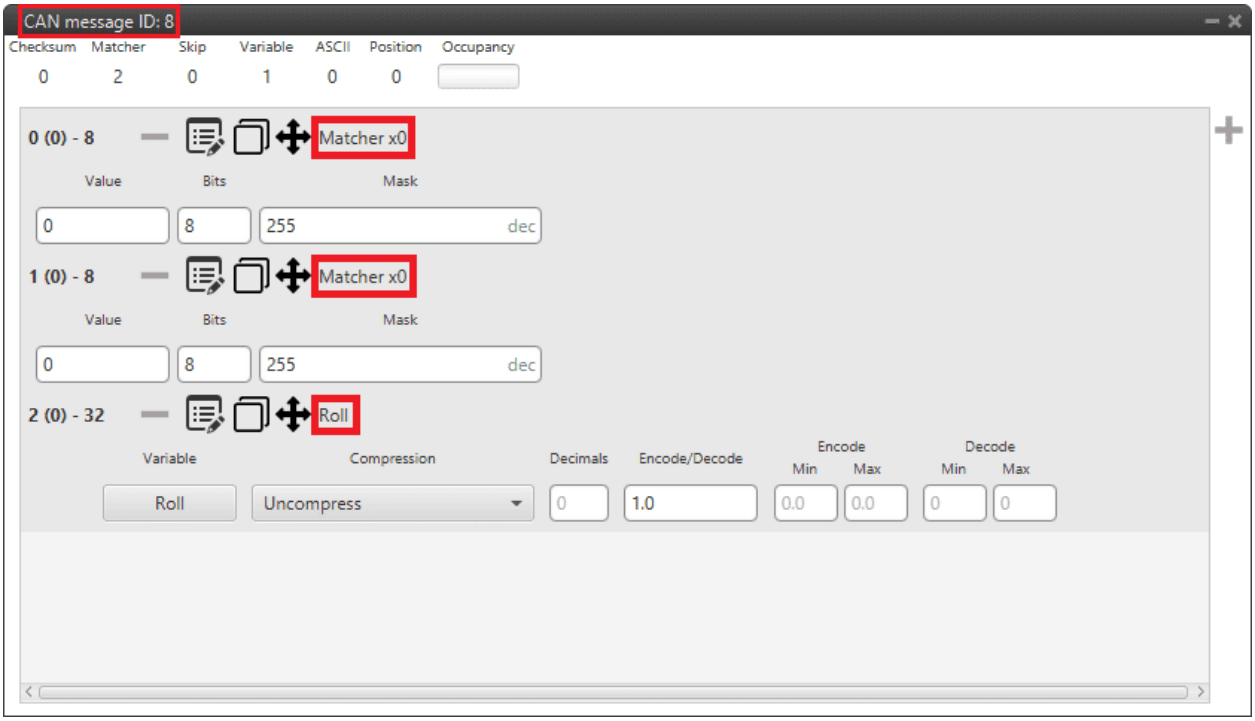


Fig. 140: **Autopilots 1x configuration - Roll message**

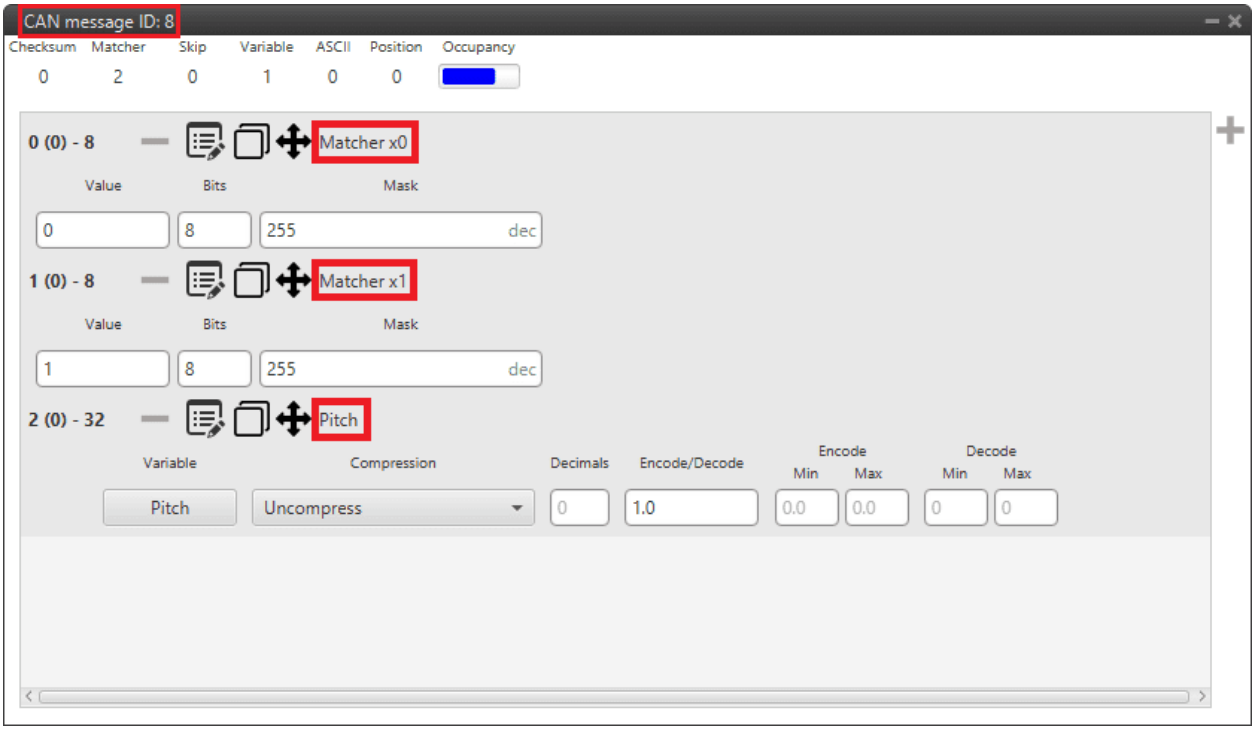


Fig. 141: **Autopilots 1x configuration - Pitch message**

Note: The "**Position not fixed**" variable, which is of type **bit**, has had to be "converted" as follows

to a **real variable** (single-precision float), since that is the **type of variable expected by the arbiter**.

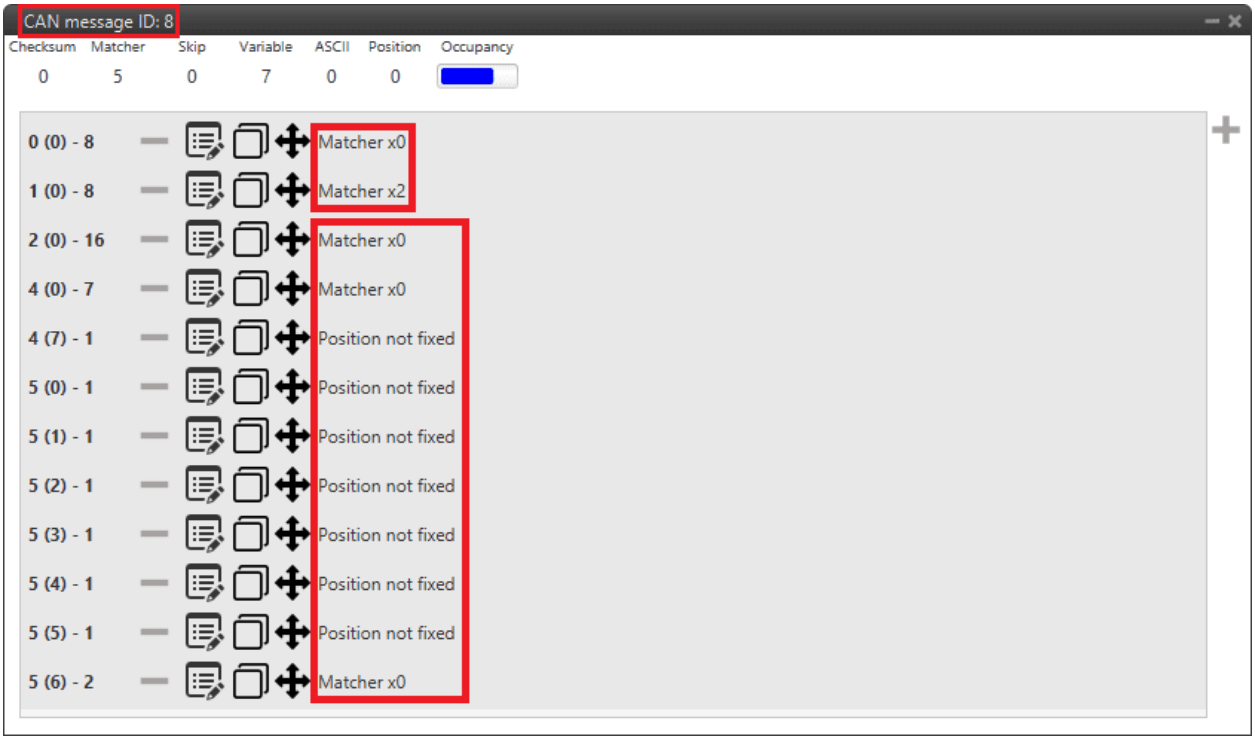


Fig. 142: **Autopilots 1x configuration - Position not fixed message**

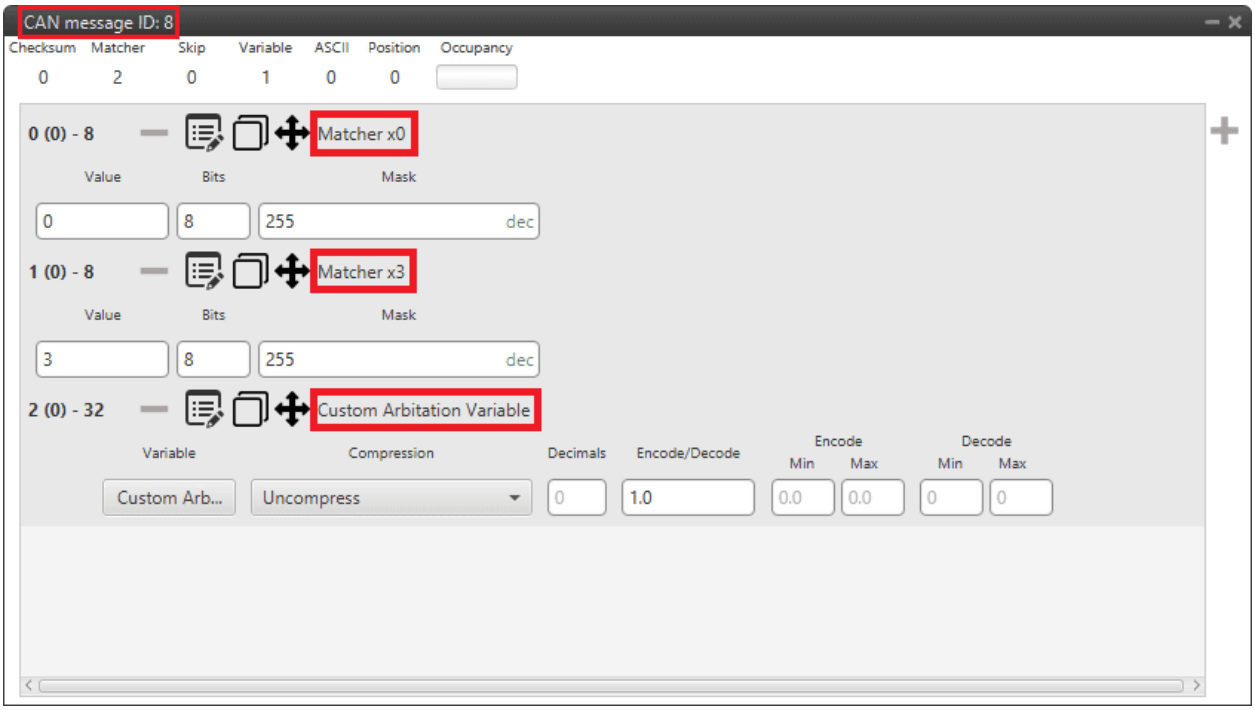


Fig. 143: **Autopilots 1x configuration - Custom Arbitration Variable message**

For more information on configuring CAN custom messages, refer to the *Custom Messages types - Input/Output* section of this manual.

10. Go to Input/Output menu → CAN Setup panel → **Custom Message 0 tab** (since *Custom Message 0* has been connected to the input filter) → **TX** to configure **the messages to be read from the arbiter**.

- Add **4 messages** and configure them as follows:

ID	Can id	Endianness	Time out	Bit ID
00	255	Little endian	1.0 s	0
01	255	Little endian	4.0 s	1
02	255	Little endian	4.0 s	2
03	255	Little endian	4.0 s	3

More information on CAN messages configuration can be found in the *RX Messages (Custom Messages) - Input/Output* section of this manual.

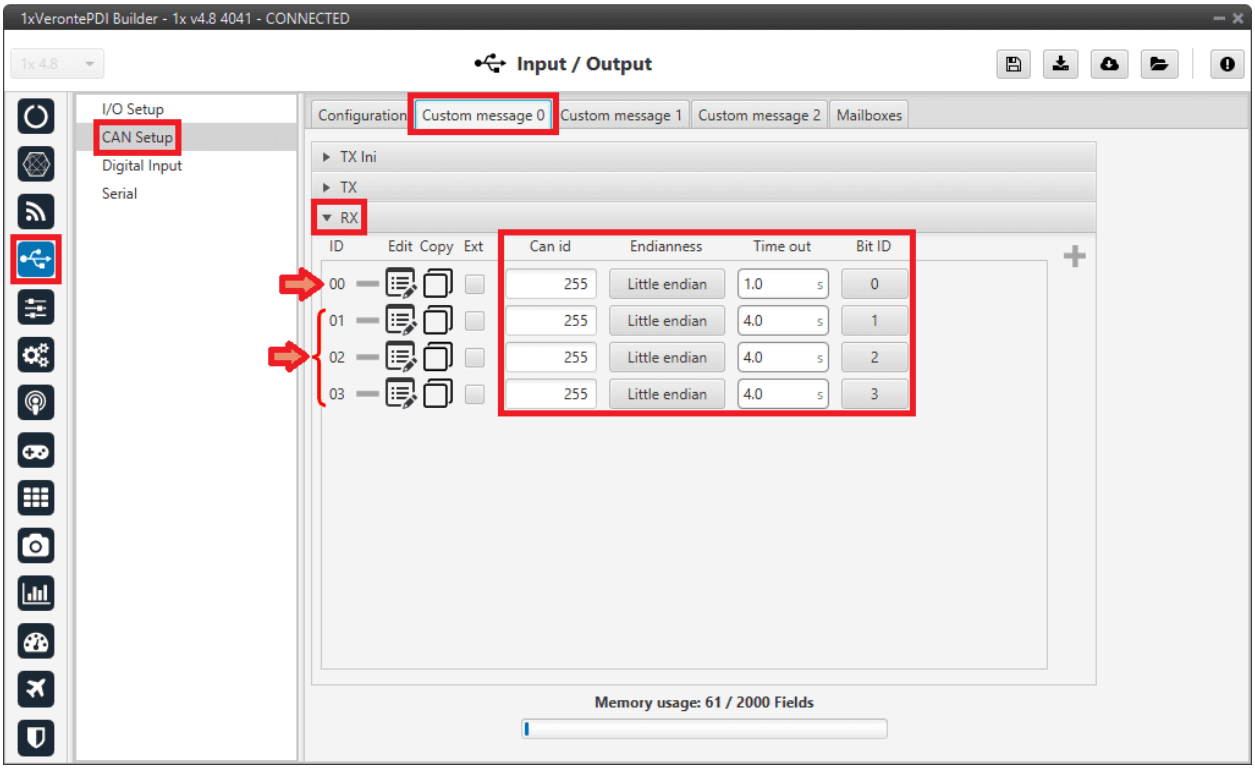


Fig. 144: **Autopilots 1x configuration - Custom messages 0 (RX) configuration**

- Then, to create each message, click on its corresponding  button:
  - **ID 00 ⇒ Status message**. It must be built following the Status Message - CAN Bus protocol section of the **4x Software Manual**.

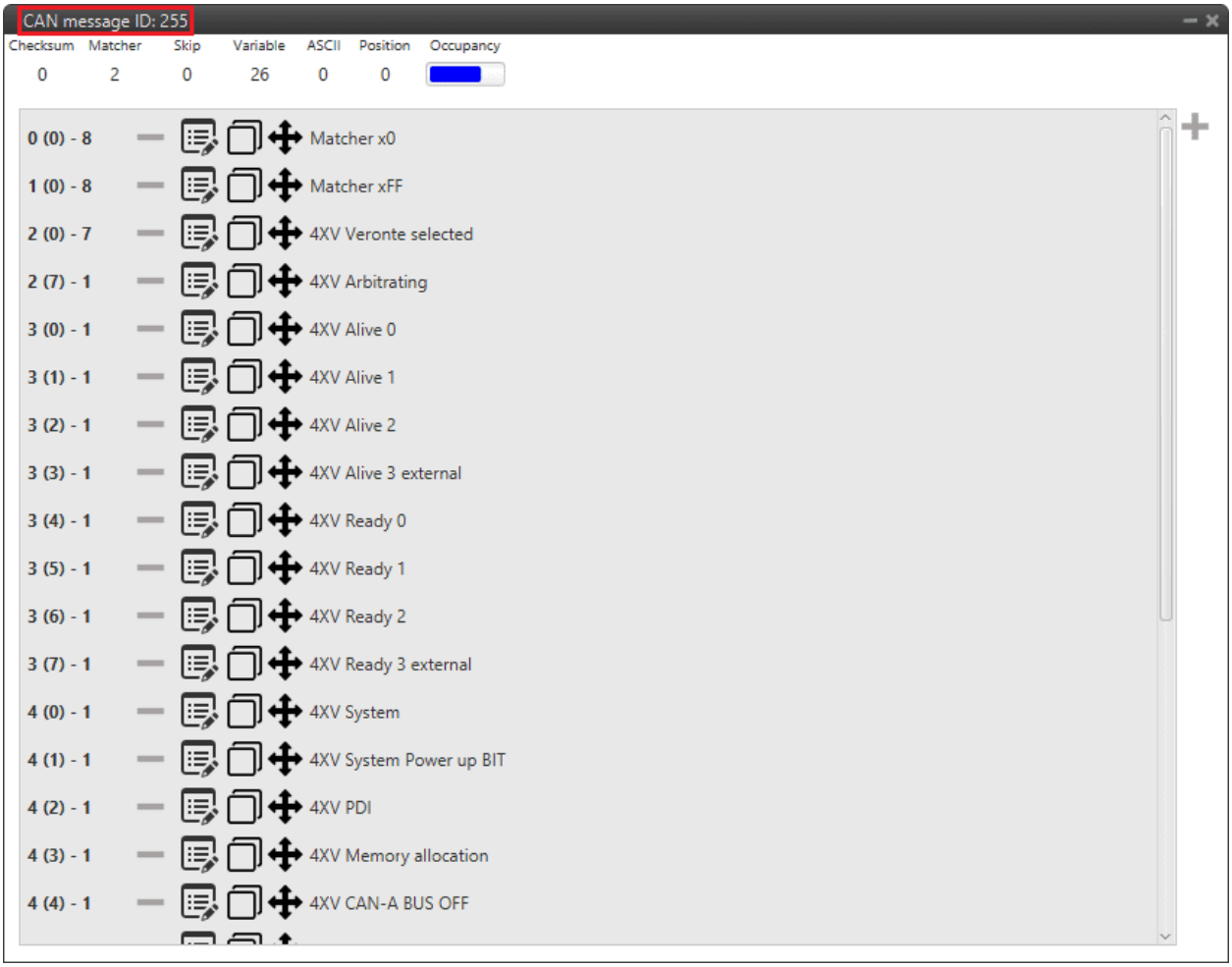


Fig. 145: **Autopilots 1x configuration - Status message**

– **ID 01-03** ⇒ **Scores messages**. They must be built following the Score Message - CAN Bus protocol section of the **4x Software Manual**.

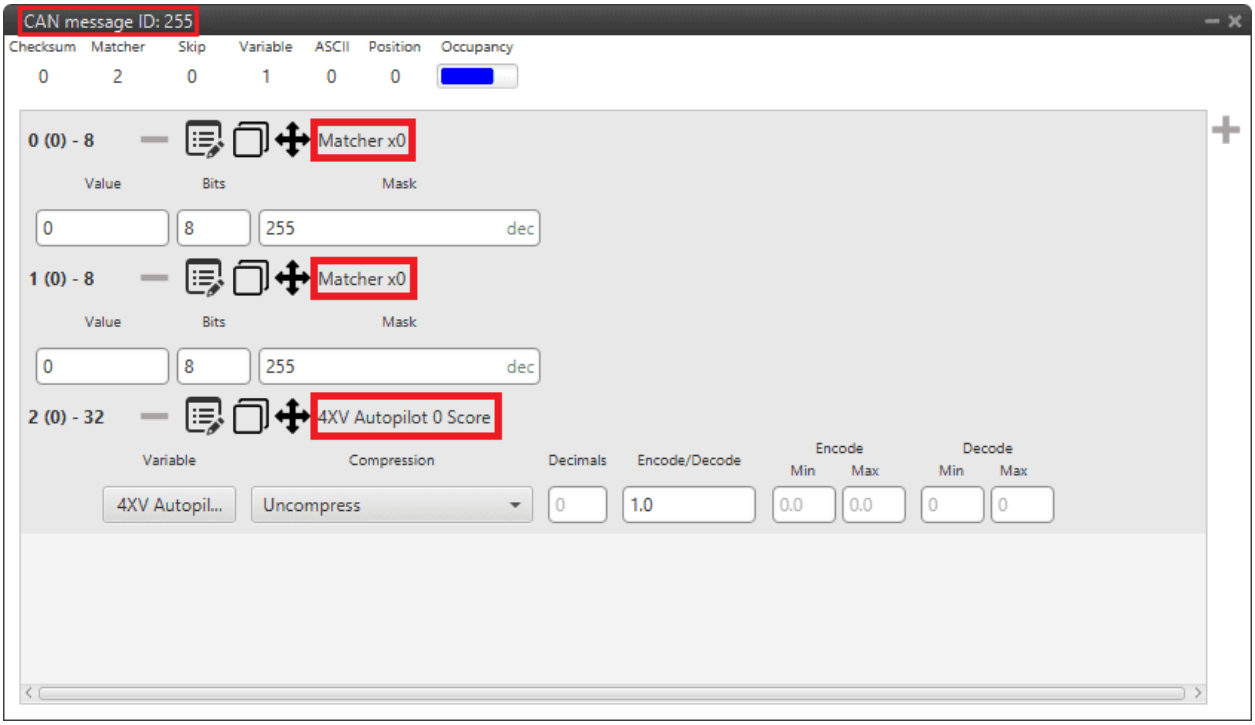


Fig. 146: **Autopilots 1x configuration - AP 0 Score message**

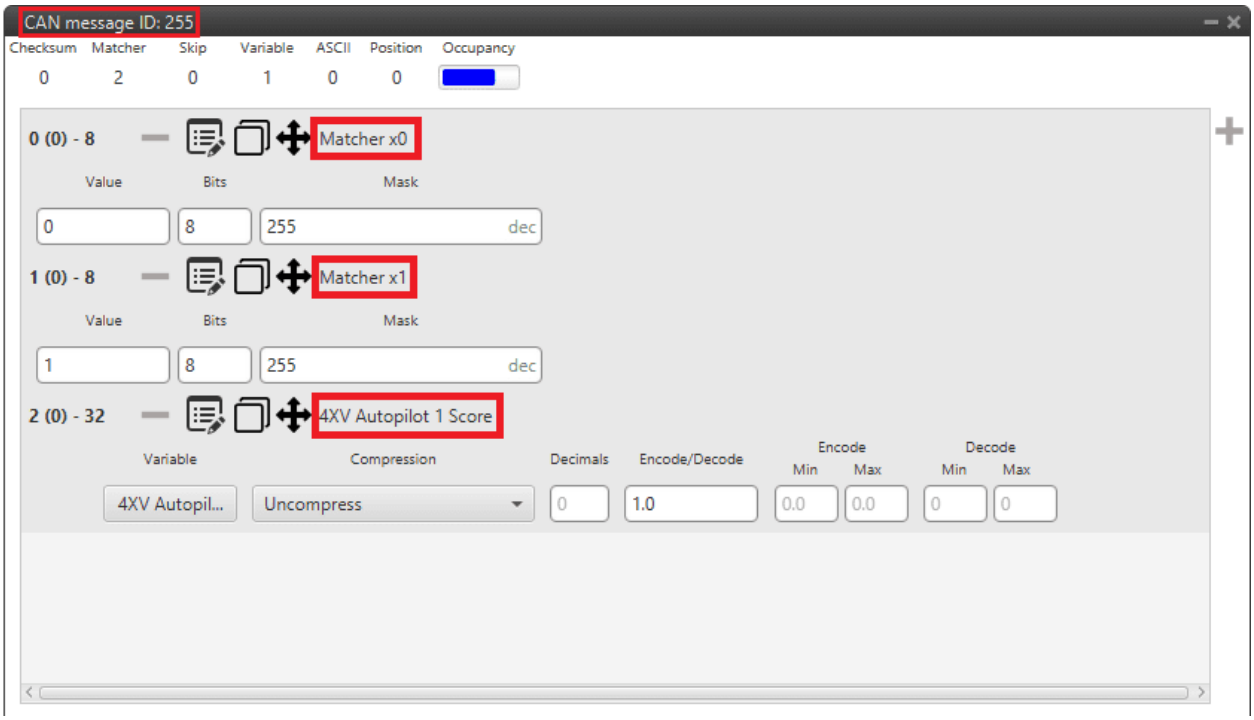


Fig. 147: **Autopilots 1x configuration - AP 1 Score message**

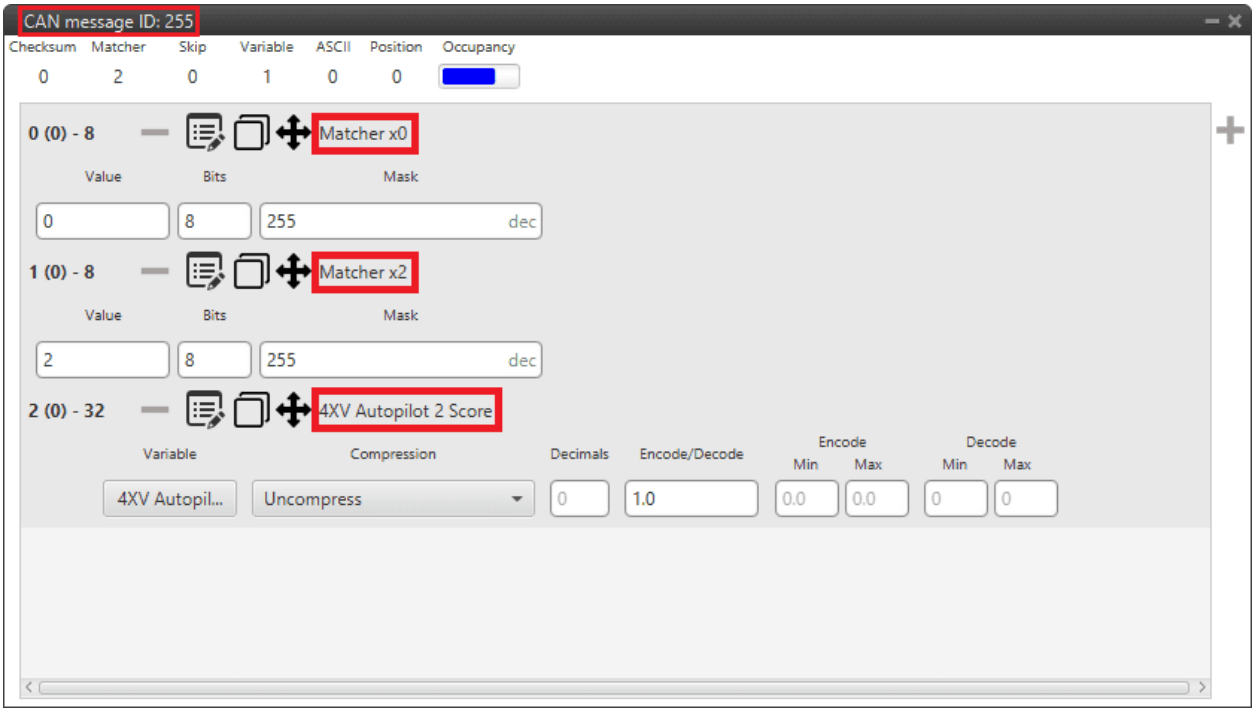


Fig. 148: **Autopilots 1x configuration - AP 2 Score message**

For more information on configuring CAN custom messages, refer to the *Custom Messages types - Input/Output* section of this manual.

11. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

    Configure at least **4** mailboxes in **CAN A** bus with the same configuration as the Input filter to correctly receive the messages from the arbiter:

    ⇒ **ID: 255 DEC**

    ⇒ **Mask: 111 1111 1111 BIN**

> **Warning:** Remember that it is necessary to have **at least 1 free mailbox for TX messages**.

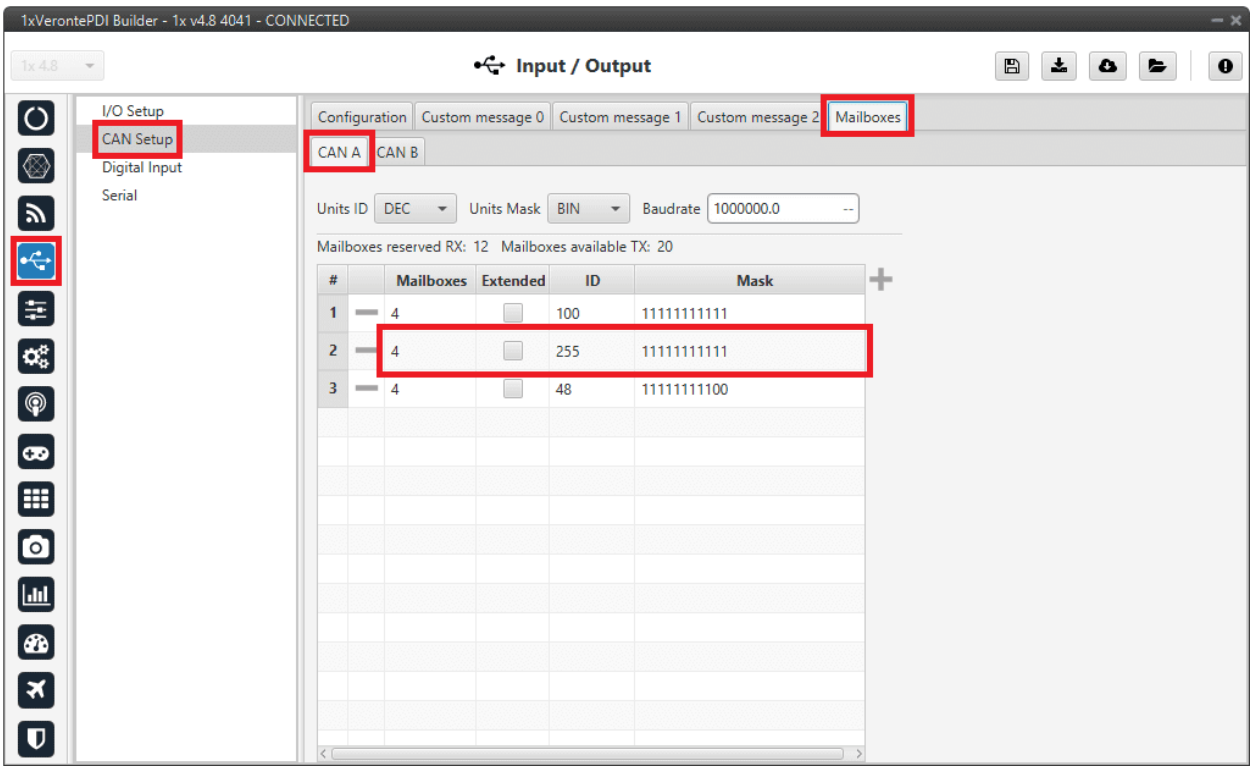


Fig. 149: **Autopilots 1x configuration - Mailboxes configuration**

Important: After making this configuration for **AP 0**, users will have to upload the same configuration on **AP 1** and **AP 2** and **change the IDs of the CAN messages** configured for sending messages (TX) to the **arbiter**.

For **AP 1** it will be necessary to change the Id 8 to **9**, and for **AP 2**, to change it to **10**.

It is also advisable to slightly modify the name of the configuration to be able to distinguish them quickly. This is done from the *Unit name panel*.

Warning: The **Arbiter address** parameter must be set again when uploading the configuration to another AP. This is because this address is unique for each **Autopilot 4x** and it is not exported when downloading the configuration nor applied when uploading a PDI.

3.7.6.1.2 Configuration for external radio communication through RS232

The following is the configuration necessary to establish communication with an external radio connected to one of the multiplexed ports of **Veronte Autopilot 4x**, in this case RS232.

The purpose of this implementation is that the **Autopilot 1x selected by the arbiter** receives the communications from the other two autopilots **via CAN** and redirects them to the **RS232 port** to send them to the radio. As far as reception is concerned, this is not necessary since all 3 **Autopilots 1x** can receive communications over the RS232 port.

In addition, to avoid overloading the CAN bus with telemetry, the telemetry address has been changed to a **dynamic address** of Veronte applications and depending on whether the current autopilot is the selected AP or not, the routing of communications acts in one way or another:

- If it is the **autopilot selected** ⇒ the telemetry is sent by the RS232.
- If it is **not** the **autopilot selected** ⇒ the telemetry is redirected to a port that is not in use to avoid being sent by CAN.

The following schematic broadly summarizes the configuration that will be explained:

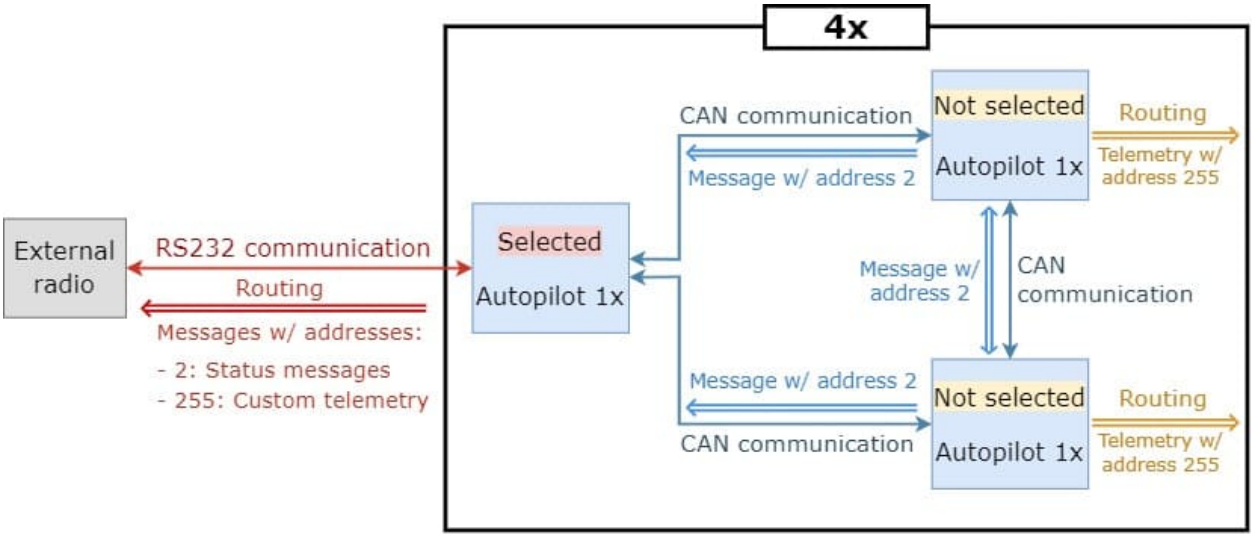


Fig. 150: **Autopilot 4x - External radio communication through RS232 diagram**

Important: The following are the IDs for CAN communications **normally** assigned to each AP, however users may use any IDs they wish:

AP 0	AP 1	AP 2
100	101	102

In this example, **AP 0** is configured, so **Id 100** is associated with it.

The explanation of everything shown in the diagram above has been divided according to the different parts to configure:

- *Telemetry configuration*
- *I/O ports configuration*
- *CAN communication configuration*

- *Routing configuration*

3.7.6.1.2.1 Telemetry configuration

First, configure the custom telemetry to be sent via radio by adding a **Data vector** with the desired variables to a **dynamic address** of **Veronte applications**. To do this:

1. Go to Telemetry menu → **Telemetry panel**.
   - Click on ➕ to add a new data vector, which will be added by default with address 2 (*App 2 ⇒ Data to VApp*).
   - Then configure it as follows:
     - **Frequency**: 15 Hz are recommended.
     - **Address**: Enter a **dynamic address** of Veronte applications, which are in the range **255-599**. In this example **255** is used. For more information on the available addresses, see List of addresses section of the **1x Software Manual**.

     Note: Users only need to enter the address, then the software will automatically recognize that it is a dynamic address of Veronte applications and replace it with "App dynamic *address entered*", in this case "App dynamic 255".
   - Add as much custom telemetry as needed.

For more information on configuring data vectors, see the *Data vectors - Telemetry* section of the present manual.

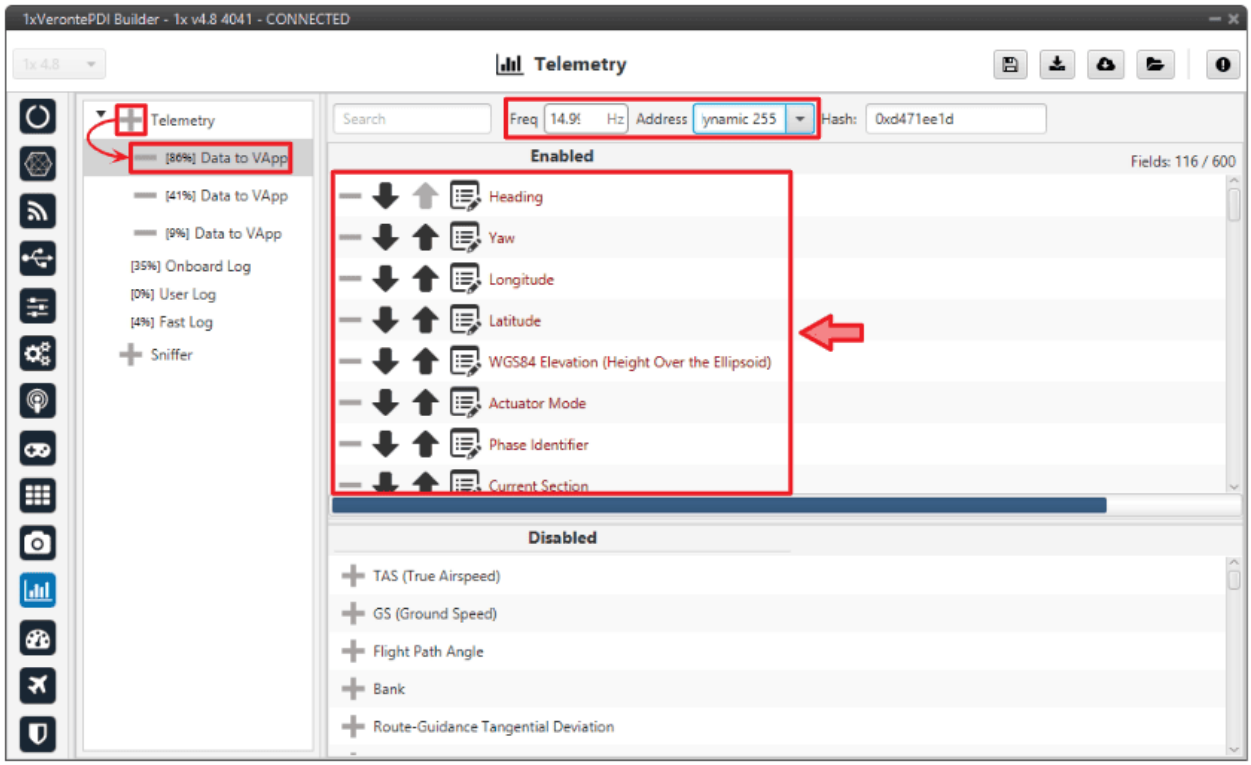


Fig. 151: **External radio communication through RS232 - Telemetry configuration**

3.7.6.1.2.2 I/O ports configuration

Then, it is necessary to perform the ports connections for the sending and reception of CAN communications between **Autopilots 1x**.

In addition, it is recommended to use a user bit, which is the negation of the "*4X Selected*" bit (usually renamed to "*4x not selected*"), to disable the sending of these communications in the case where the current AP is the one selected by the arbiter. This is done to reduce the CAN bus traffic load.

For this purpose:

2. Go to UI menu → Variables panel → **Bits tab**.

   Rename a user bit that is not being used for another purpose in the configuration to be the "*4x not selected*" bit.

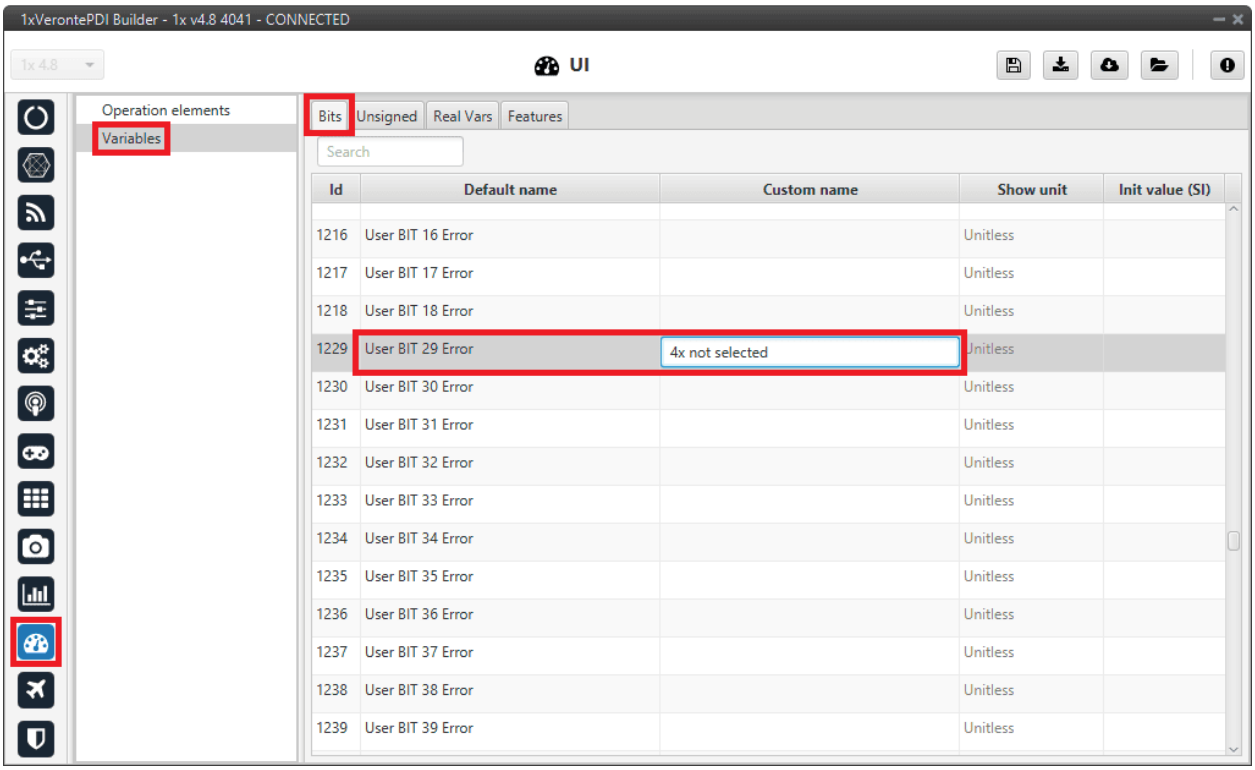


Fig. 152: **External radio communication through RS232 - User bit renamed**

3. Go to **Block Programs menu**.

   Create a program to store the negation value of the "*4X Selected*" bit in the previously renamed user bit. To do this:

   3.1. Add a new program by clicking on ✚ and rename it as desired, in this example *AP Communication*.

   3.2. Configure it as a continuously running program (it should appear as ϟ).

   3.3. Add the blocks "**NOT**", "**Read Bit**" and "**Write Bit**" to the created program.

   3.4. Connect to the logic block "NOT" the "**Read Bit**" block as **input** and the "**Write Bit**" as **output**.

   3.5. Select the variable **4X Selected** for the "Read Bit".

   3.6. Select for the "Write Bit" the user bit renamed as **4x not selected**.

For more information on blocks, read *Block Programs* section of the present manual.

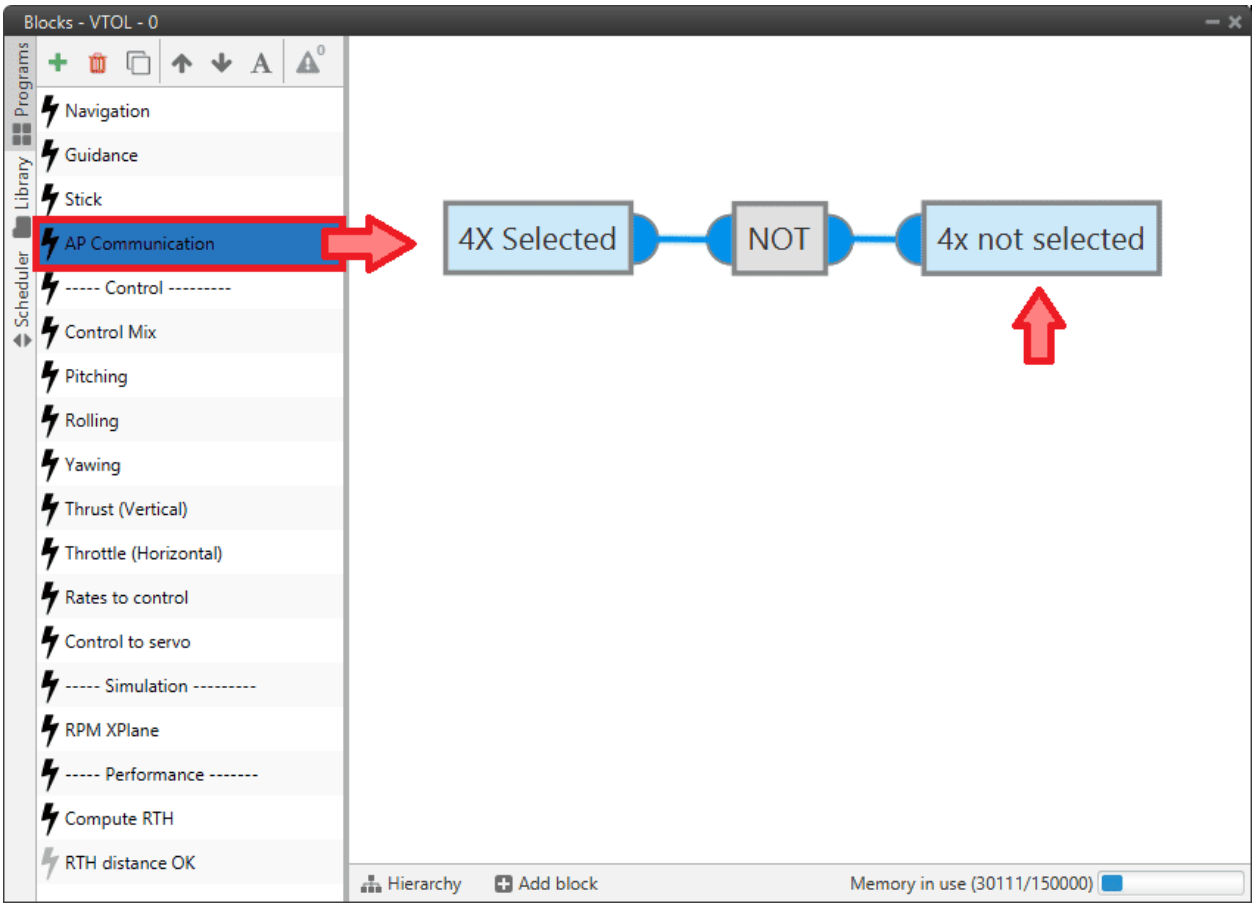


Fig. 153: **External radio communication through RS232 - User bit configuration**

4. Go to Input/Output menu → **I/O Setup panel**.

   Several connections have to be made here for the different communications:

   - For the communicaction with the **radio**, a connection through RS232 port should be carry out:

     Bidirectionally connect the **RS232** producer to a **Commgr port** consumer, in this case *Commgr port 3* is selected. Then, the equivalent **Commgr port** producer (in this case *Commgr port 3* producer) should automatically connect to the **RS232** consumer.

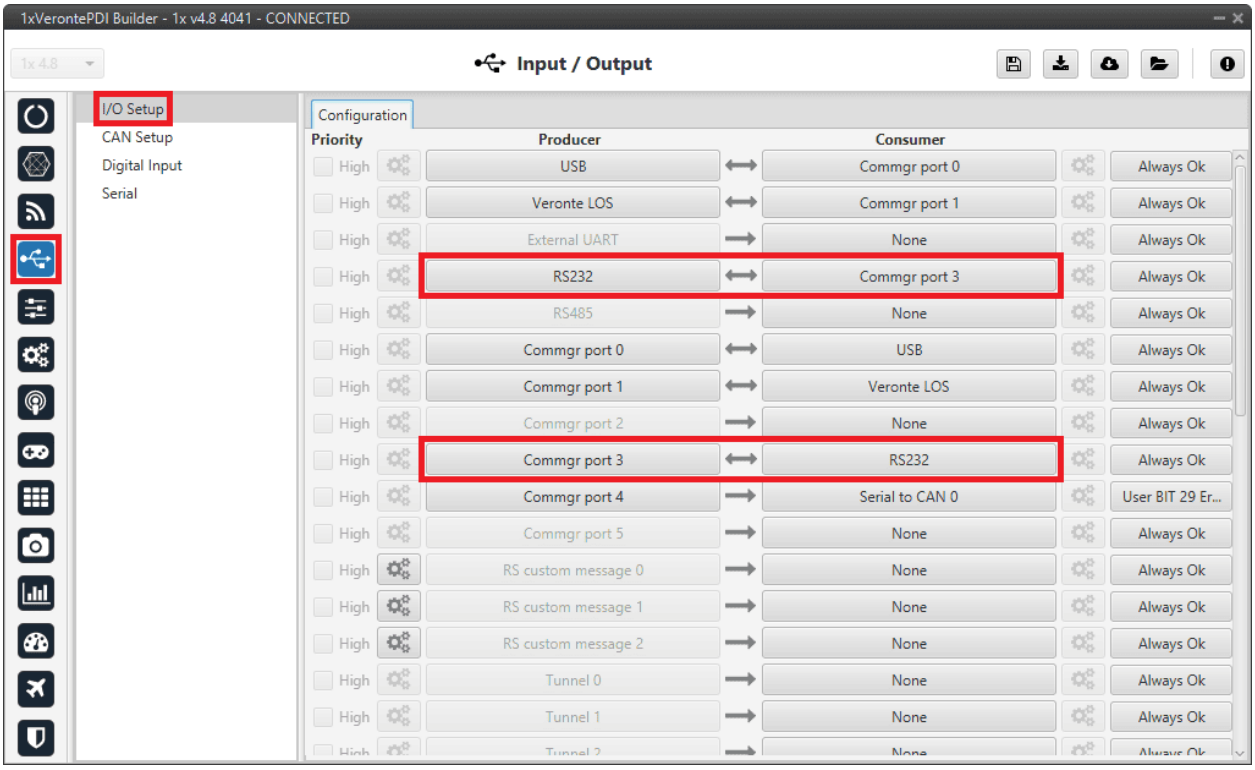


Fig. 154: **External radio communication through RS232 - I/O Setup configuration: Radio**

- For the **transmission** of CAN communications from the current **Autopilot 1x** to the other **Autopilots 1x**, the following operation must be performed:

  Connect a **Commgr port** producer to a **Serial to CAN** consumer, in this case *Commgr port 4* and *Serial to CAN 0* have been selected. In addition, to **disable** the transmission in case this is the selected AP, the user bit previously configured in Block programs should be assigned to the connection (in this case, **User BIT 29**).

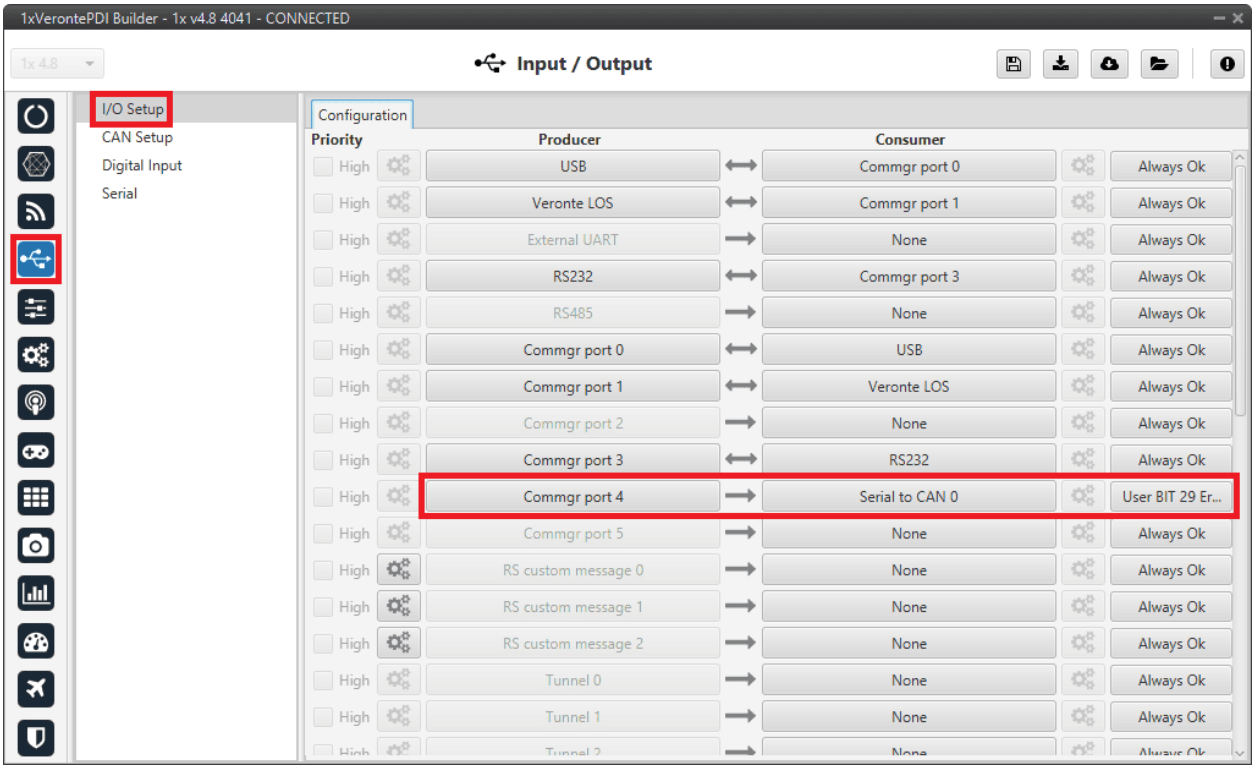


Fig. 155: **External radio communication through RS232 - I/O Setup configuration: Transmission**

- For **reception** of CAN communications from the other **Autopilots 1x** on the current AP:

  Connect two **CAN to serial** producers to two **Commgr port** consumers. In this case, *CAN to serial 0* and *CAN to serial 1* have been connected to *Commgr port 4* and *Commgr port 5* respectively.

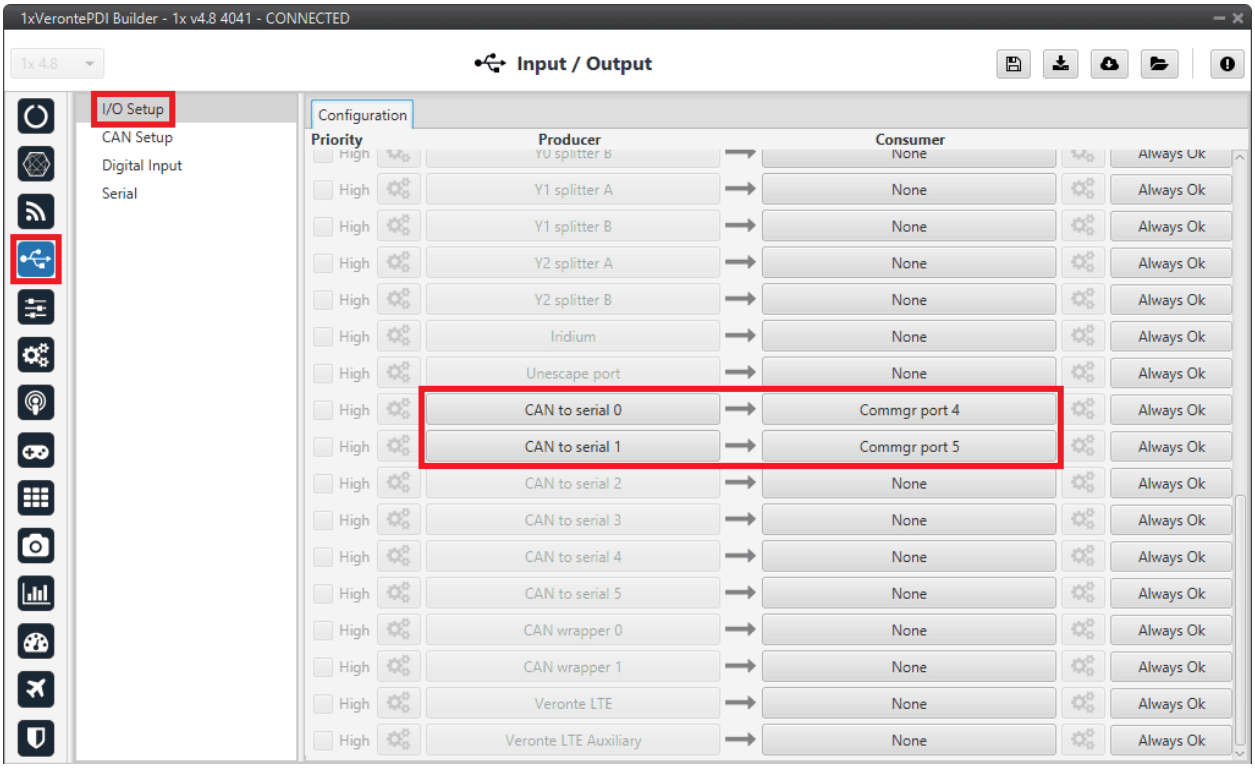


Fig. 156: **External radio communication through RS232 - I/O Setup configuration: Reception**

3.7.6.1.2.3 CAN communication configuration

Next, configure the sending of the CAN communication from the current AP to the other two **Autopilots 1x** as well as the reception of their communication.

In addition, it is also recommended to use the user bit "*4x not selected*" to **disable** sending CAN communications when the current AP is also the selected AP.

To do this:

5. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   The following settings must be made here for the different communications:

   - For the **transmission** of CAN communications from the current **Autopilot 1x** to the other **Autopilots 1x**:

     Connect a **Serial to CAN** producer to an **Output filter** consumer, in this case *Serial to CAN 0* and *Output filter 0* have been selected.

     Note: The **Serial to CAN producer** index must match the one previously selected as **Serial to CAN consumer** in the **I/O Setup** panel.

     In addition, to disable the transmission in case this is the selected AP, the user bit previously configured in Block programs should be assigned to the connection (in this case, **User BIT 29**).

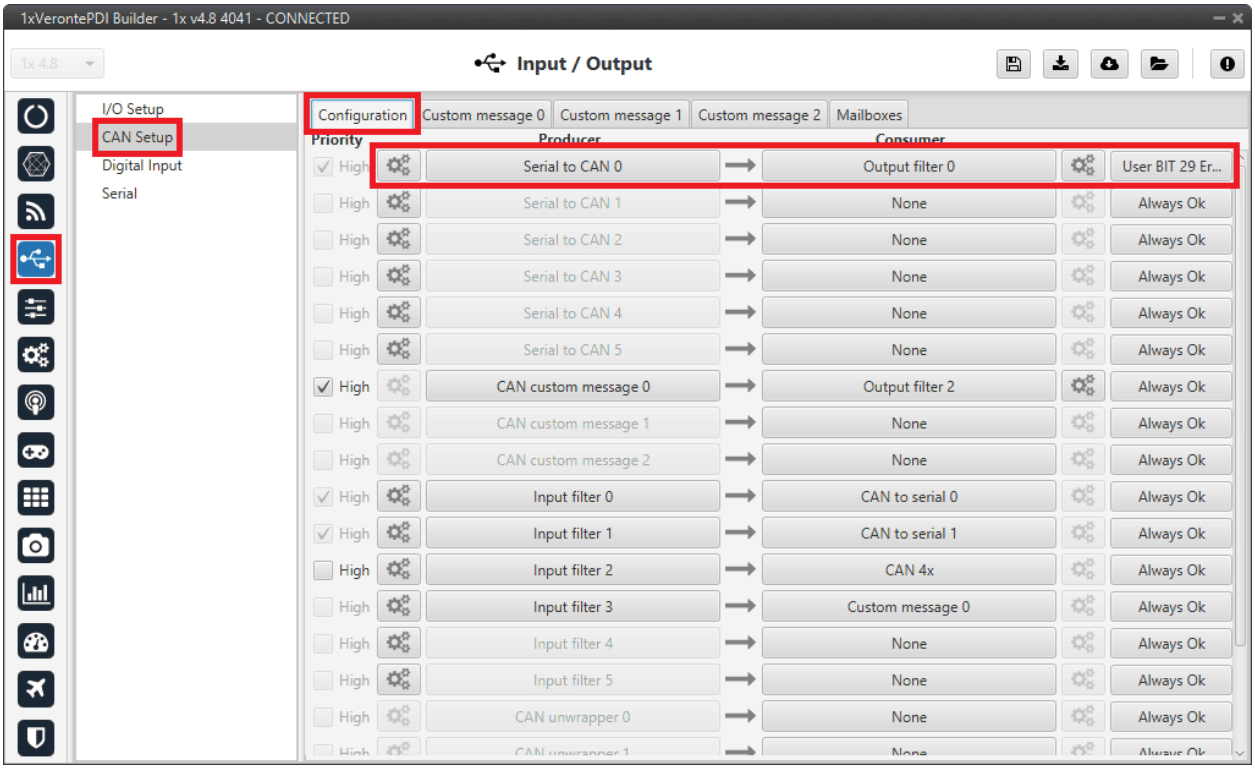


Fig. 157: **External radio communication through RS232 - CAN Setup configuration: Transmission**

– **Serial to CAN** producer should be set to **Id 100** as this is the Id that has been assigned to the current AP.

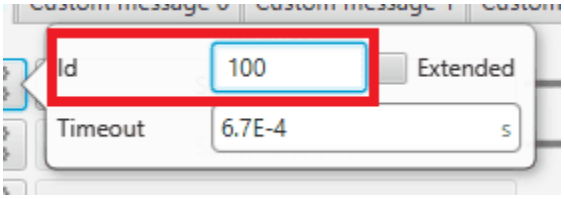


Fig. 158: **External radio communication through RS232 - Serial to CAN configuration**

– **Output filter** is configured to **CAN A** port.

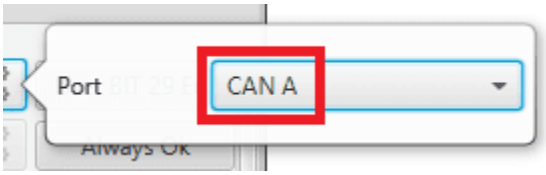


Fig. 159: **External radio communication through RS232 - Output filter configuration**

- For **reception** of CAN communications from the other **Autopilots 1x** on the current AP:

  Connect two **Input filter** producers to two **CAN to serial** consumers. In this case, *Input filter 0* and *Input filter 1* have been connected to *CAN to serial 0* and *CAN to serial 1* respectively.

Note: The **CAN to serial consumers** indices must match those previously selected as **CAN to serial producers** in the **I/O Setup** panel.

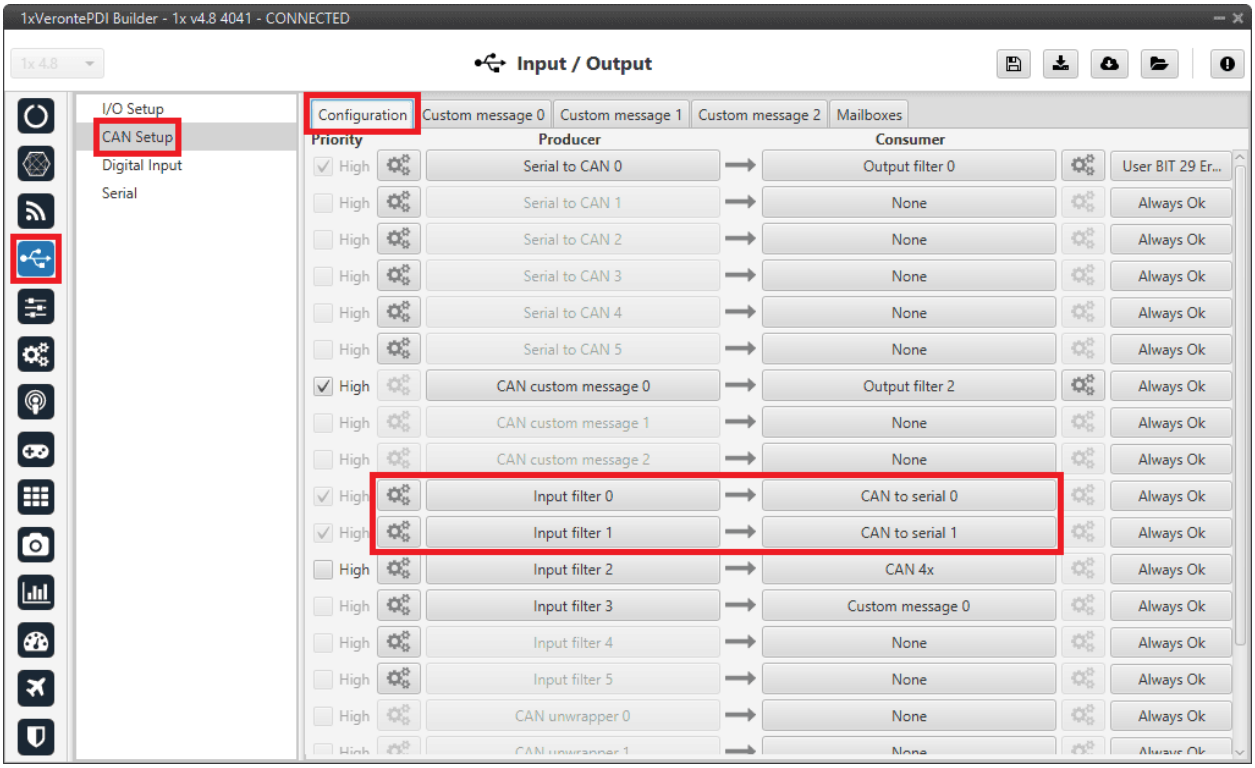


Fig. 160: **External radio communication through RS232 - CAN Setup configuration: Reception**

Input filter producers should be configure with the following parameters in order to correctly receive communications from the other two **Autopilots 1x**:

- Port: **CAN A**
- Id: **101 / 102** ⇒ CAN Ids assigned to **AP 1** and **AP 2** respectively
- Mask: **2047**

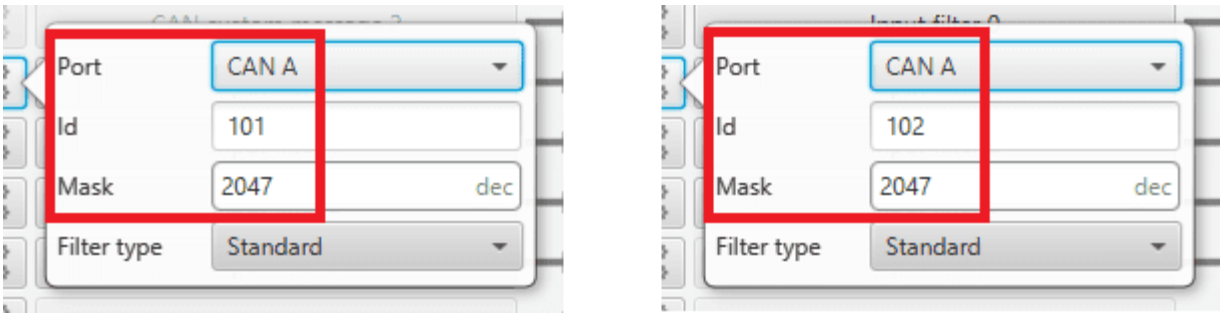


Fig. 161: **External radio communication through RS232 - Input filter 0/1 configuration**

6. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Configure at least **4** mailboxes on **CAN A** bus for receiving messages from each of the other two autopilots. They must have the same configuration as the Input filters to receive them correctly:

   ⇒ **ID: 101 / 102 DEC**

   ⇒ **Mask: 111 1111 1111 BIN**

> **Warning:** Remember that it is necessary to have **at least 1 free mailbox for TX messages**.

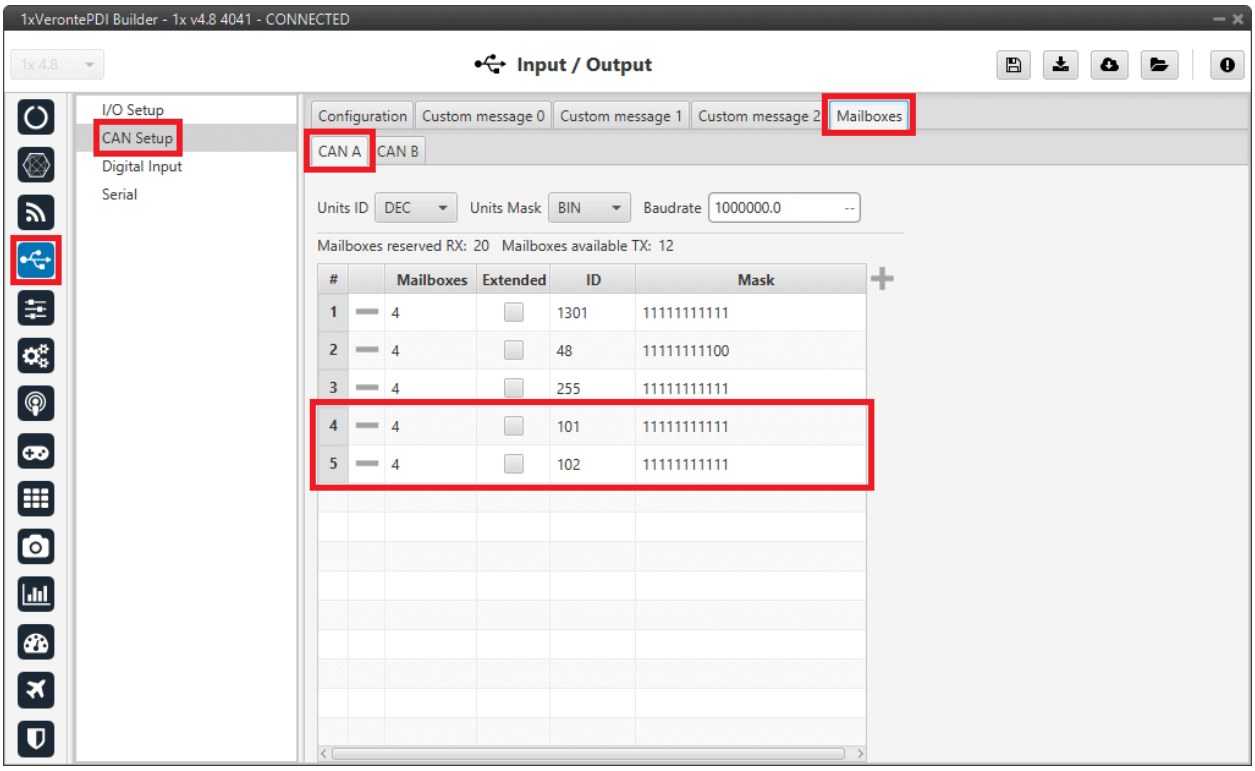


Fig. 162: **External radio communication through RS232 - Mailboxes configuration**

3.7.6.1.2.4 Routing configuration

Finally, the following steps represent the necessary configuration to be performed by users for routing communications.

7. Go to Communications menu → **Ports panel**.

   2 **routing** configurations must be done:

   - **Routing 0**: This will be used when the current AP is also the selected AP. It is configured to output through **PORT 3** (which is connected to **RS232** on the **I/O Setup panel**) the following:
     - All messages with address **2** ⇒ **Status messages** from both the selected AP and the other 2 **Autopilots 1x**.
     - All messages with address **255** ⇒ **Custom telemetry** that has been configured to this dynamic address of Veronte applications.
   - **Routing 1**: This will be used when the current AP is not the selected AP. It is configured to force telemetry with address **255** (custom telemetry) to be redirected to a port that is not in use, in this case **PORT 5**.

   For more information on the configuration of this panel, see the *Ports - Communications* section of the present manual.

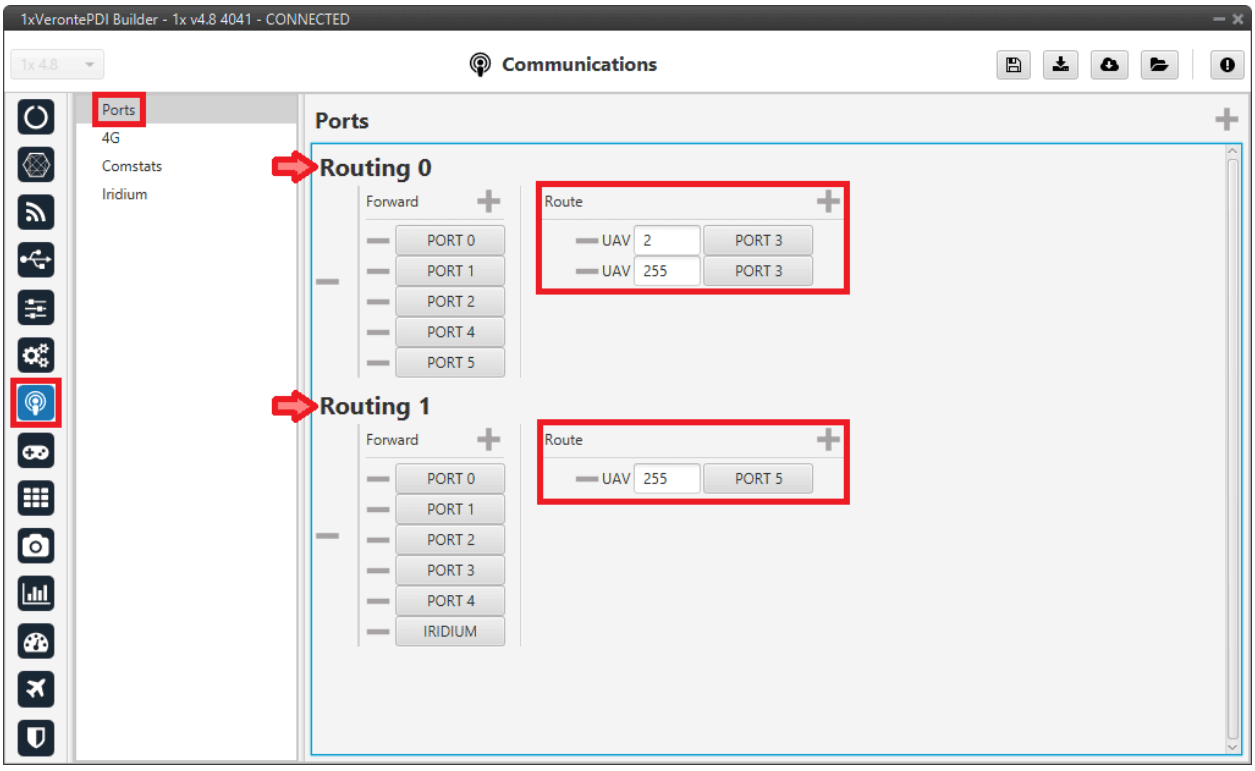


Fig. 163: **External radio communication through RS232 - Routing configuration**

8. Go to **Automations menu**.

   To switch between both routing configurations it is necessary to create 2 automations that will be triggered depending on whether the current autopilot is the selected AP or not (with the value of the **4X Selected** variable).

First, create an automation to select the routing configuration when the **current AP is also the selected AP** ⇒ **Routing 0** configuration. To do this:

A. Click on the **New automation** button and rename it as desired, in this example *4x routing table 0*.

B. Go to **Events** side and configure an **Alarm** event as follows:

- Enter a custom name for this event, *AP Selected* has been entered.
- Select **All ok** in the *Type* option, as it is desired that the action is triggered when the bit is in the "**true**" state.
- Click on ➕ to add the variable that has to activate the alarm ⇒ **4X Selected** bit.

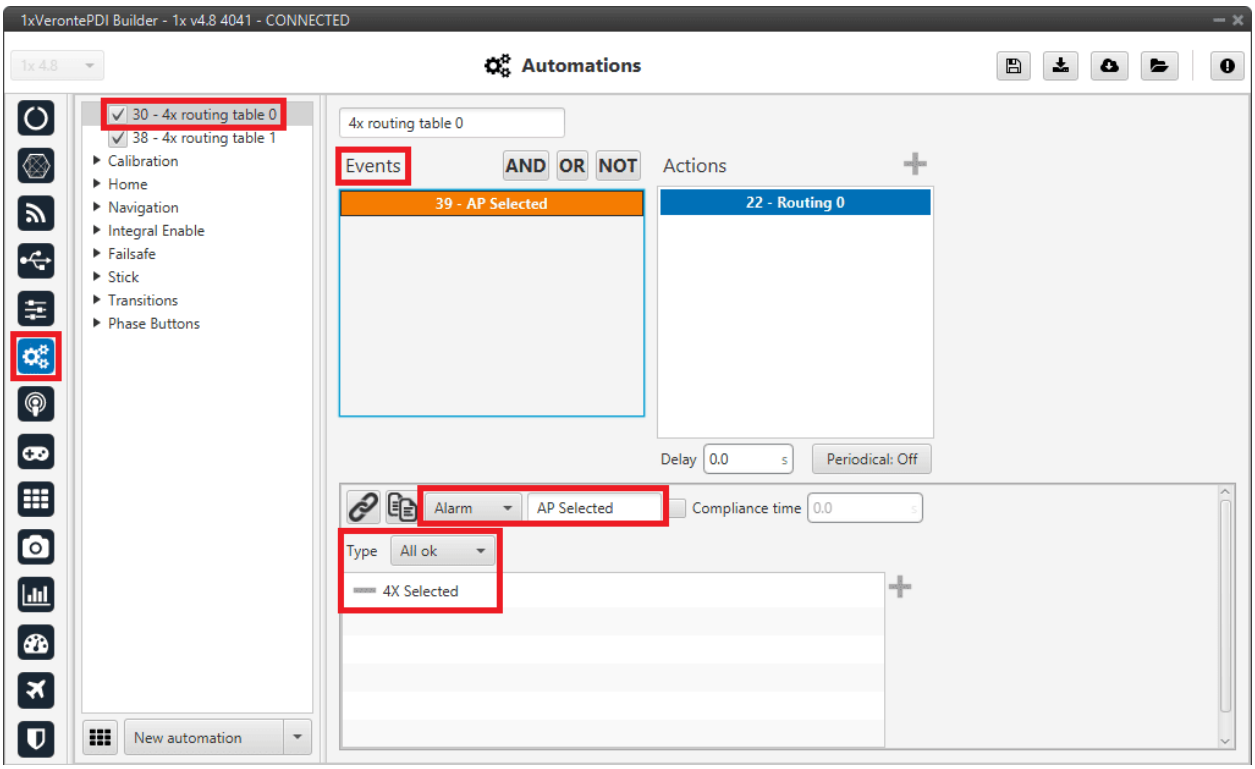


Fig. 164: **External radio communication through RS232 - Routing 0 event**

C. Go to **Actions** side, add a new **Ports** action and configure it as follows:

- Enter a custom name for this action, *Routing 0* has been chosen.
- Select the **Routing 1** option as this is the automation to select the prevously defiend *Routing 0* configuration.

> **Warning:** **Routing 1** option in **automations** corresponds to **Routing 0 in the Ports panel**.

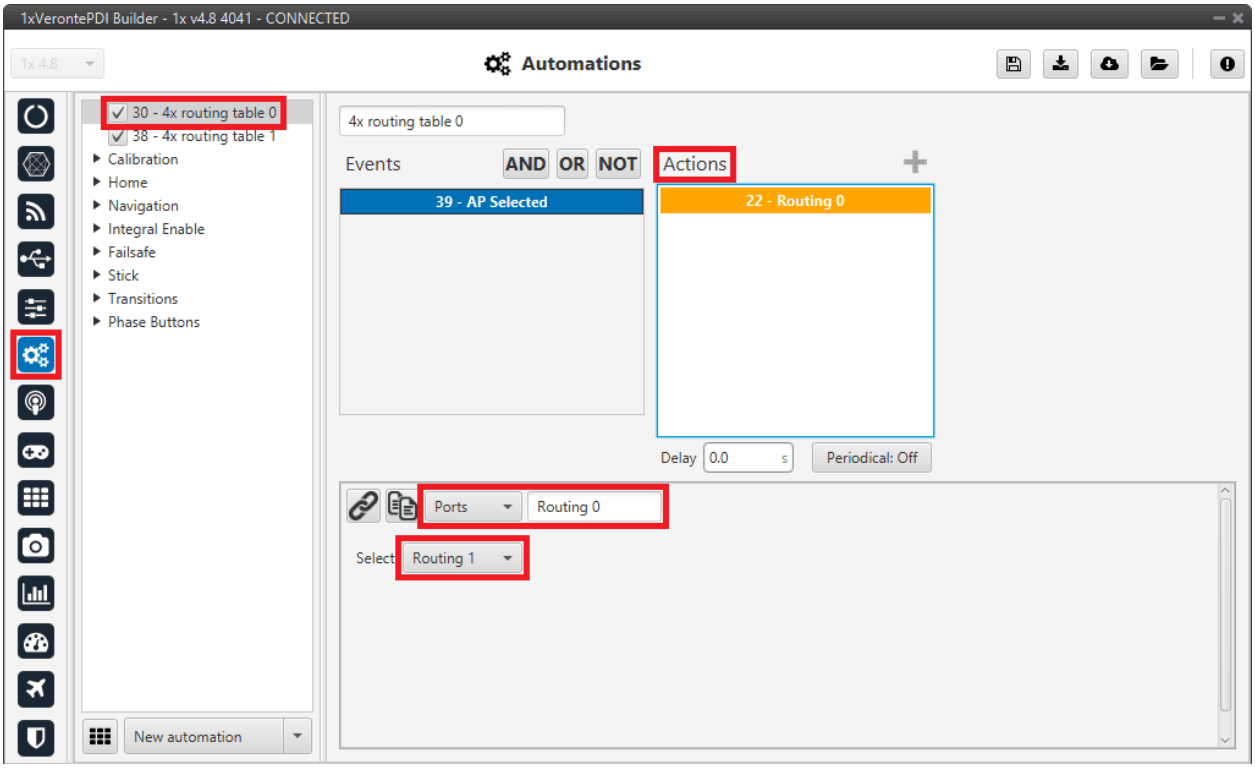


Fig. 165: **External radio communication through RS232 - Routing 0 action**

For more information on the configuration of these automations, see *Automation* section of this manual.

Repeat the same steps (with slightly differences) to create the automation in case **this is not the selected AP** and the **Routing 1** configuration must be used:

D. Click on the **New automation** button and rename it as desired, in this example *4x routing table 1*.

E. Go to **Events** side, select the previously created event "**AP Selected**" and add to it the **NOT** boolean operation. Therefore, the action will be triggered when the **4X Selected** bit is in "**false**" state.

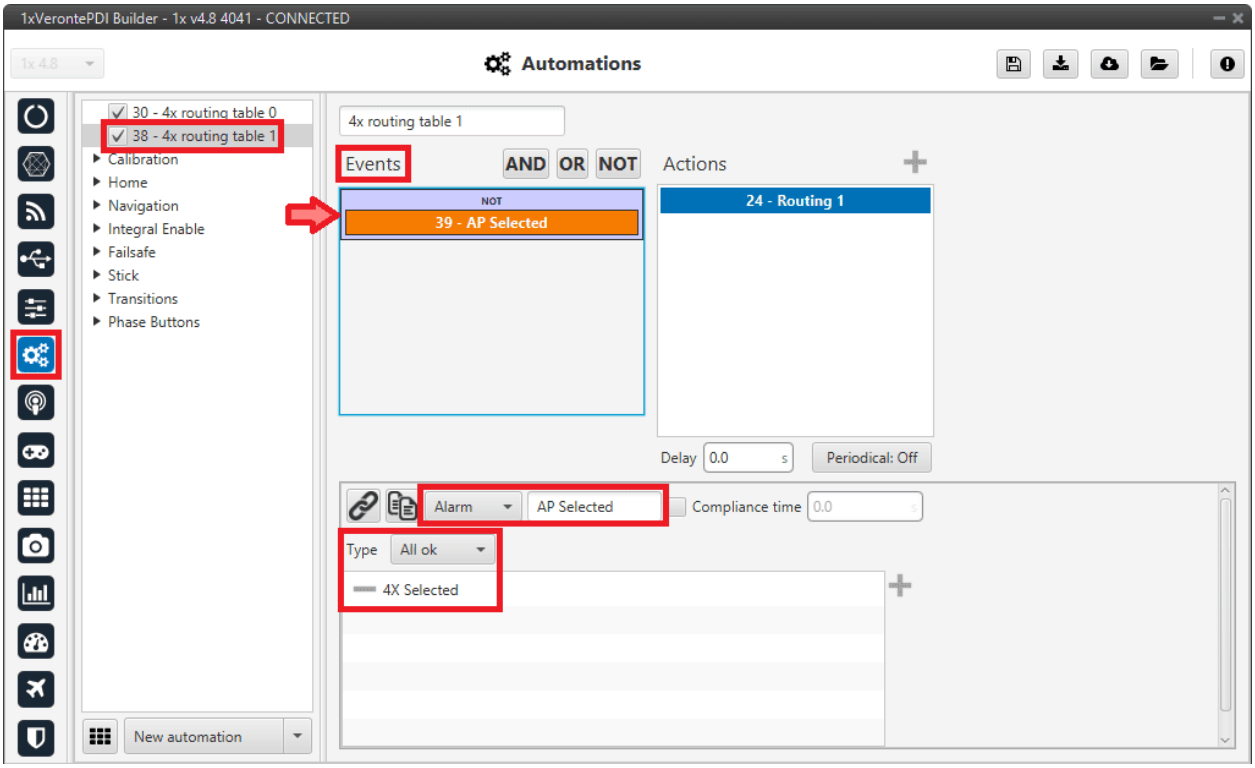


Fig. 166: **External radio communication through RS232 - Routing 1 event**

F. Go to **Actions** side, add a new **Ports** action and configure it as follows:

- Enter a custom name for this action, *Routing 1* has been chosen.
- Select the **Routing 2** option as this is the automation to select the prevously defiend *Routing 1* configuration.

> **Warning:** **Routing 2** option in **automations** corresponds to **Routing 1 in the Ports panel**.

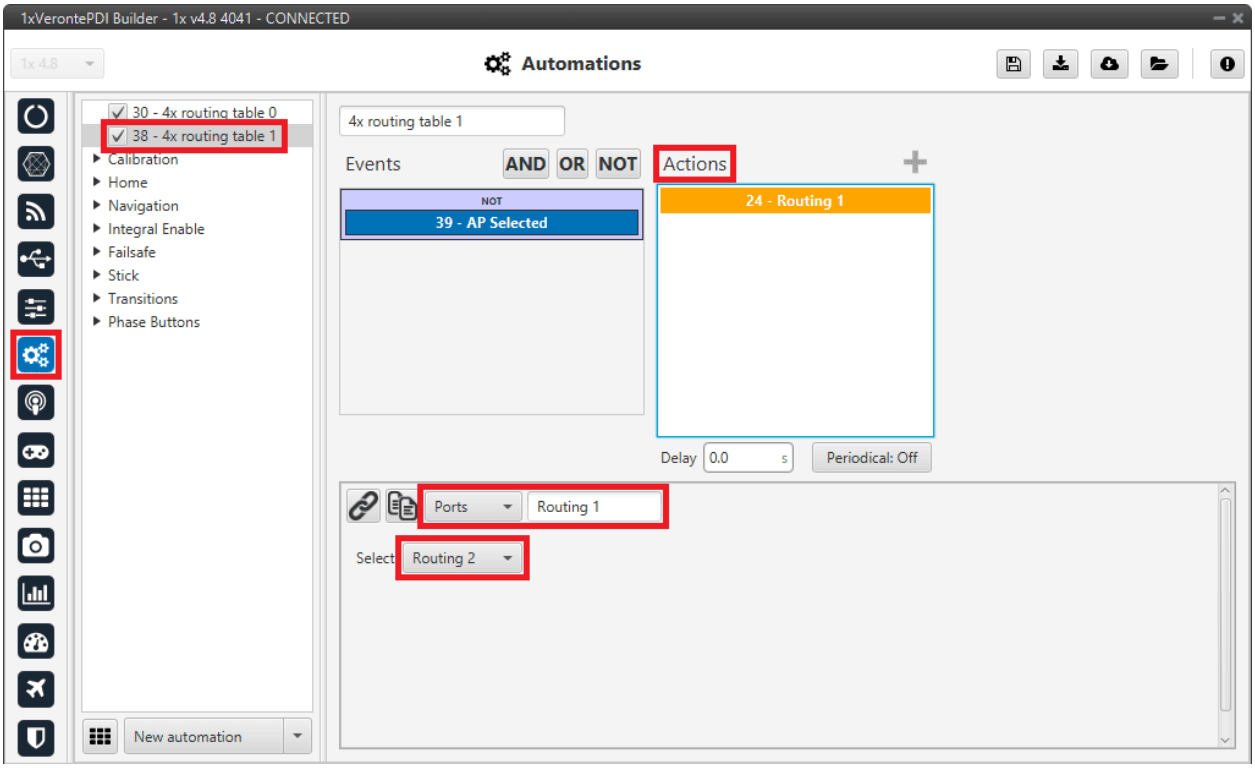


Fig. 167: **External radio communication through RS232 - Routing 1 action**

Important: After making this configuration for **AP 0**, users will have to upload the same configuration on **AP 1** and **AP 2** and **change the CAN communications IDs** configured in **CAN Setup panel**:

IDs	Serial to CAN 0 producer	Input filter 0 producer	Input filter 1 producer	Mailboxes for AP 0 messages	Mailboxes for AP 2/1 messages
AP 1	101	100	102	100	102
AP 2	102	100	101	100	101

It is also advisable to slightly modify the name of the configuration to be able to distinguish them quickly. This is done from the *Unit name panel*.

3.7.6.1.3 Arbiters communication

As it is sometimes not possible to connect the **Arbiters**, which are inside the **Veronte Autopilot 4x**, directly to the PC to configure them (access **4x PDI Builder**), **Veronte Autopilot 1x** is connected to the PC and a connection is made between **Arbiters and Autopilot 1x via CAN**.

Important: **Arbiter B** can only be configured in this way.

To be able to communicate with **Arbiters** via CAN, the following connection is required:

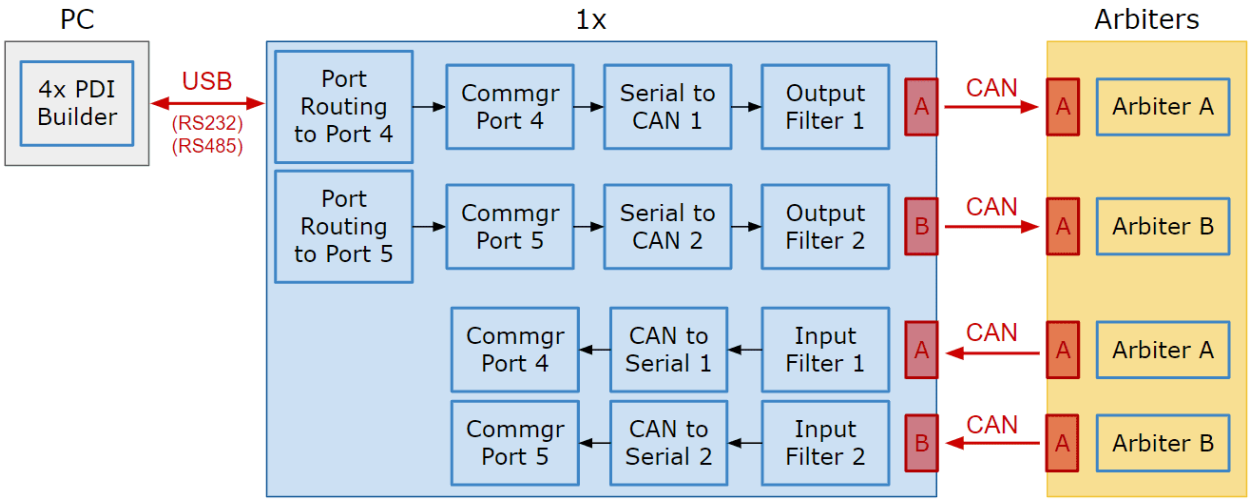


Fig. 168: **Communication diagram PC ↔ Arbiters**

Follow the steps below to make this configuration:

1. Go to Input/Output menu → **I/O Setup panel**.

   Check the connection between the computer and the **Autopilot 1x** (usually via USB, but RS232 and RS485 are also possible).

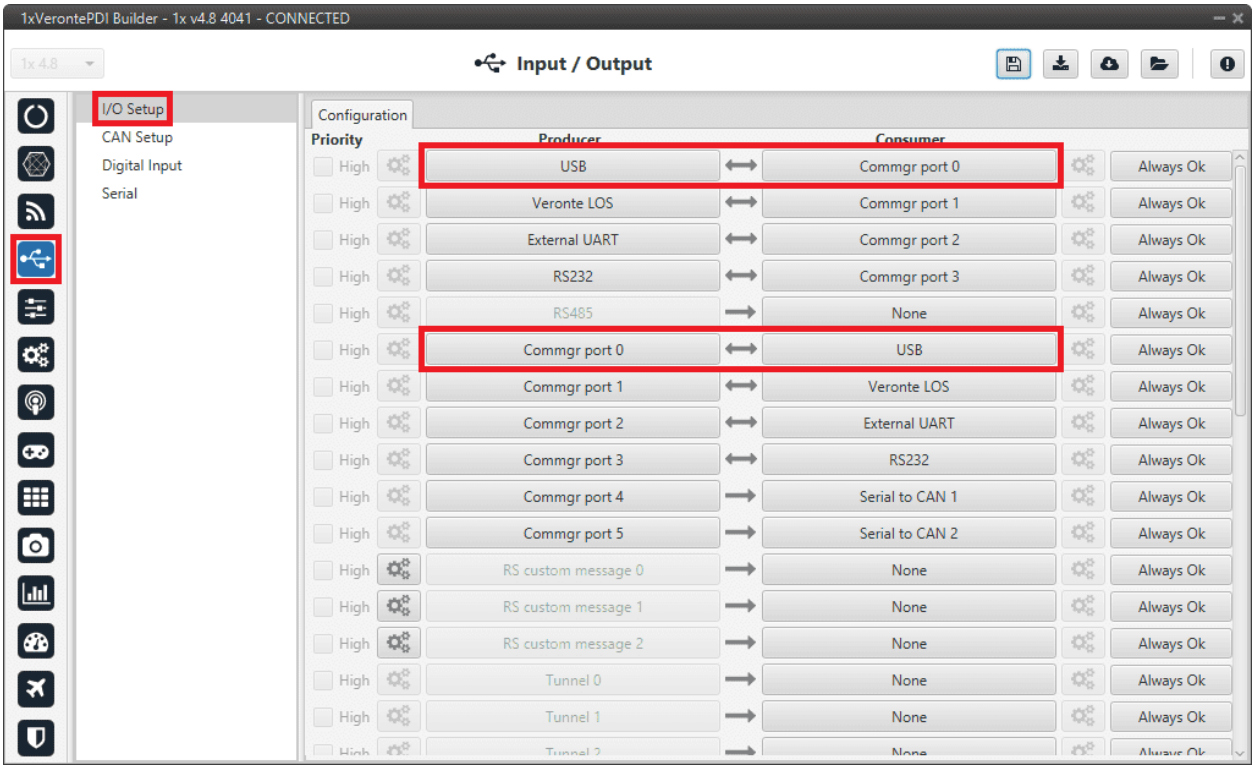


Fig. 169: **Arbiters communication - I/O Setup configuration: PC connection**

2. Go to Communications menu → **Ports panel**.

   Remove Port 4 and 5 from the Forward group and **add Port 4 and 5 to the Route group**, with target Arbiters' Address:

   - Address of Arbiter **A**: **50 000** + Serial number of **4x**
   - Address of Arbiter **B**: **54 000** + Serial number of **4x**

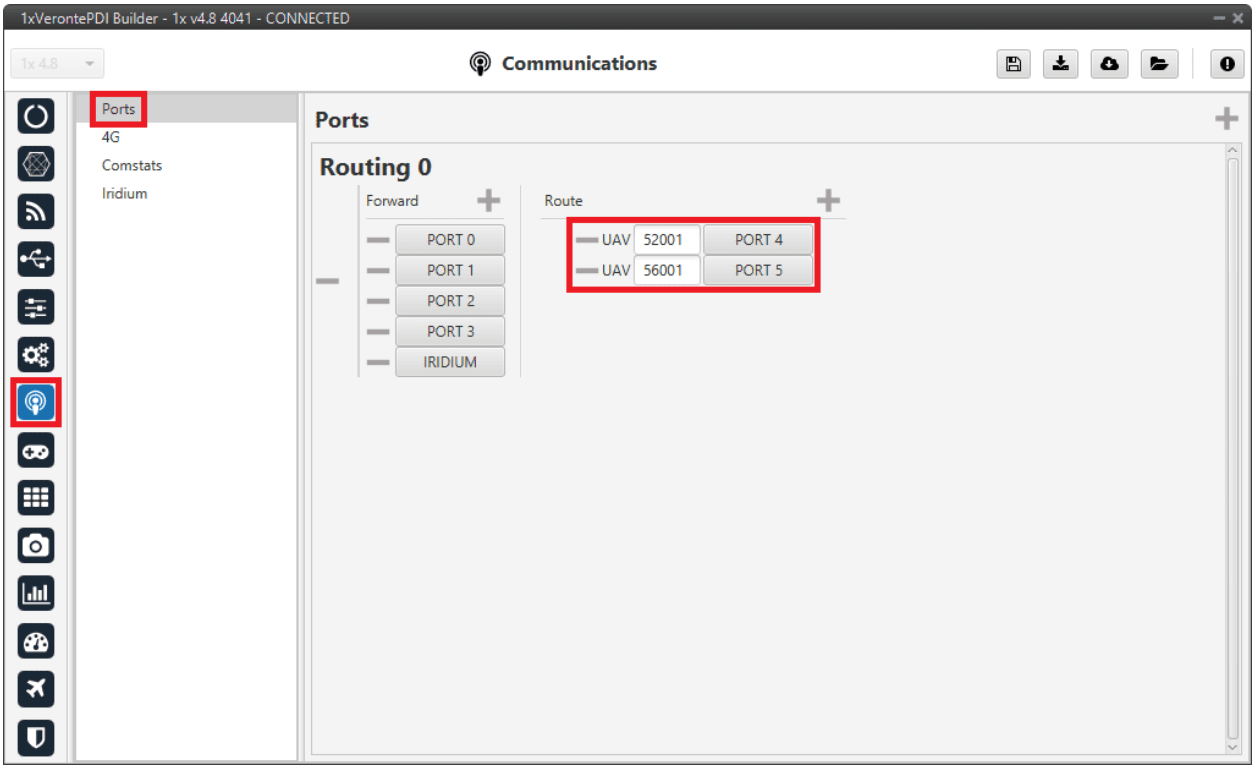


Fig. 170: **Arbiters communication - Routing configuration**

Note: This is just an example, users can choose *Ports* other than 4 and 5.

---

3. Go to Input/Output menu → **I/O Setup panel**.

   Connect **Commgr port 4** to **Serial to CAN 1** consumer and **Commgr port 5** to **Serial to CAN 2** consumer:

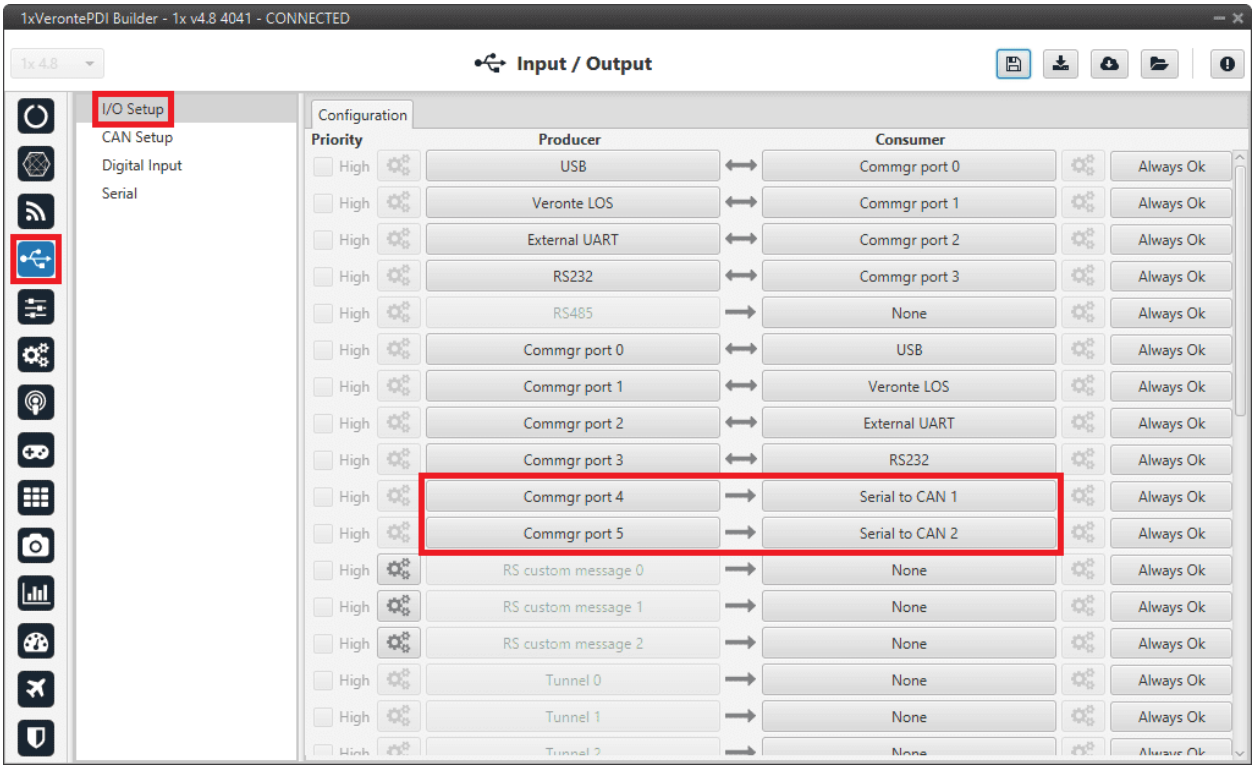


Fig. 171: **Arbiters communication - I/O Setup configuration: Serial to CAN**

Then, connect **CAN to serial 1** to **Commgr port 4** and **CAN to serial 2** to **Commgr port 5**:

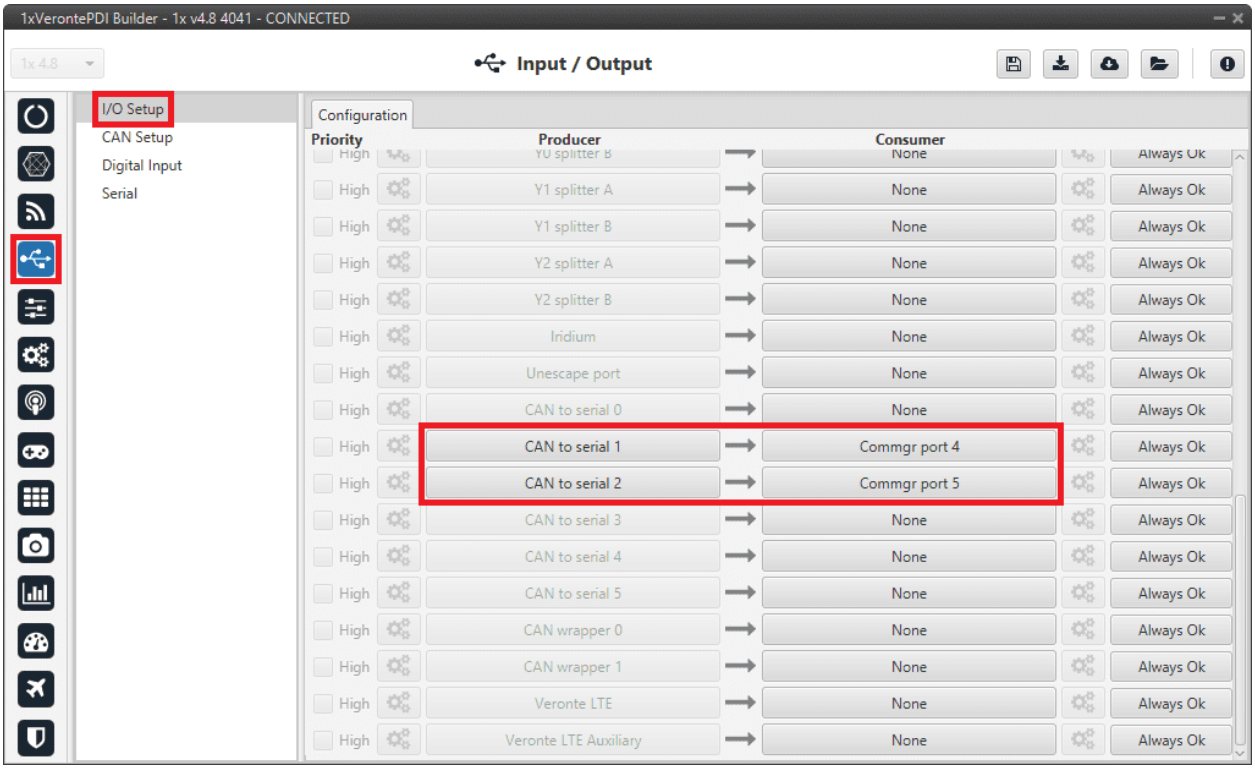


Fig. 172: **Arbiters communication - I/O Setup configuration: CAN to serial**

Note: This is just an example, users can choose *Serial to CAN* and *CAN to serial* other than 1 and 2.

---

4. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.
   - Connect **Serial to CAN 1** to **Output filter 1** and **Serial to CAN 2** to **Output filter 2**.

     In addition, connect **Input filter 1** to **CAN to serial 1** and **Input filter 2** to **CAN to serial 2**:

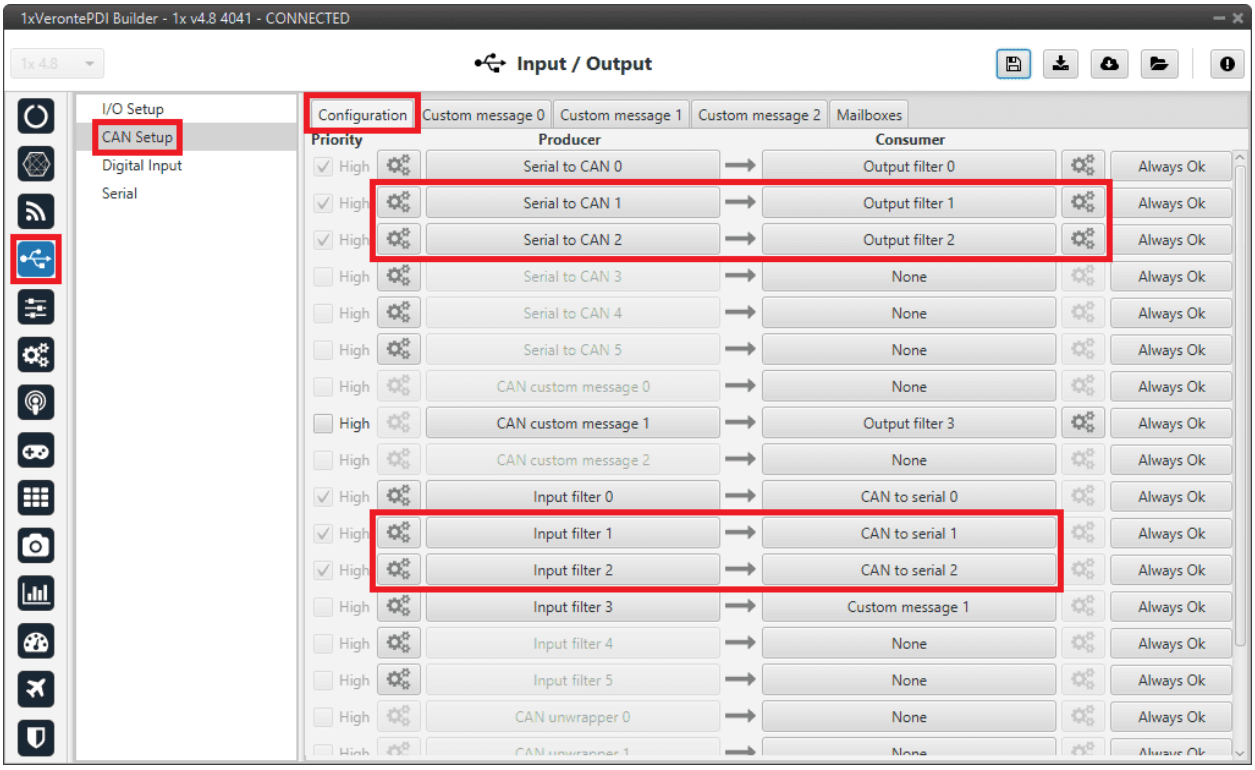


Fig. 173: **Arbiters communication - CAN Setup configuration**

- Set for both **Serial to CAN 1/2** the same **CAN ID: 1302**.

Fig. 174: **Arbiters communication - Serial to CAN 1/2 configuration**

- Select for **Output filter 1** the **CAN A**, as this is the CAN of **Autopilot 1x** that is connected to Arbiter A:

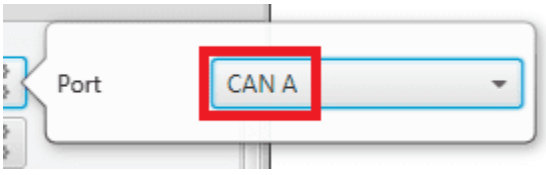


Fig. 175: **Arbiters communication - Output filter 1 configuration**

- Select for **Output filter 2** the **CAN B**, as this is the CAN of **Autopilot 1x** that is connected to Arbiter B:

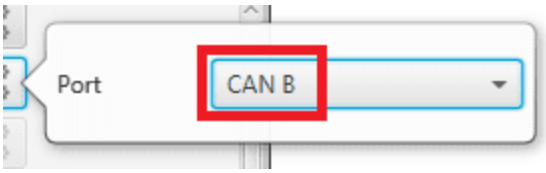


Fig. 176: **Arbiters communication - Output filter 2 configuration**

- Configure both **Input filter 1/2** with **CAN ID: 1301**.

  Select for **Input filter 1** the **CAN A**, as this is the CAN of **Autopilot 1x** that is connected to Arbiter A:

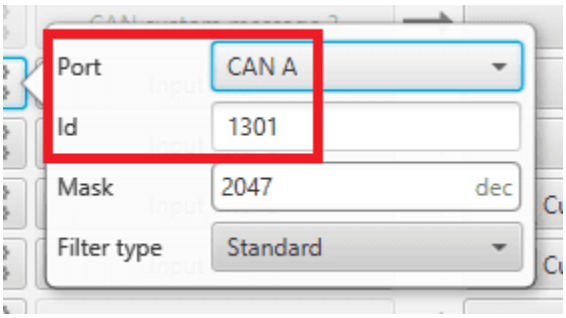


Fig. 177: **Arbiters communication - Input filter 1 configuration**

Select for **Input filter 2** the **CAN B**, as this is the CAN of **Autopilot 1x** that is connected to Arbiter B:

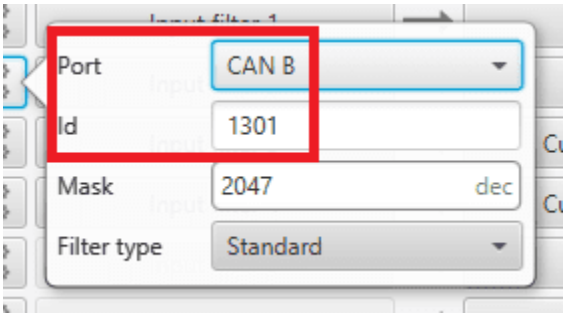


Fig. 178: **Arbiters communication - Input filter 2 configuration**

Note: This is just an example, users can choose *Input filter* and *Output filter* other than 1 and 2.

---

5. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.
   - Set the Baudrate for both CANs, **CAN A: 1 000 000** and **CAN B: 500 000**.
   - Configure at least 10 reception **mailboxes with ID 1301** for both **CAN A and B**:

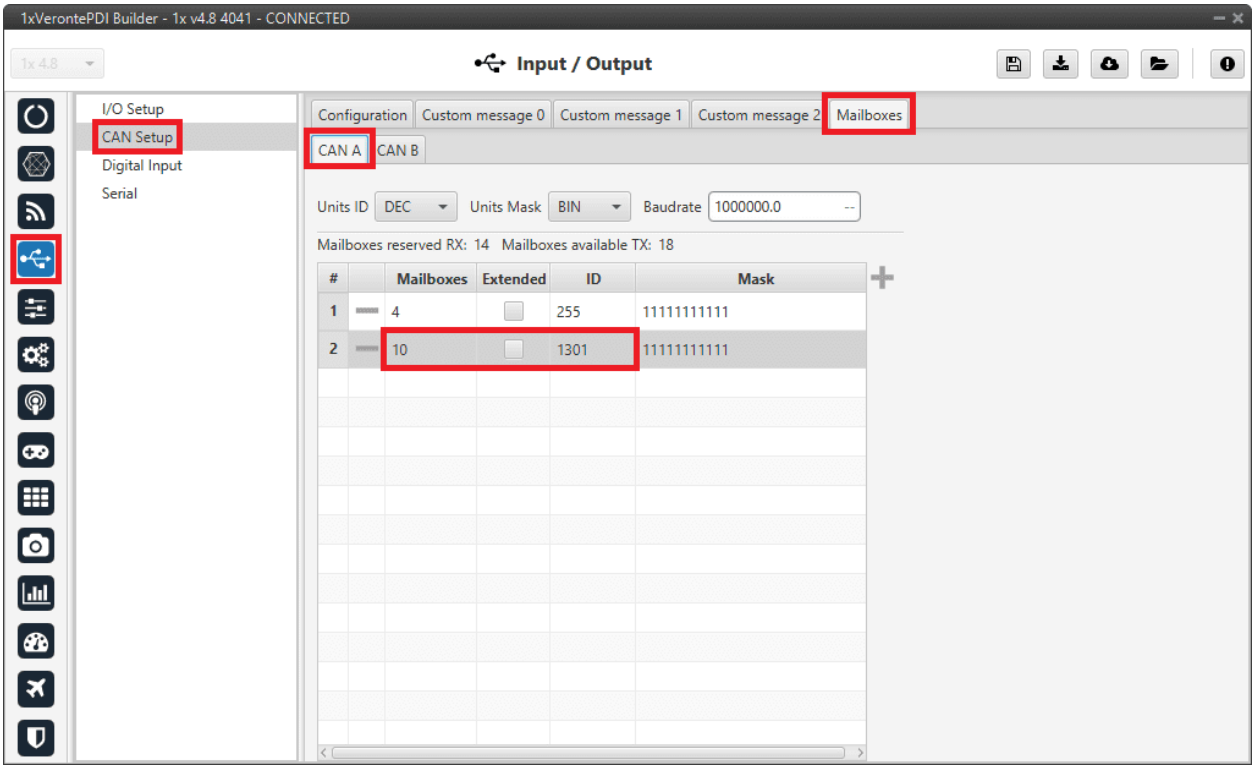


Fig. 179: **Arbiters communication - Mailboxes configuration: CAN A**

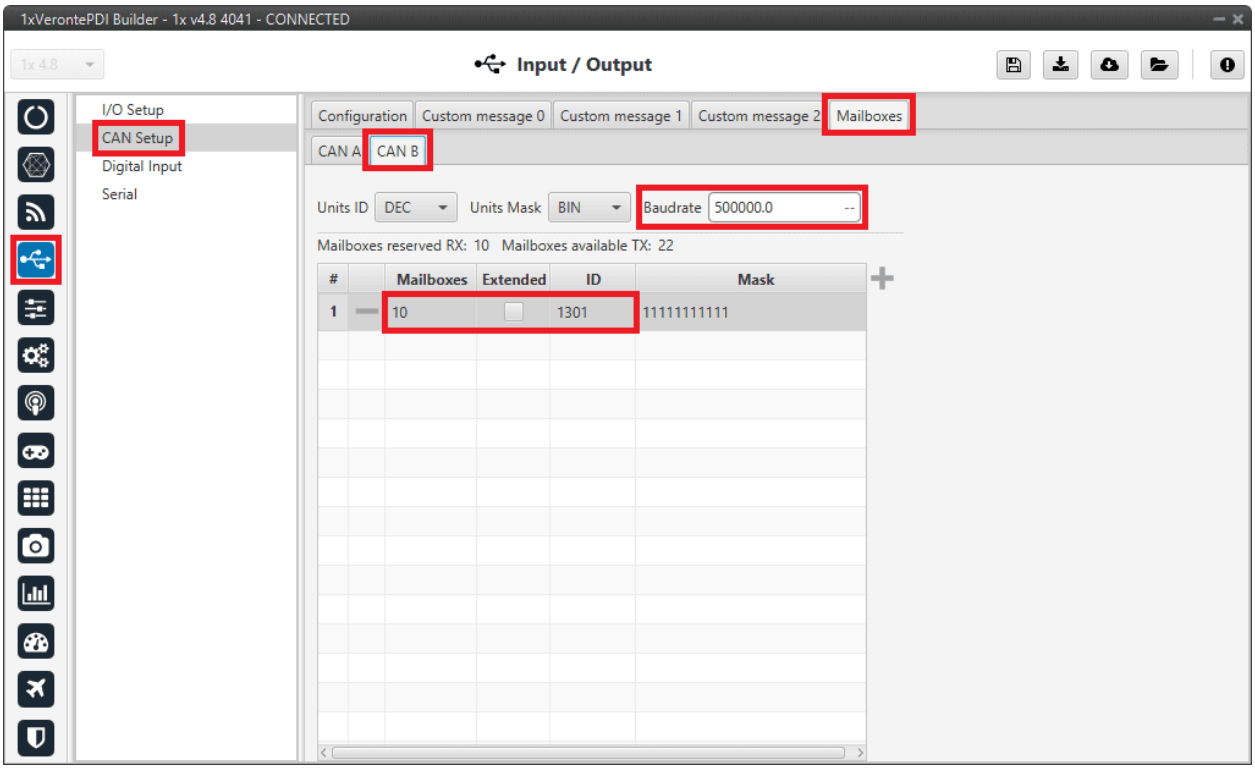


Fig. 180: **Arbiters communication - Mailboxes configuration: CAN B**

3.7.6.2 CEX/MEX

As it is sometimes not possible to connect a CEX/MEX directly to the PC in order to configure it (access **CEX/MEX PDI Builder**), the **Veronte Autopilot 1x** is connected to the computer and a connection is made between **CEX/MEX and Veronte Autopilot 1x via CAN**.

To be able to communicate with CEX/MEX via CAN, the following connection is necessary:

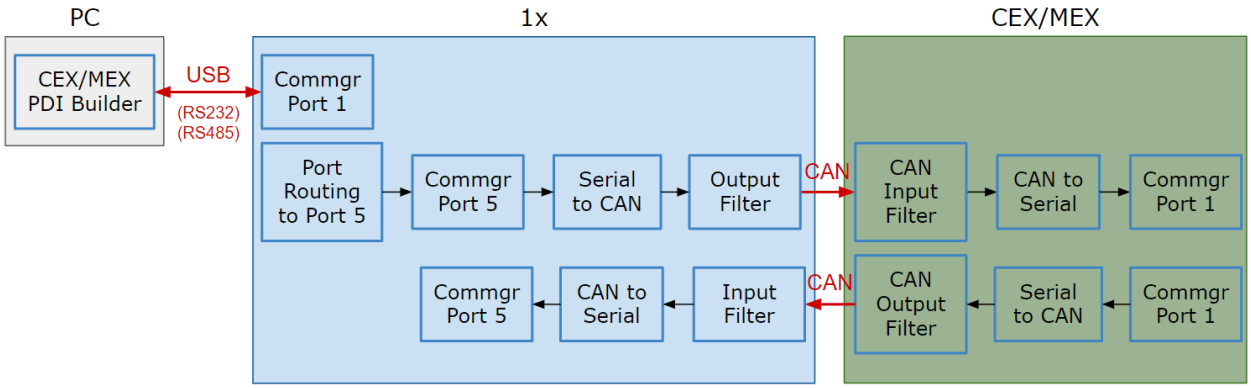


Fig. 181: **Communication diagram PC ↔ CEX/MEX**

Note:

- **1x** usually has this configuration by default, but check it out.
- As the steps to be performed in **CEX PDI Builder and MEX PDI Builder are exactly the same**, **only the steps for one of them will be detailed**. The interface may differ slightly, but the configuration is the same.

Follow the steps below to make this configuration:

3.7.6.2.1 1x PDI Builder side

1. Go to Input/Output menu → **I/O Setup panel**.

   Check the connection between the computer and the **1x** (usually via USB, but RS232 and RS485 are also possible).

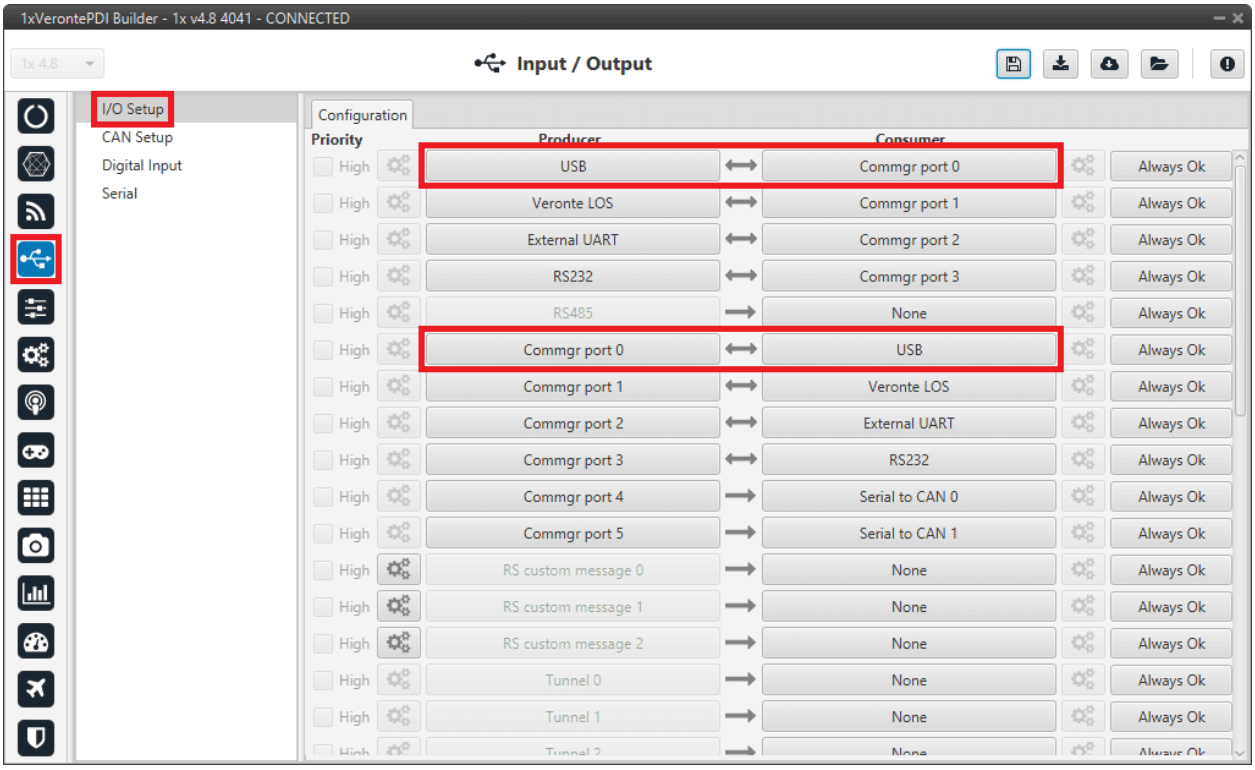


Fig. 182: **1x PDI Builder - I/O Setup configuration**

2. Go to Communications menu → **Ports panel**.

   Remove Port 5 from the Forward group and **add Port 5 to the Route group**, with target CEX's Address:

   ⇒ **Address = 44000 + Serial number**.

   The CEX address must be in the **range 45000 - 49999**.

Note:

- For **MEX**, the address should look like this:
  - **Address = 42000 + Serial number**.
  - The MEX address must be in the **range 43000 - 43999**.
- If the theorical address does not work, 999 (unknown) can be used as sometimes the address has not been set in CEX/MEX.

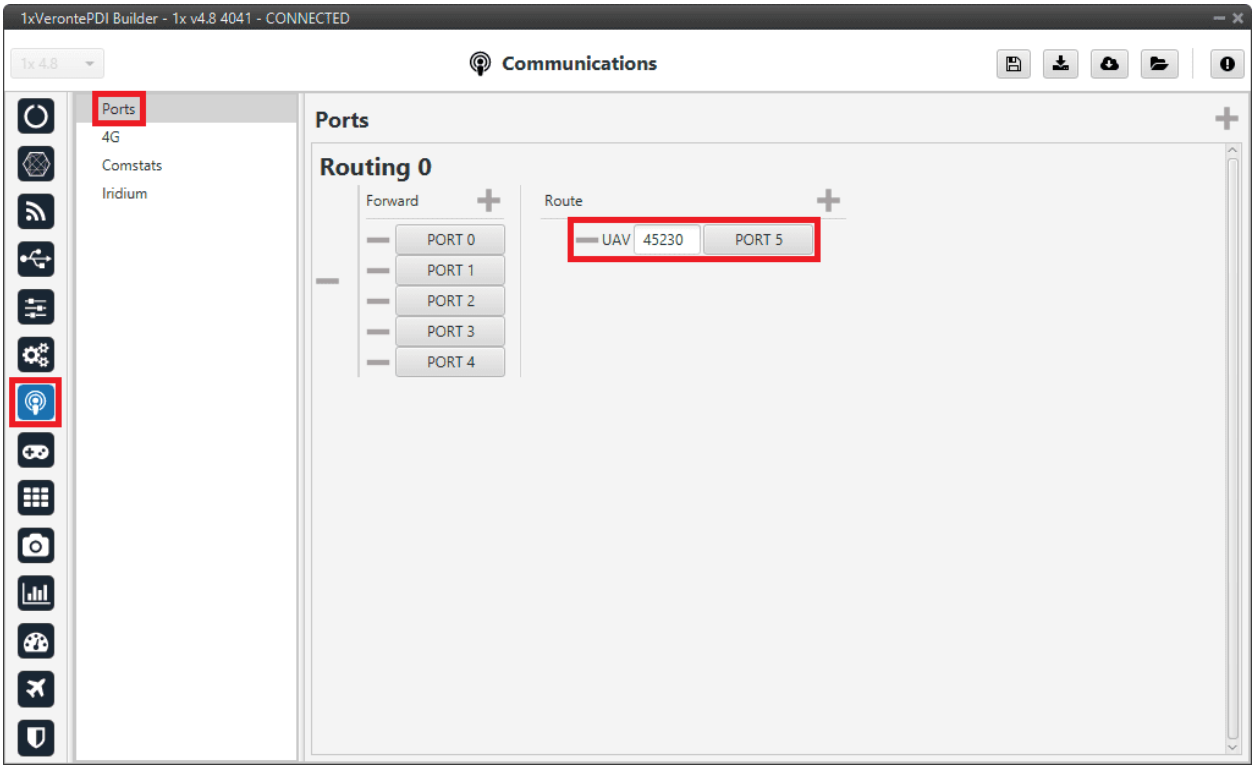


Fig. 183: **1x PDI Builder - Routing configuration**

3. Go to Input/Output menu → **I/O Setup panel**.

   Connect **Commgr port 5** to **Serial to CAN 1** consumer:

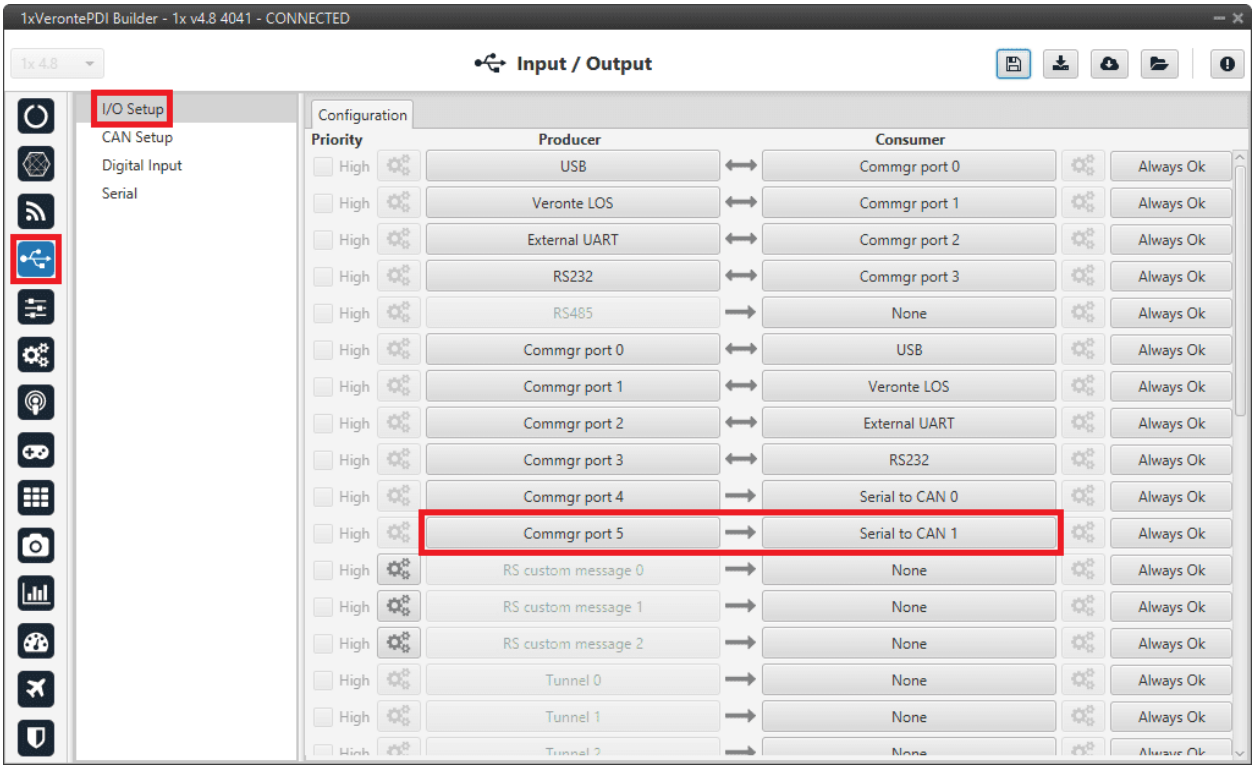


Fig. 184: **1x PDI Builder - I/O Setup configuration: Serial to CAN**

Then, connect **CAN to serial 1** to **Commgr port 5**:

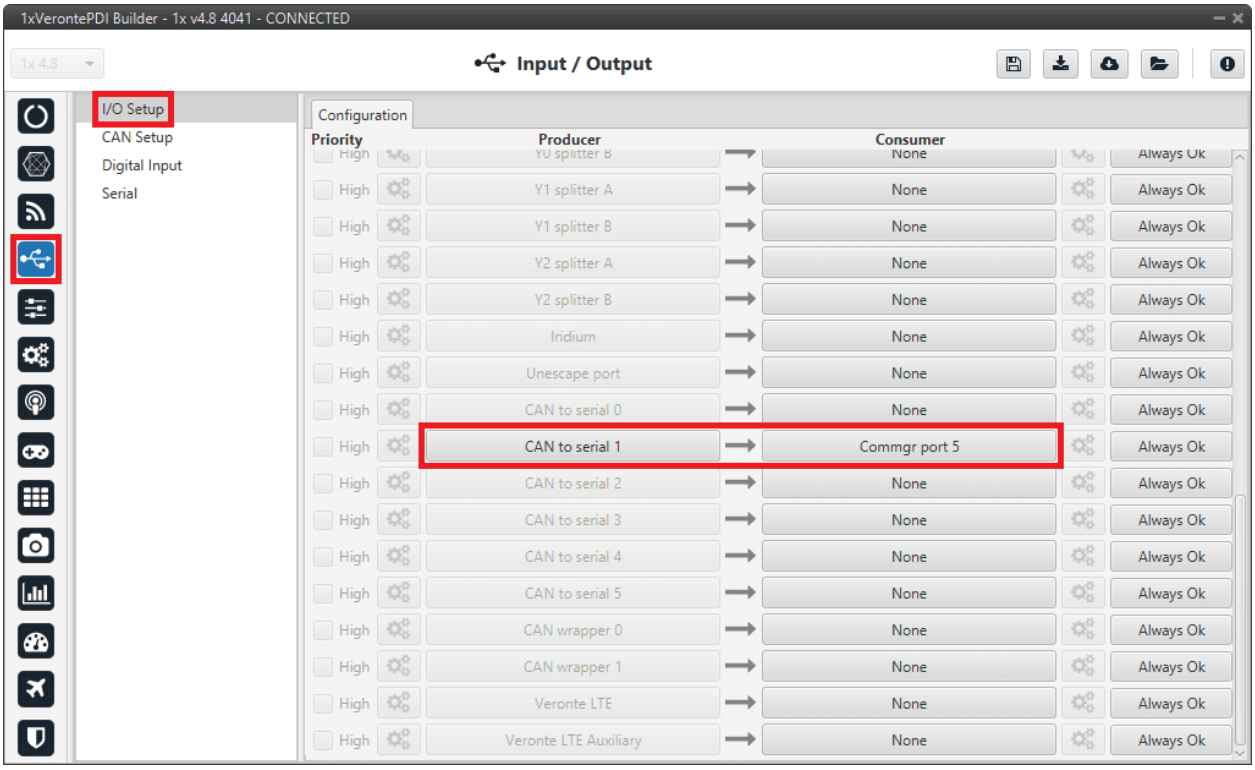


Fig. 185: **1x PDI Builder - I/O Setup configuration: CAN to serial**

4. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect a **Serial to CAN** with the right Id (**CAN ID 1302**) to an **Output filter**.

   In addition, connect an **Input filter** with the right Id (**CAN ID 1301**) to a **CAN to serial**:

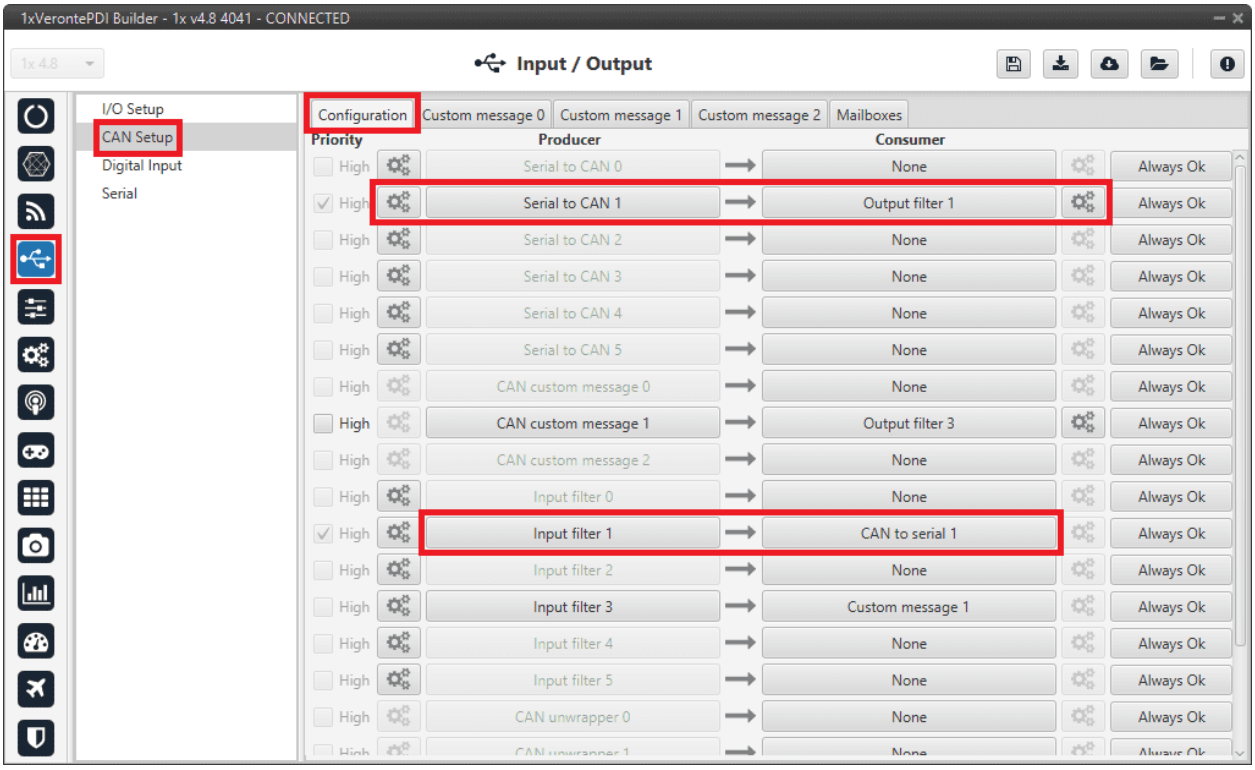


Fig. 186: **1x PDI Builder - CAN Setup configuration**

Fig. 187: **1x PDI Builder - Serial to CAN configuration**

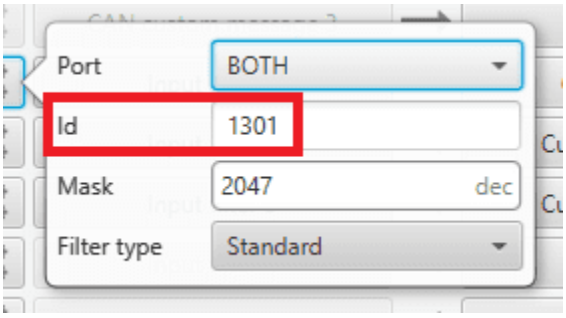


Fig. 188: **1x PDI Builder - Input filter configuration**

5. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

Finally, configure the reception **mailbox with ID 1301**, assign at least 4 mailboxes:

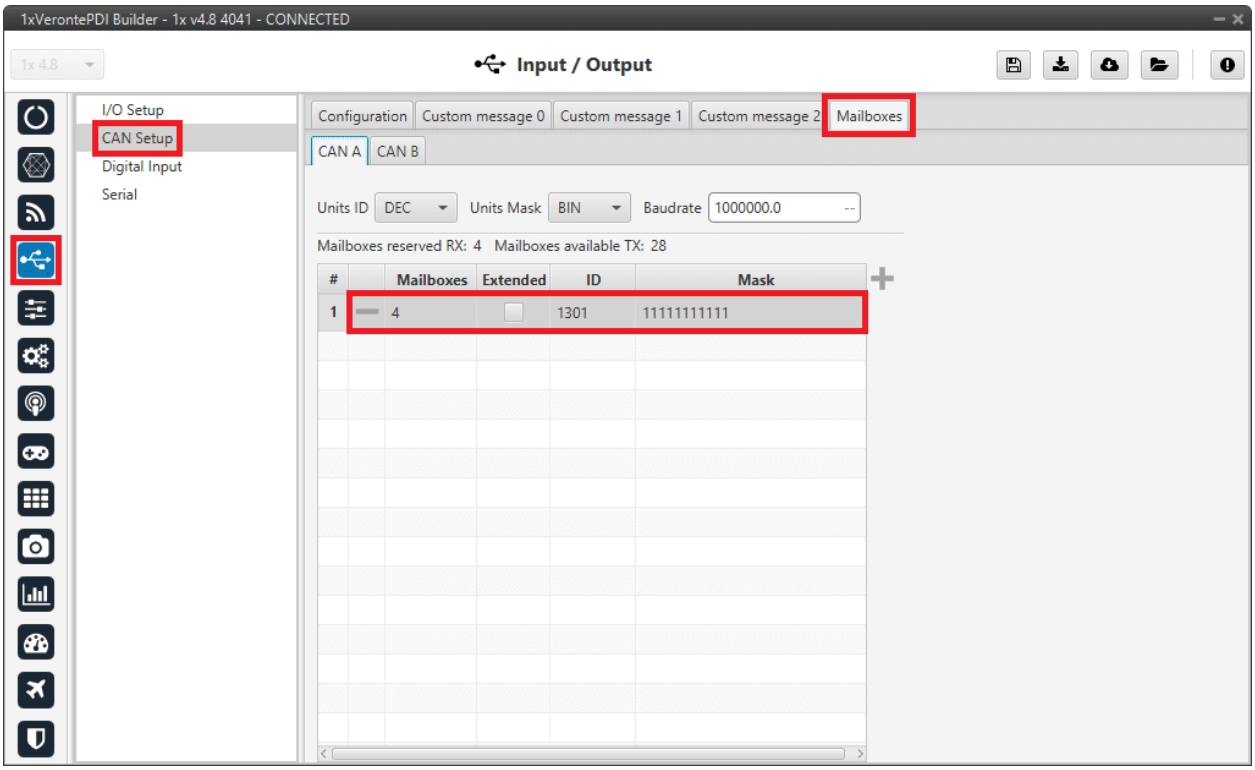


Fig. 189: **1x PDI Builder - Mailboxes configuration**

3.7.6.2.2 CEX PDI Builder side

Note: This part is already built for CEX default configuration, but the user can check it.

6. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Connect a **CAN Input Filter** with the right CAN Address (**CAN ID 1302**) to **CAN to Serial 0**.

   In addition, connect **Serial to CAN 0** with the right CAN Address (**CAN ID 1301**) to a **CAN Output Filter** port:

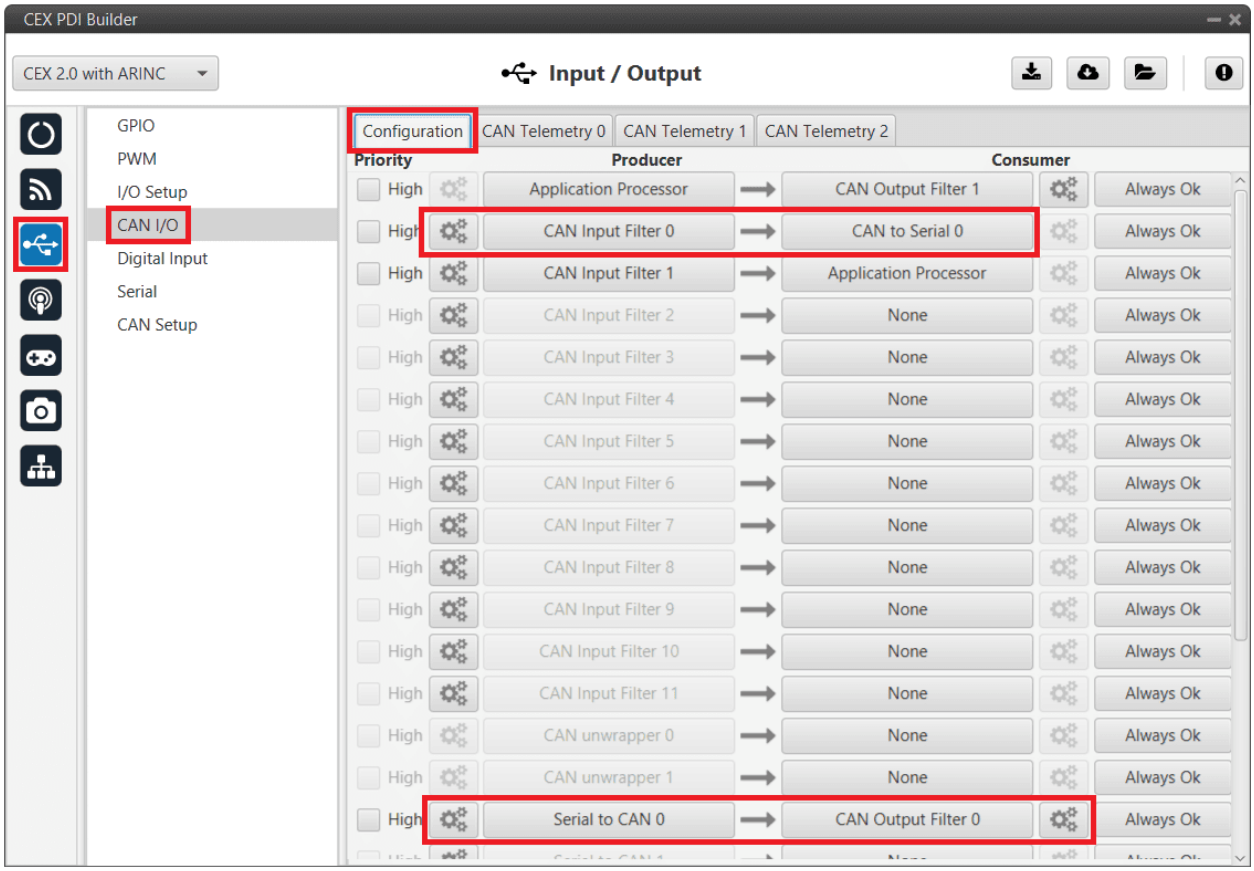


Fig. 190: **CEX PDI Builder - CAN I/O configuration**

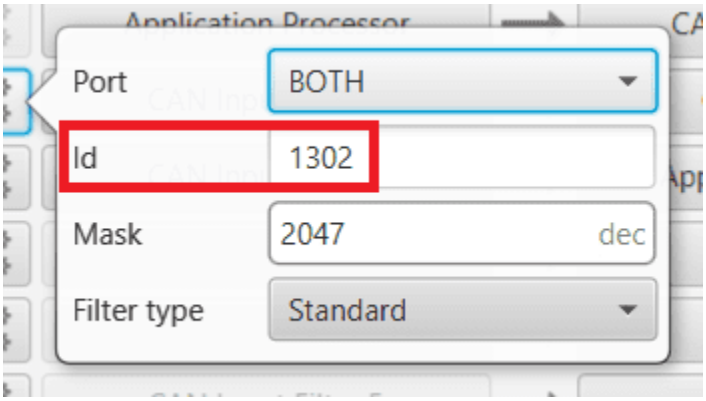


Fig. 191: **CEX PDI Builder - CAN Input Filter configuration**

Fig. 192: **CEX PDI Builder - Serial to CAN configuration**

7. Go to Input/Output menu → **I/O Setup panel**.

   Connect **CAN to Serial 0** to any **Commgr port**, in this case *Commgr port 0* is used.

   In addition, connect **Commgr port 0** to **Serial to CAN 0** consumer:

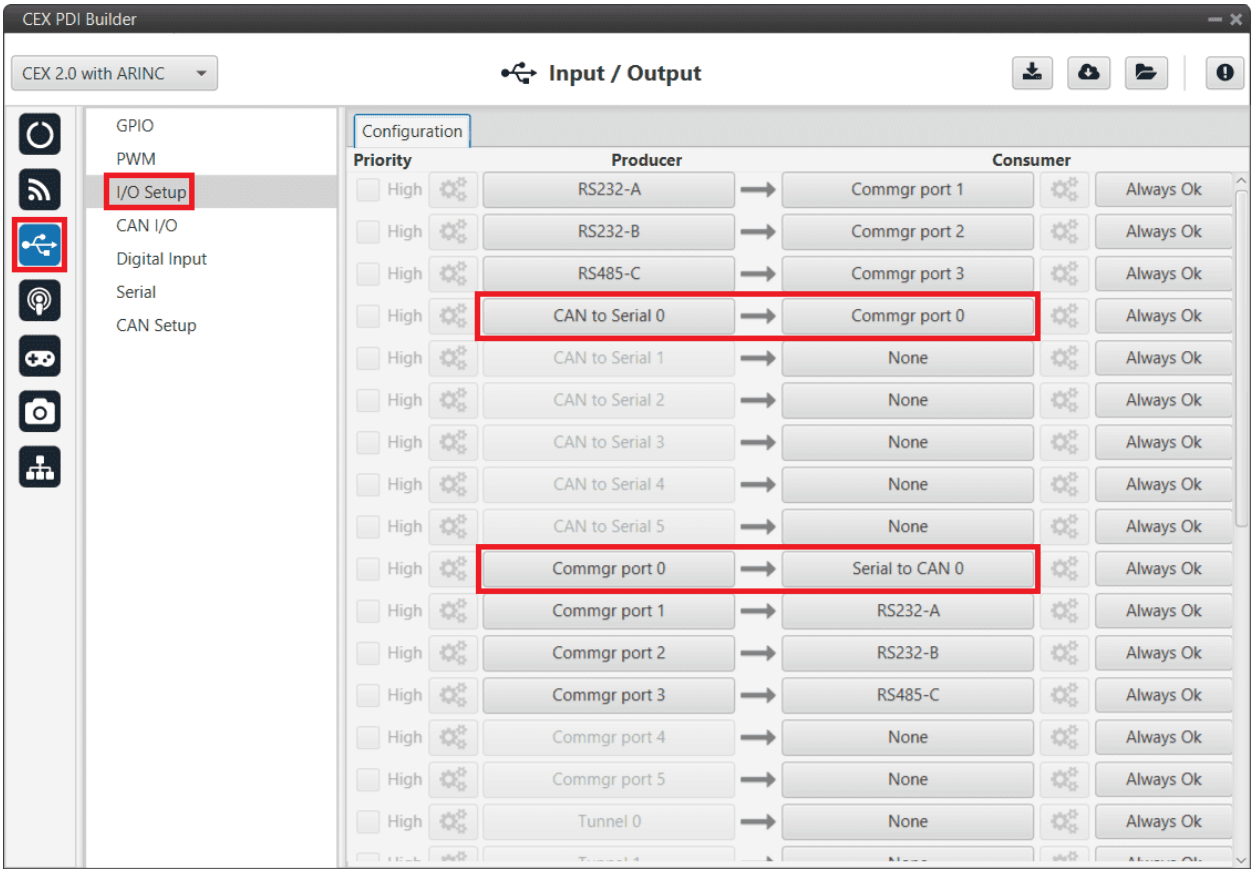


Fig. 193: **CEX PDI Builder - I/O Setup configuration**

8. Go to Input/Output menu → **CAN Setup panel**.

   Finally, configure the reception **mailbox with ID 1302**, assign at least 4 mailboxes:

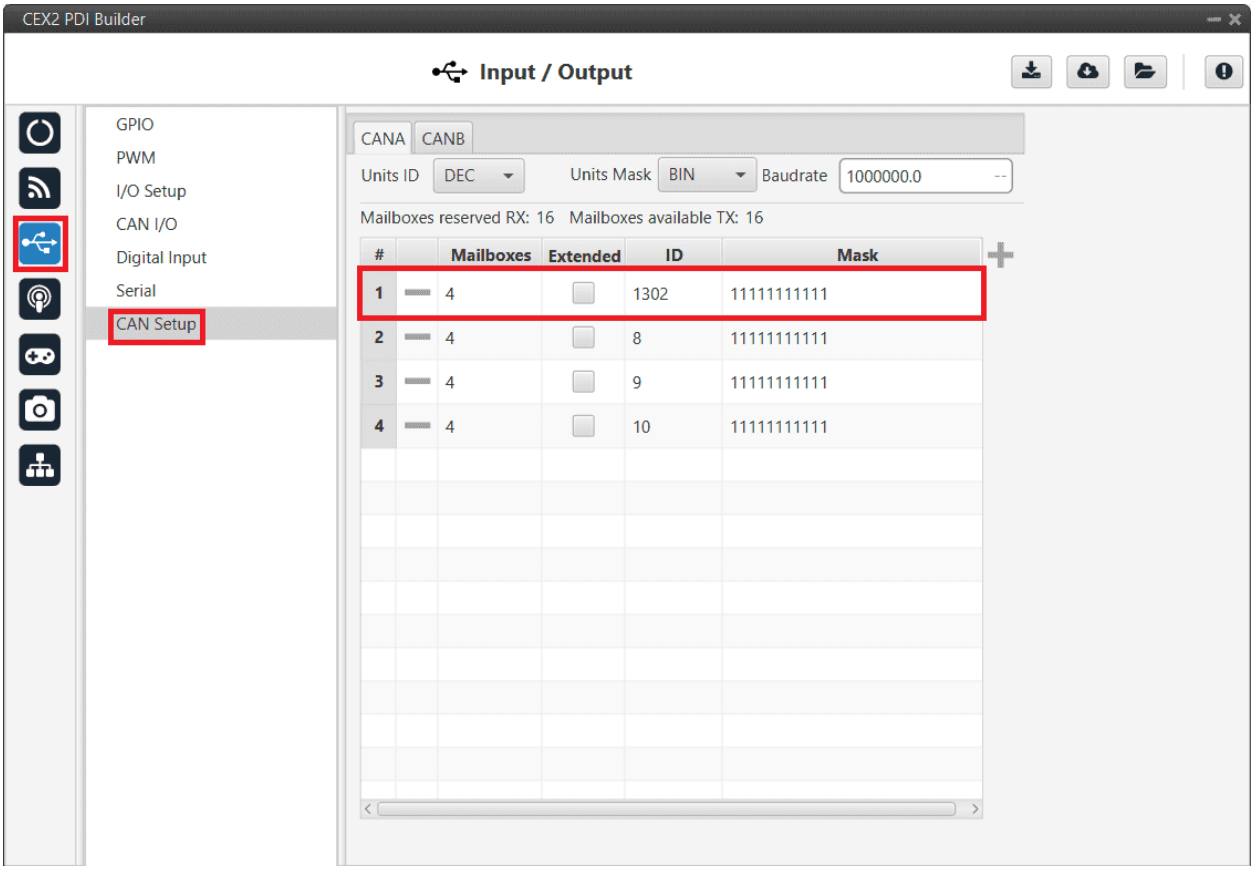


Fig. 194: **CEX PDI Builder - CAN Setup (Mailboxes) configuration**

3.7.6.3 MC01

In order to communicate a **Veronte Autopilot 1x** with a **MC01** via CAN, the following connection is required:

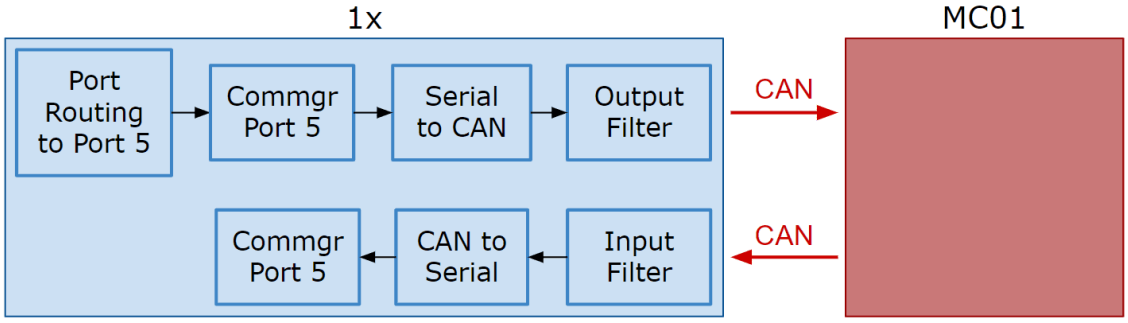


Fig. 195: **Communication diagram 1x ↔ MC01**

The following steps explain how to configure the communication between an **Autopilot 1x** and a **MC01**.

3.7.6.3.1 MC01 PDI Builder side

1. By default, **MC01** is configurated with a connection Serial to CAN, with the following **Standard** CAN IDs:
   - Tx CAN Id: 1301
   - Rx CAN Id: 1302

3.7.6.3.2 1x PDI Builder side

2. Go to Communications menu → **Ports panel**.

   Remove Port 5 from the Forward group and **add Port 5 to the Route group**, with target MC01's Address. This address must be chosen in the destination path of the **MC01** (40117 for the example).

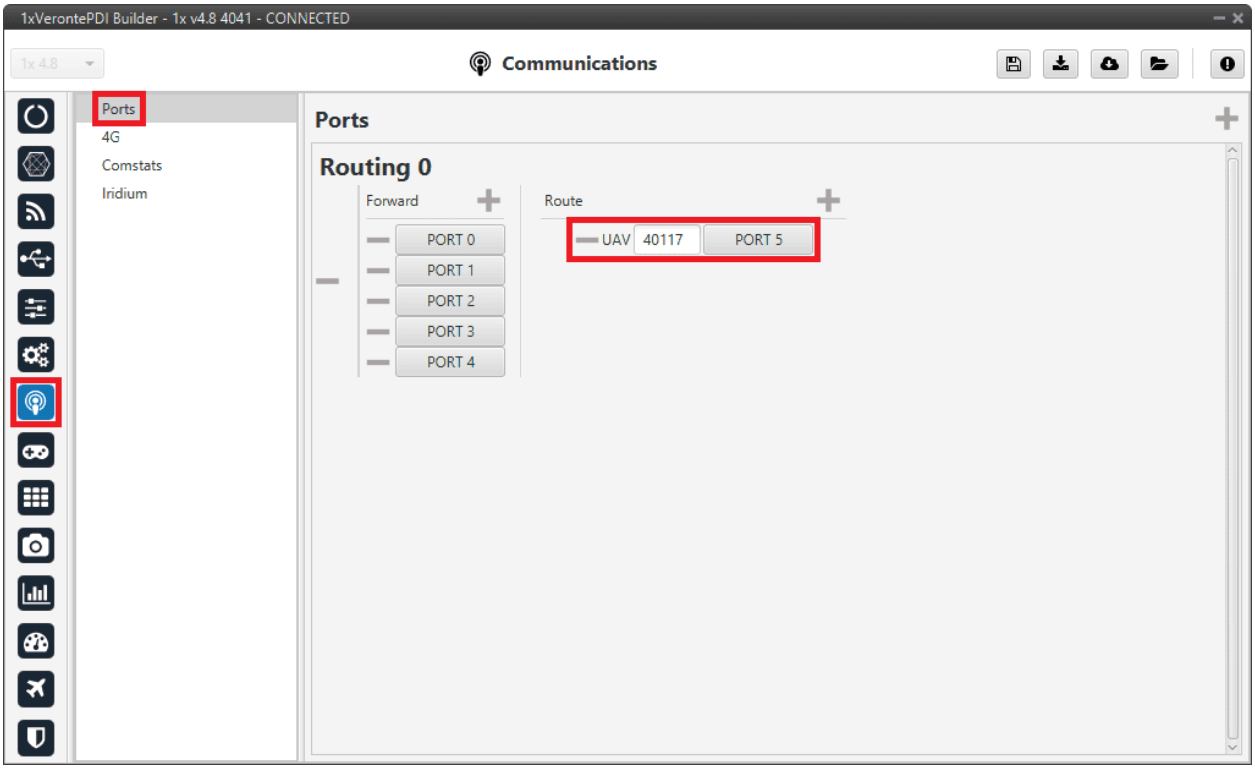


Fig. 196: **1x PDI Builder - Routing configuration**

3. Go to Input/Output menu → **I/O Setup panel**.

   Connect the **Commgr port 5** to the **Serial to CAN 1**.

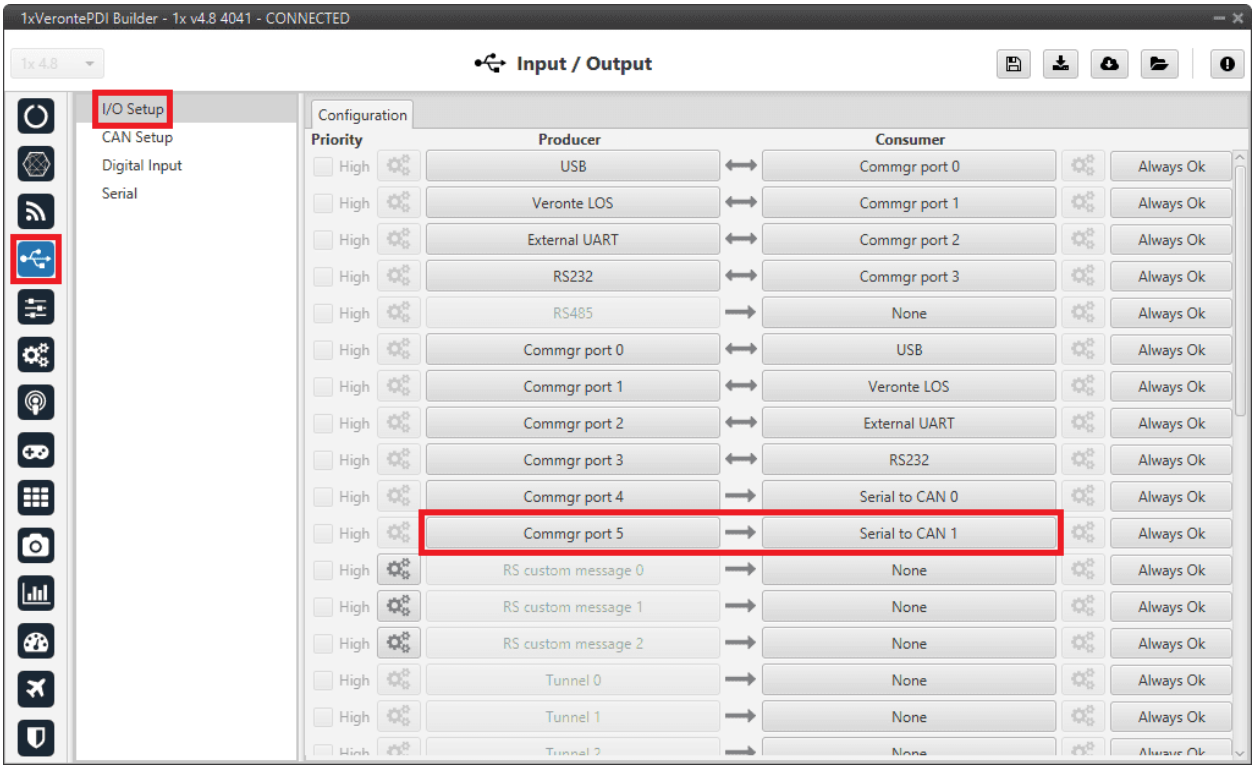


Fig. 197: **1x PDI Builder - I/O Setup configuration: Serial to CAN**

Then, connect **CAN to serial 1** to **Commgr port 5**:

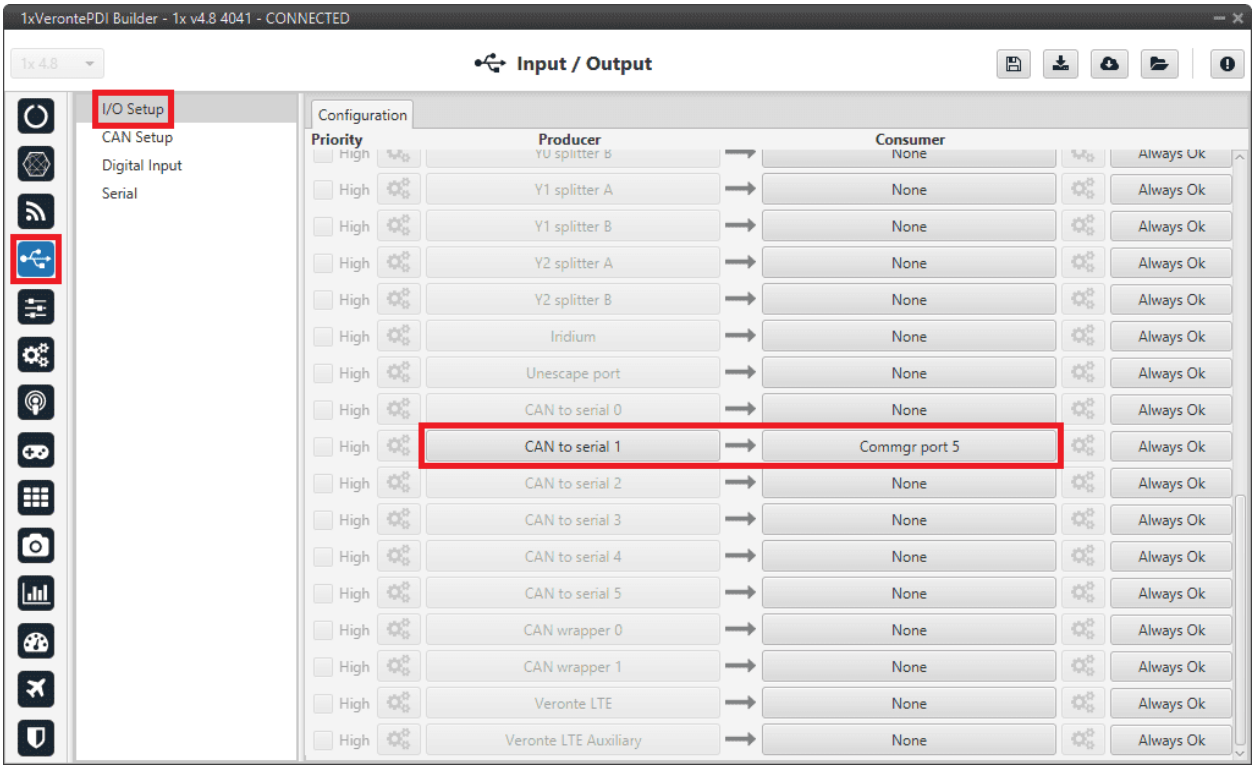


Fig. 198: **1x PDI Builder - I/O Setup configuration: CAN to serial**

4. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect a **Serial to CAN** with the right Id (**CAN ID 1302**) to an **Output filter**.

   In addition, connect an **Input filter** with the right Id (**CAN ID 1301**) to a **CAN to serial**:

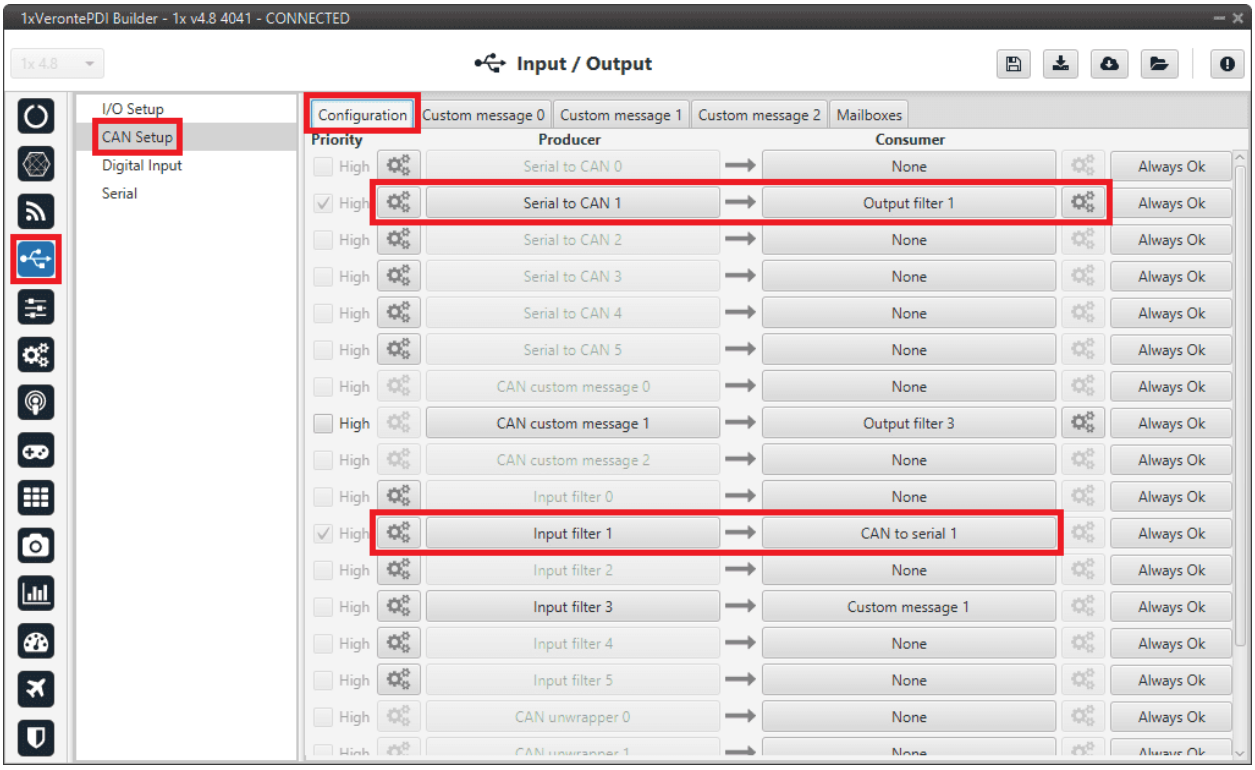


Fig. 199: **1x PDI Builder - CAN Setup configuration**

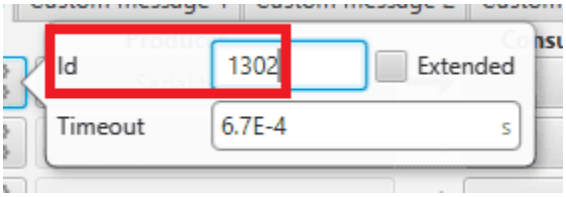


Fig. 200: **1x PDI Builder - Serial to CAN configuration**

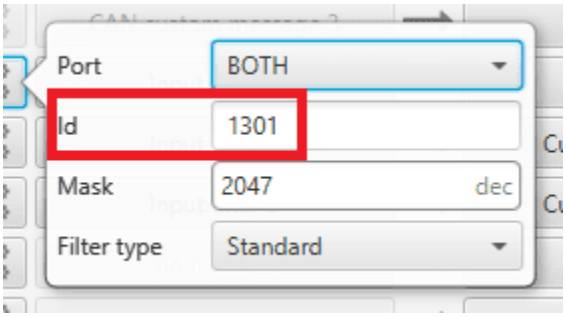


Fig. 201: **1x PDI Builder - Input filter configuration**

5. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Finally, configure the reception **mailbox with ID 1301**, assign at least 1 mailbox:

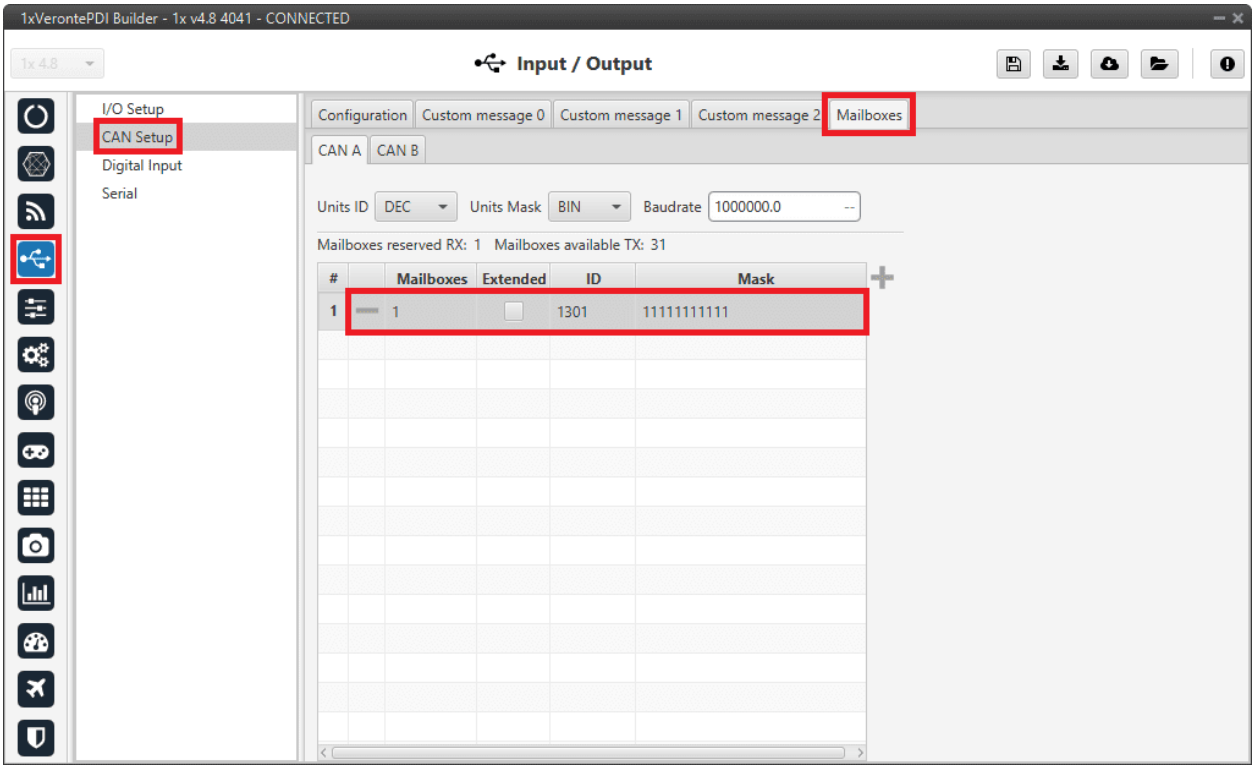


Fig. 202: **1x PDI Builder - Mailboxes configuration**

3.7.6.4 MC110/MC24

In order to send commands from a **Veronte Autopilot 1x** to a **MC110/MC24** via CAN and vice versa, the following connection is required:

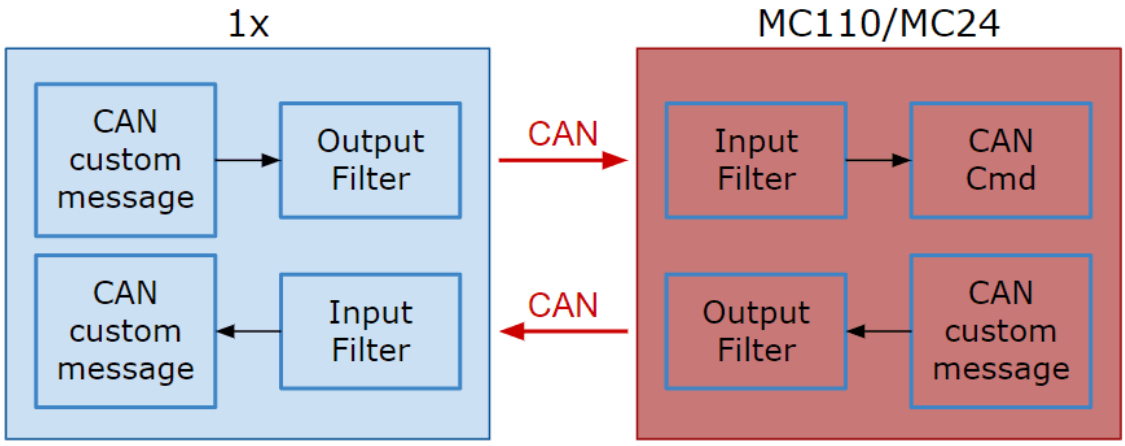


Fig. 203: **Communication diagram 1x ↔ MC110/MC24**

Note: As the steps to be performed in **MC110 PDI Builder and MC24 PDI Builder are exactly the same, only the steps for one of them will be detailed**. The interface may differ slightly, but the configuration is the same.

3.7.6.4.1 CAN commands from Autopilot 1x to MC110

Follow the steps below to make this configuration:

3.7.6.4.1.1 1x PDI Builder side

1. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect a **CAN custom message** producer (in this case *CAN custom message 1* is used) to an **Output filter** consumer, in this example *Output filter 3*.

   In addition, configure the **Output filter** with the **correct CAN Bus**, in this example CAN A has been selected:

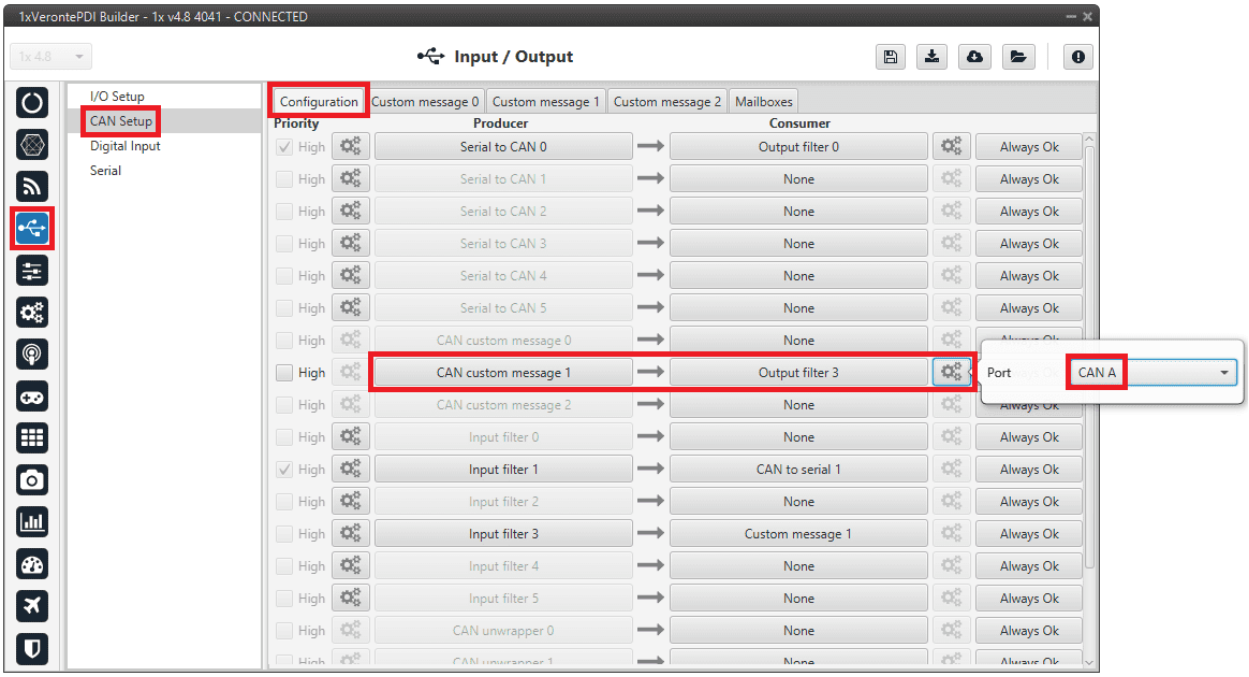


Fig. 204: **1x PDI Builder - CAN Setup configuration**

2. Go to Input/Output menu → CAN Setup panel → **Custom message 1 tab** (since the producer *CAN custom message 1* has been connected to the output filter).

   - Add a new message in **TX** (as it is for tranmission) with **CAN ID 1434**. More information on the configuration of CAN messages can be found in the *TX/TX Ini Messages (Custom Messages) - Input/Output* section of this manual.

   - Next, configure the message to be sent with whatever variable users wish to use to command. The variable should be set to **compressed signed 32-bit**.

     Users should send the values from **0 to max_rpm** (or from **-max_rpm to max_rpm** if negative commands are desired to be allowed).

To do this, it is recommended to control the variable internally as a throttle, for this set the **Encode from 0 to 1** (or **from -1 to 1** for negative speeds). And for decode it to rpm values, the **Decode** parameter must be configured from **0 to max_rpm** (or from **-max_rpm to max_rpm** if negative commands are allowed):

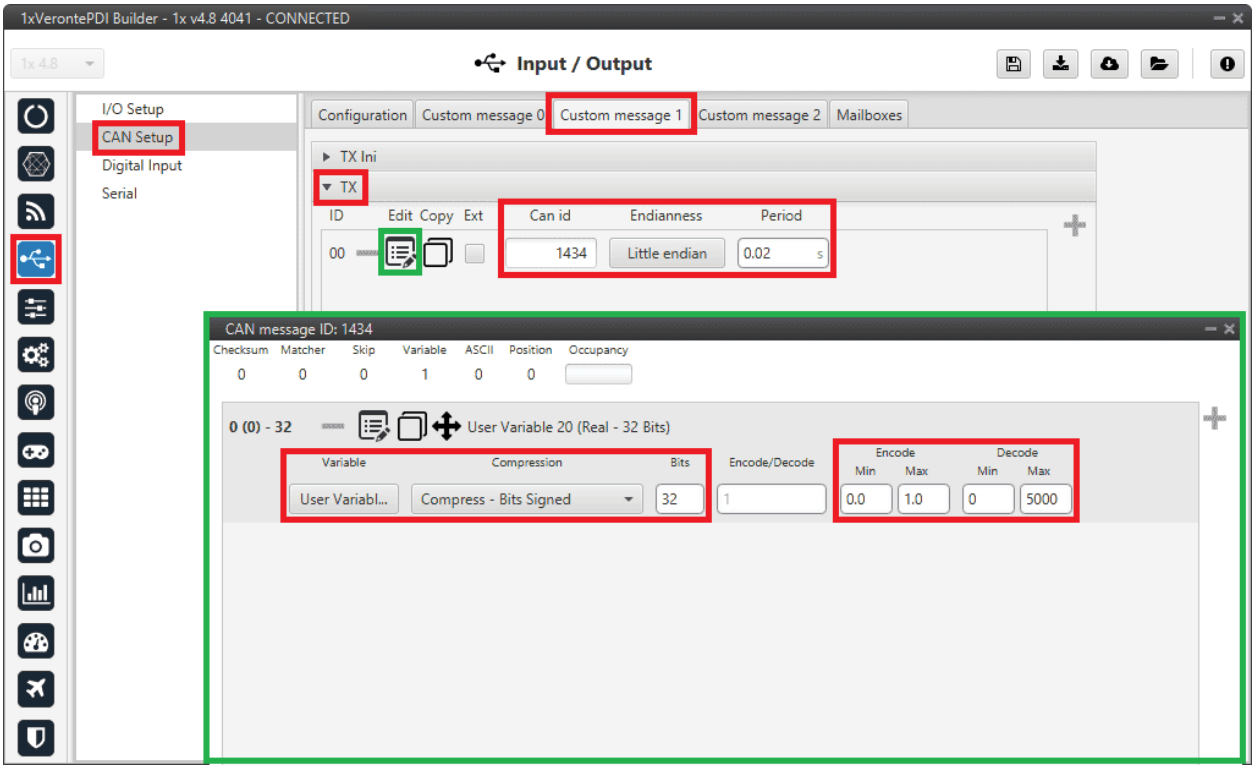


Fig. 205: **1x PDI Builder - CAN custom message 1 configuration**

For more information on configuring CAN custom messages, refer to the *Custom Messages types - Input/Output* section of this manual.

Warning: Remember that it is necessary to have **at least 1 free mailbox for TX messages**.

3.7.6.4.1.2 MC110 PDI Builder side

3. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Connect an **Input filter** producer, in this example *Input filter 1*, to the **CAN Cmd** consumer.

   In addition, configure the input filter with the following parameters:

   - Port: **CAN A**
   - **CAN Id 1434**
   - Mask: **2047** dec
   - Filter type: **Standard**

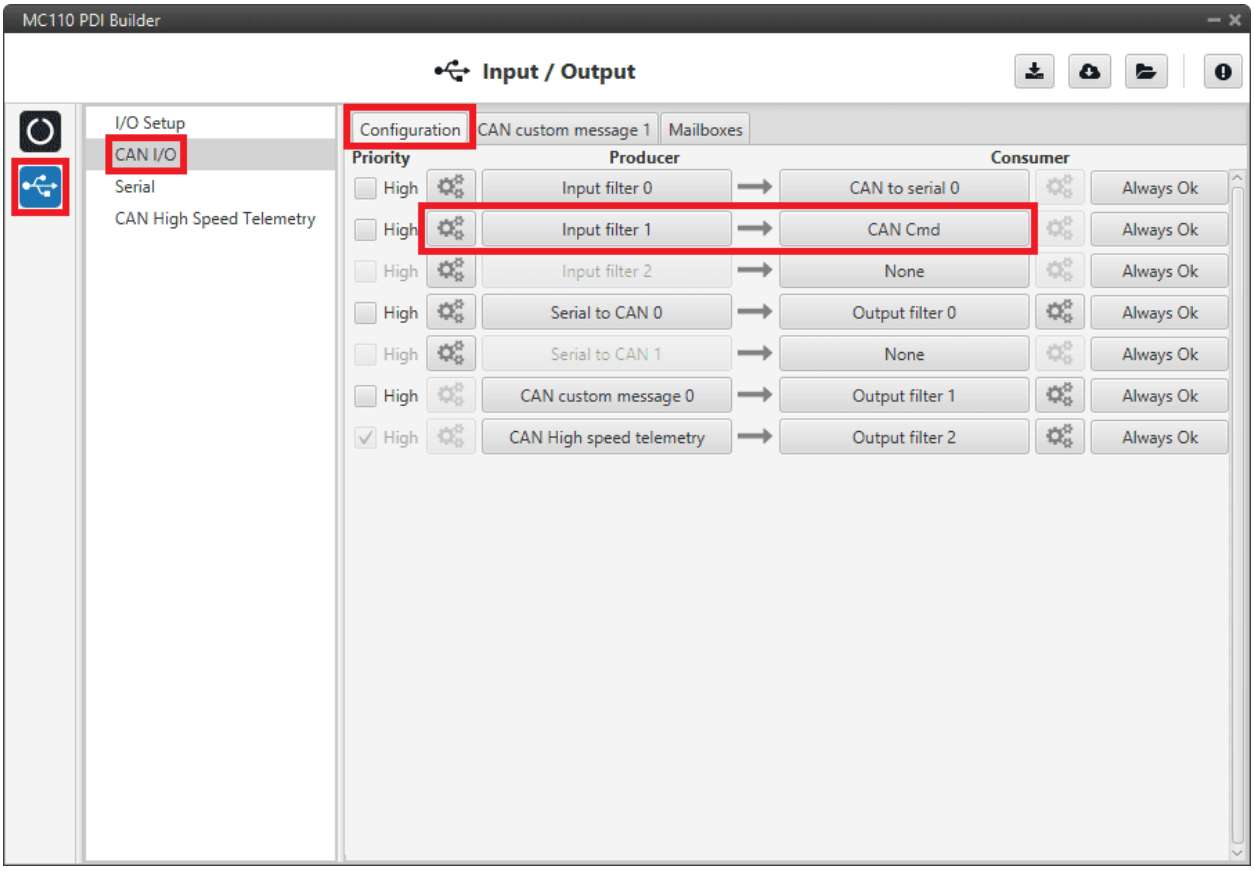


Fig. 206: **MC110 PDI Builder - CAN I/O configuration**

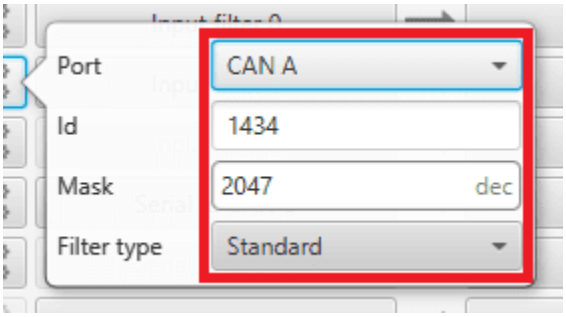


Fig. 207: **MC110 PDI Builder - Input filter configuration**

4. Go to Input/Output menu → CAN I/O panel → **Mailboxes tab**.

   Configure at least 4 reception mailboxes with **ID 1434** in the **CAN A** bus:

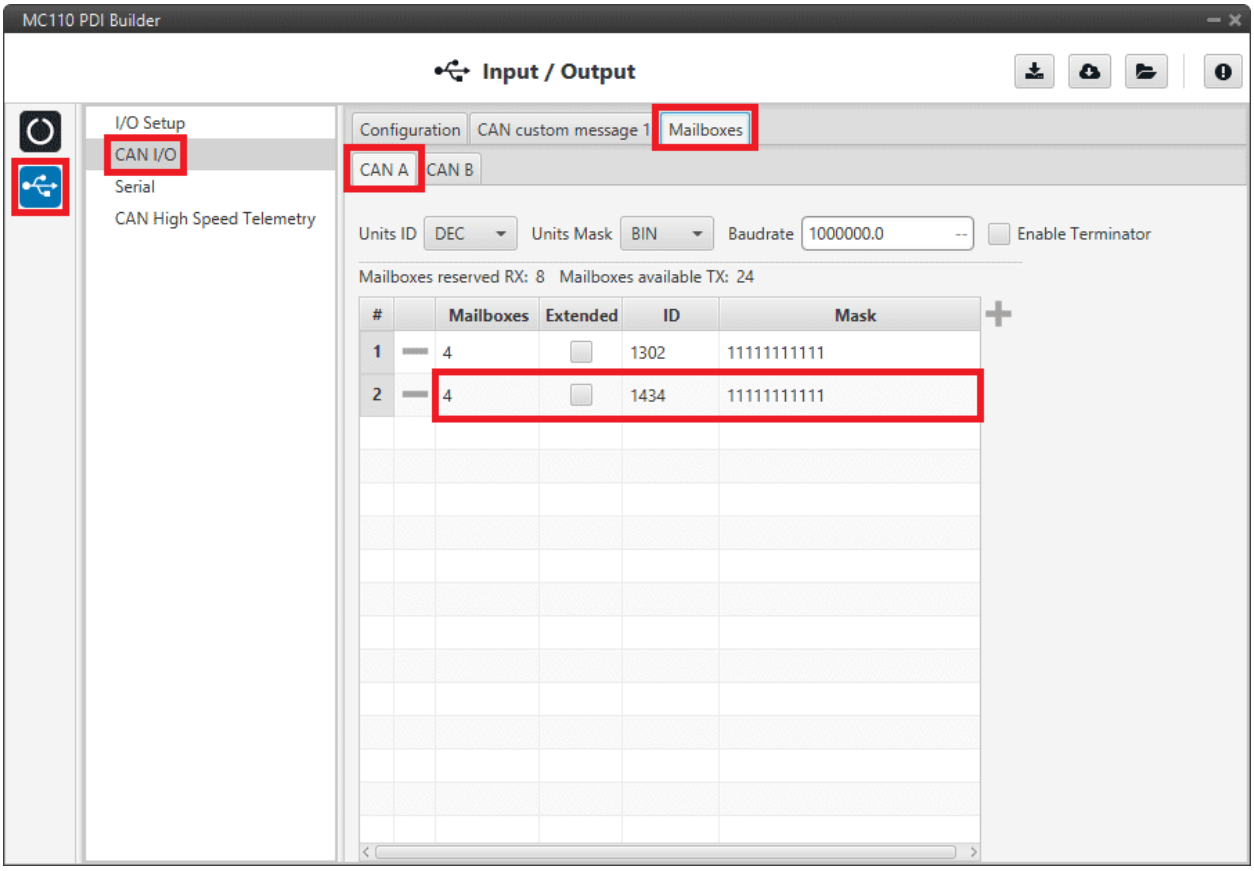


Fig. 208: **MC110 PDI Builder - Mailboxes configuration**

5. Go to MC menu → FOC Control panel → **Control Input**.

   Make sure that **m_CAN** or **m_CAN_PPM** mode is selected.

   For more information on these parameters, refer to the Control Input (FOC Control) - MC section of **MC110 PDI Builder** user manual.

3.7.6.4.2 CAN commands from MC110 to Autopilot 1x

Follow the steps below to make this configuration:

3.7.6.4.2.1 MC110 PDI Builder side

1. Go to Input/Output menu → CAN I/O panel → **Configuration tab**.

   Connect **CAN custom message 0** producer to an **Output filter** consumer, in this example *Output filter 1*.

   In addition, configure the **Output filter** with the **correct CAN Bus**, in this example CAN A has been selected:

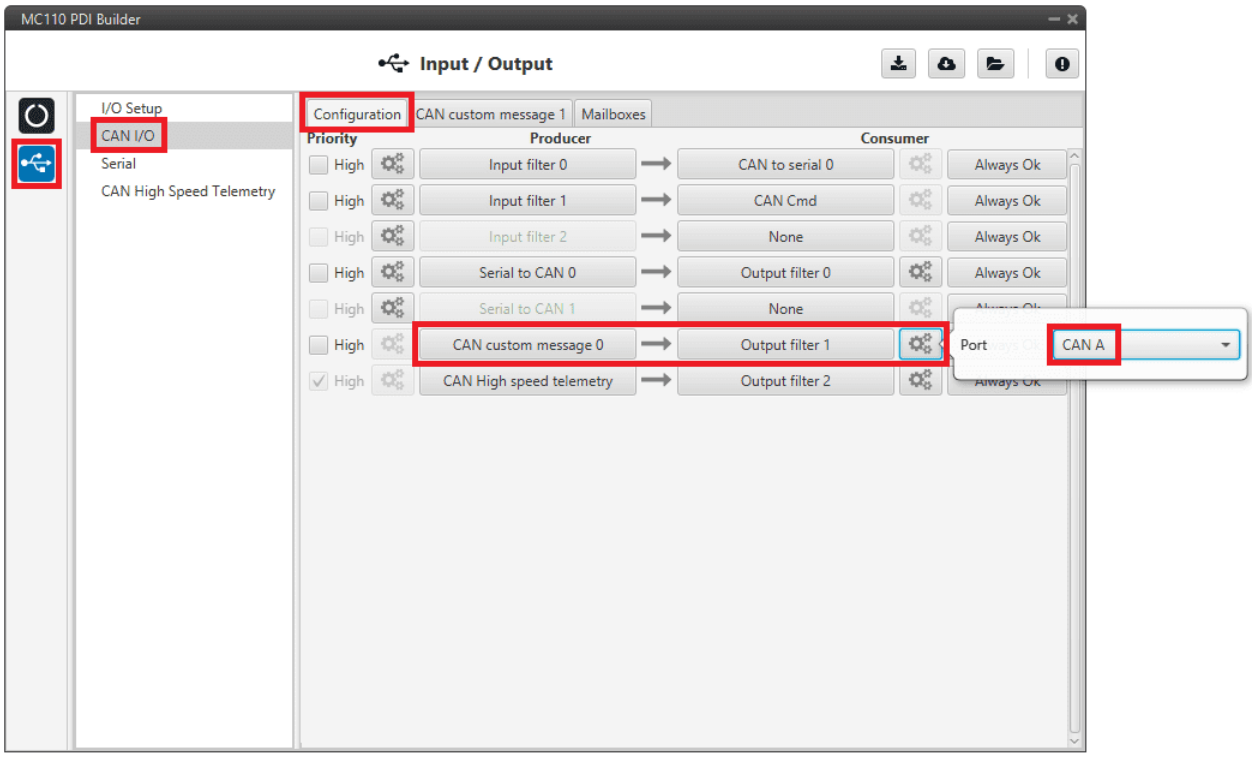


Fig. 209: **MC110 PDI Builder - CAN I/O configuration**

2. Go to Input/Output menu → CAN I/O panel → **CAN custom message 1 tab**.

   Add a new message in **TX** with the variables the user wishes to send back to **Autopilot 1x**.

   In this example, the message with **CAN ID 100** is sending the **input command value** as well as the **board temperature** as **uncompressed variables**.

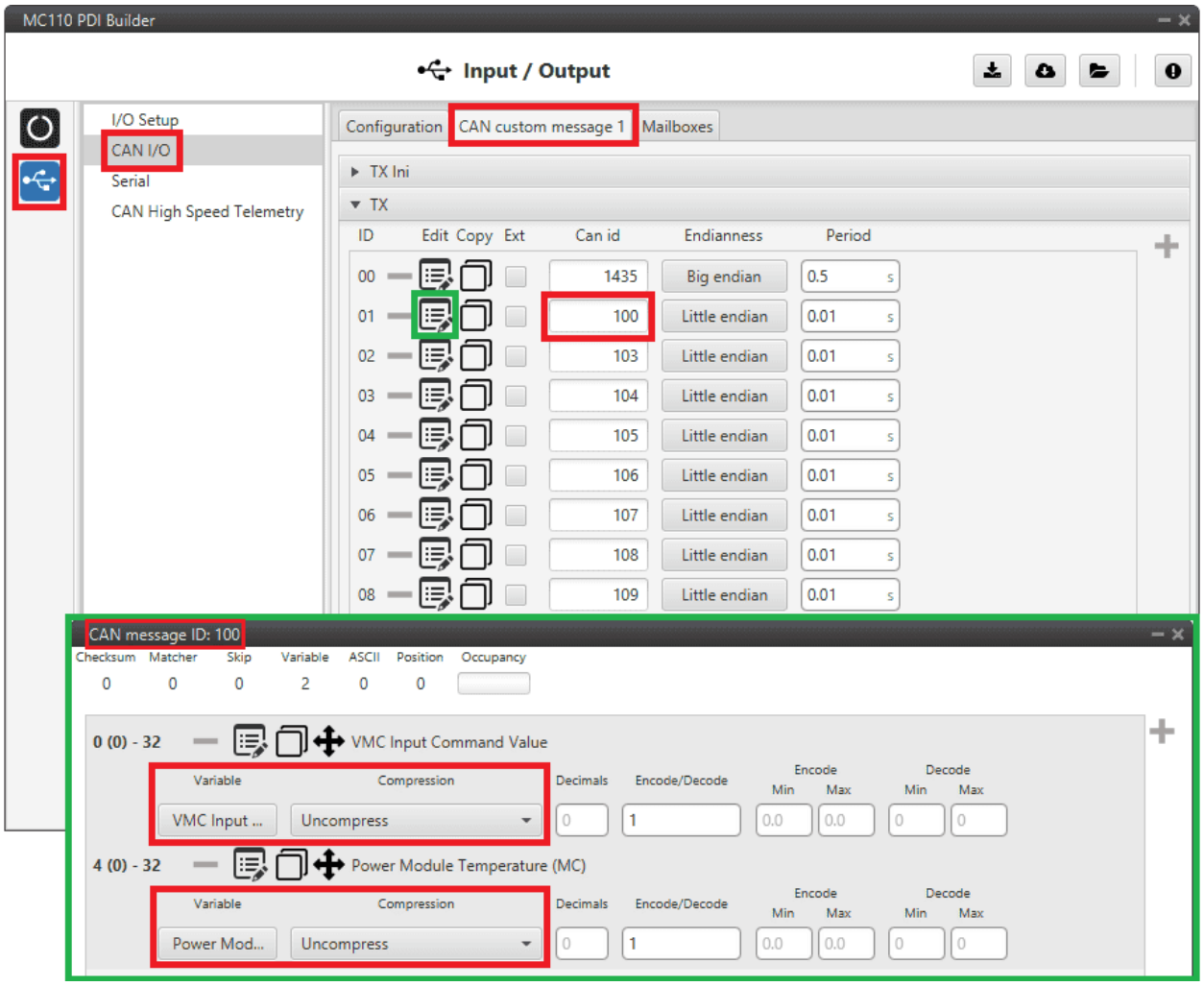


Fig. 210: **MC110 PDI Builder - CAN custom message 1 configuration**

For more information on configuring CAN custom messages, refer to the *Custom Messages types - Input/Output* section of this manual.

Note: If the variables are compressed/encoded on the MC110 side when sent, they must be decompressed/decoded on the **Autopilot 1x** unit on reception.

3.7.6.4.2.2 1x PDI Builder side

3. Go to Input/Output menu → CAN Setup panel → **Configuration tab**.

   Connect an **Input filter** producer (in this case *Input filter 3*) to a **Custom message** consumer (*Custom message 1* has been selected).

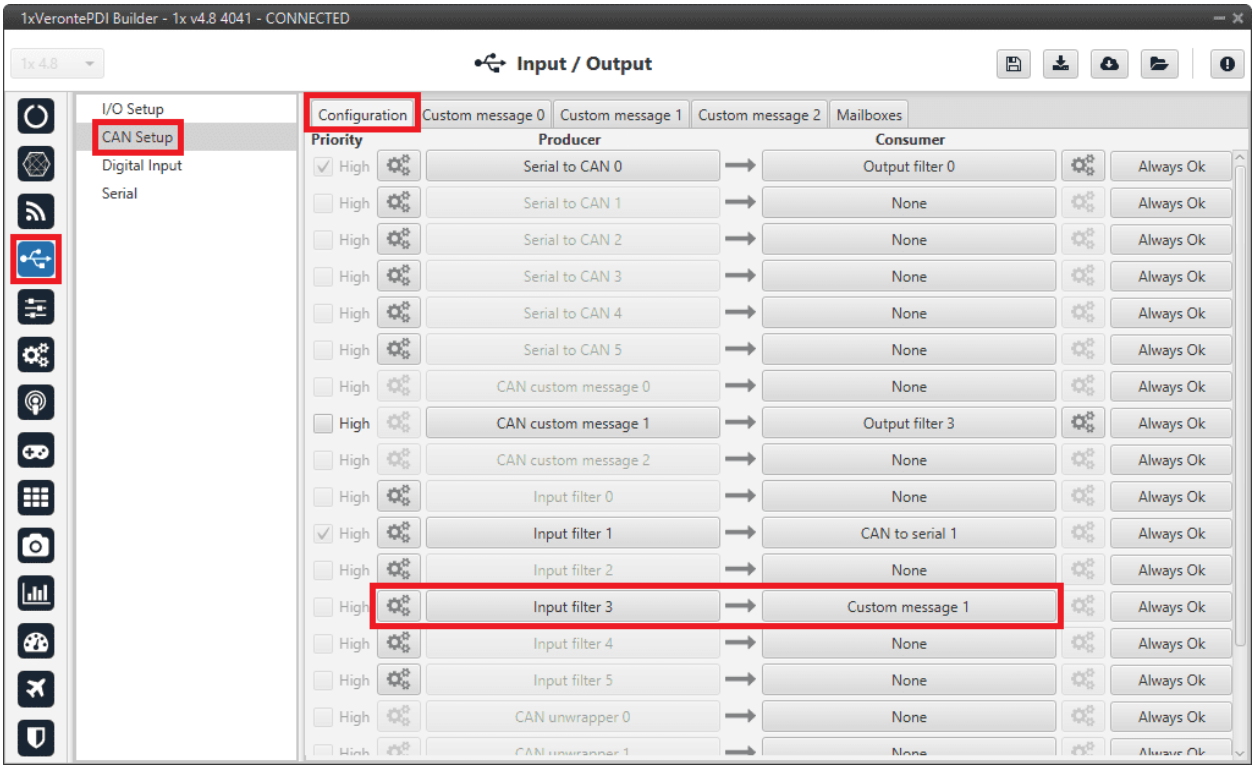


Fig. 211: **1x PDI Builder - CAN Setup configuration**

In addition, according to the send message set in the **MC110 PDI Builder** software, configure the input filter with the following parameters:

- Port: **CAN A**
- **CAN Id 100**
- Mask: **1110000** bin
- Filter type: **Standard**

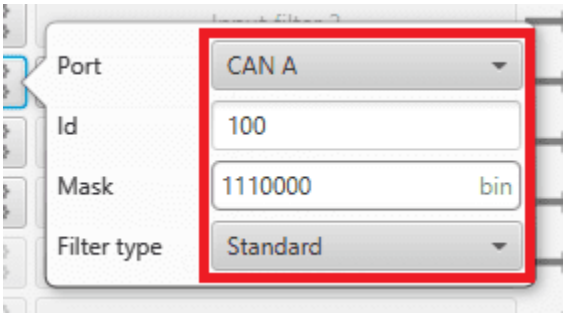


Fig. 212: **1x PDI Builder - Input filter configuration**

4. Go to Input/Output menu → CAN Setup panel → **Custom message 1 tab** (since the input filter has been connected to the *Custom message 1* consumer).

   Users can configure the reception of MC110 variables and store them internally for other uses in the configuration.

To do this, add in **RX** fields the same messages that have been configured in the **MC110 PDI Builder** as **TX Messages**:

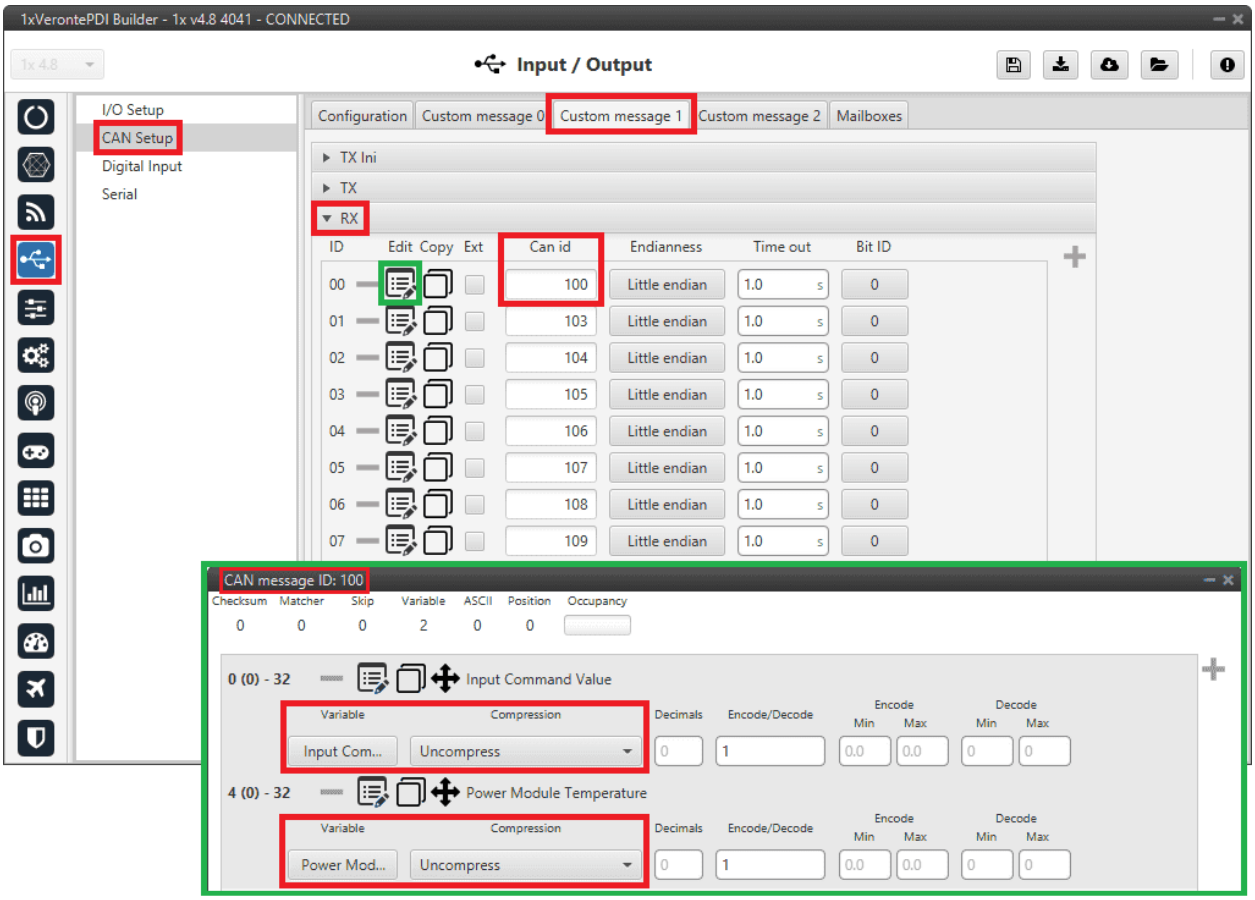


Fig. 213: **1x PDI Builder - Custom message 1 configuration**

For more information on configuring CAN custom messages, refer to the *Custom Messages types - Input/Output* section of this manual.

5. Go to Input/Output menu → CAN Setup panel → **Mailboxes tab**.

   Finally, configure the reception mailboxes.

   In this case, CAN messages with ID **100** and with ID **103 to 109** (8 messages in total) are being sent, which in **binary** is: 0110 0100 and from 0110 0111 to 0110 1101.

   Therefore, **8 mailboxes** are configured with **ID 01100100** and **mask 1110000** in the **CAN A** bus:

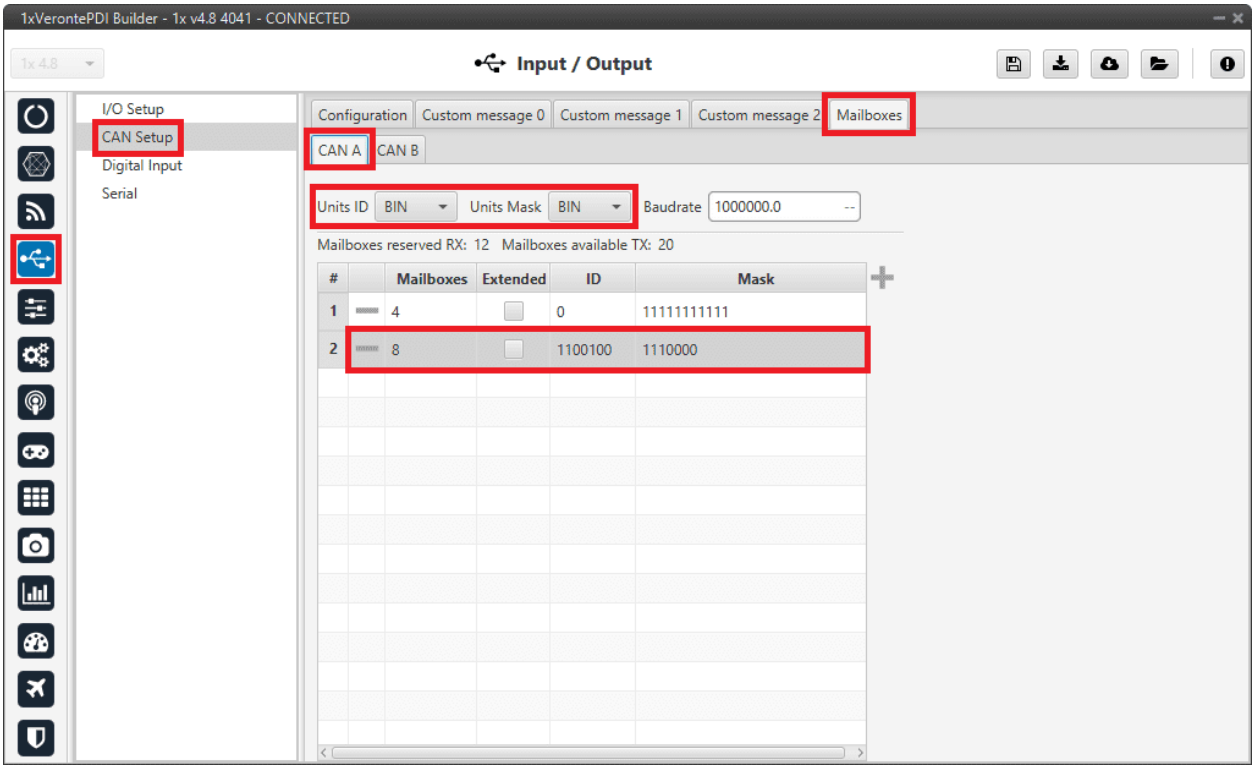


Fig. 214: **1x PDI Builder - Mailboxes configuration**

For more information on mailboxes, see the *Mailboxes (CAN Setup) - Input/Output* section of this manual.

3.7.6.5 Veronte Gimbal

This section explains the configuration required to control and operate **Veronte Gimbal 10z** or **Veronte Gimbal 30z**.

On the one hand, the following diagram illustrates the communication required between a **Veronte Autopilot 1x** and the **Veronte Gimbal to control its movements**:

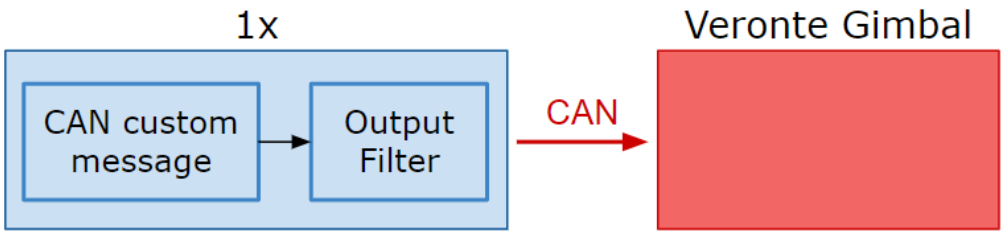


Fig. 215: **Communication diagram 1x → Veronte Gimbal**

On the other hand, to allow communication between **Veronte Autopilot 1x and the video board integrated in the Veronte Gimbal camera** the following connection is required:

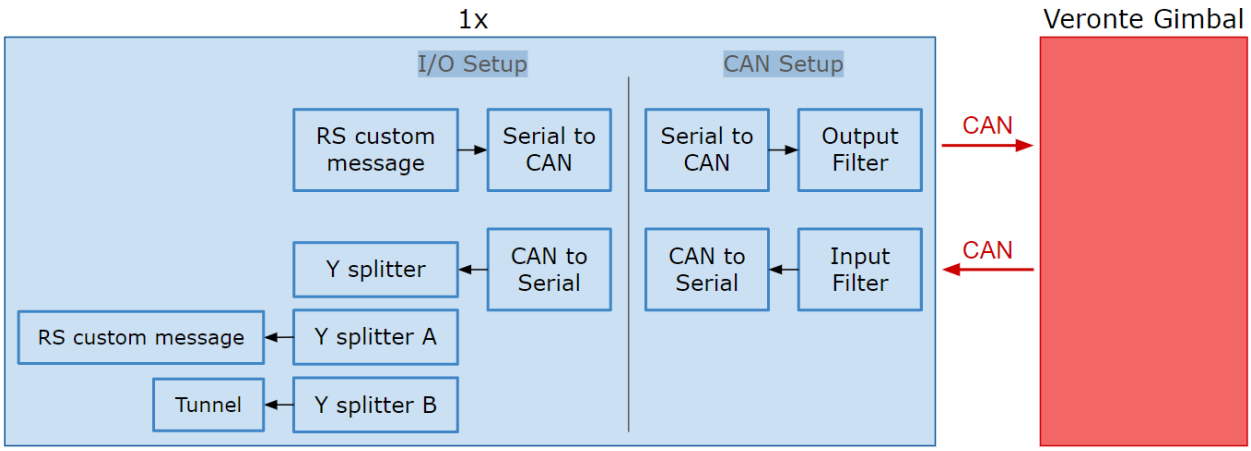


Fig. 216: **Communication diagram 1x ↔ Veronte Gimbal**

In the **1x PDI Builder** software there is already a **template** with the required configuration shown in the diagrams above. Users can access it in the following way:

1. Open **1x PDI Builder** app.
2. Click '**1xVeronte**' option.

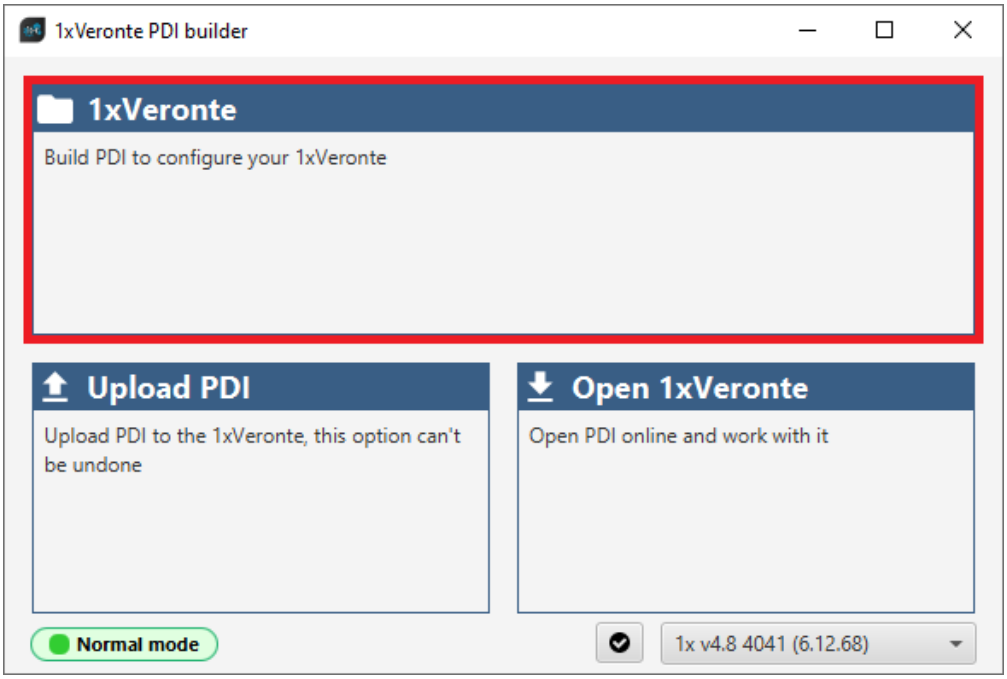


Fig. 217: **Veronte Gimbal - 1xVeronte option**

3. In the initial menu of the app, import a configuration from the repo clicking on  .

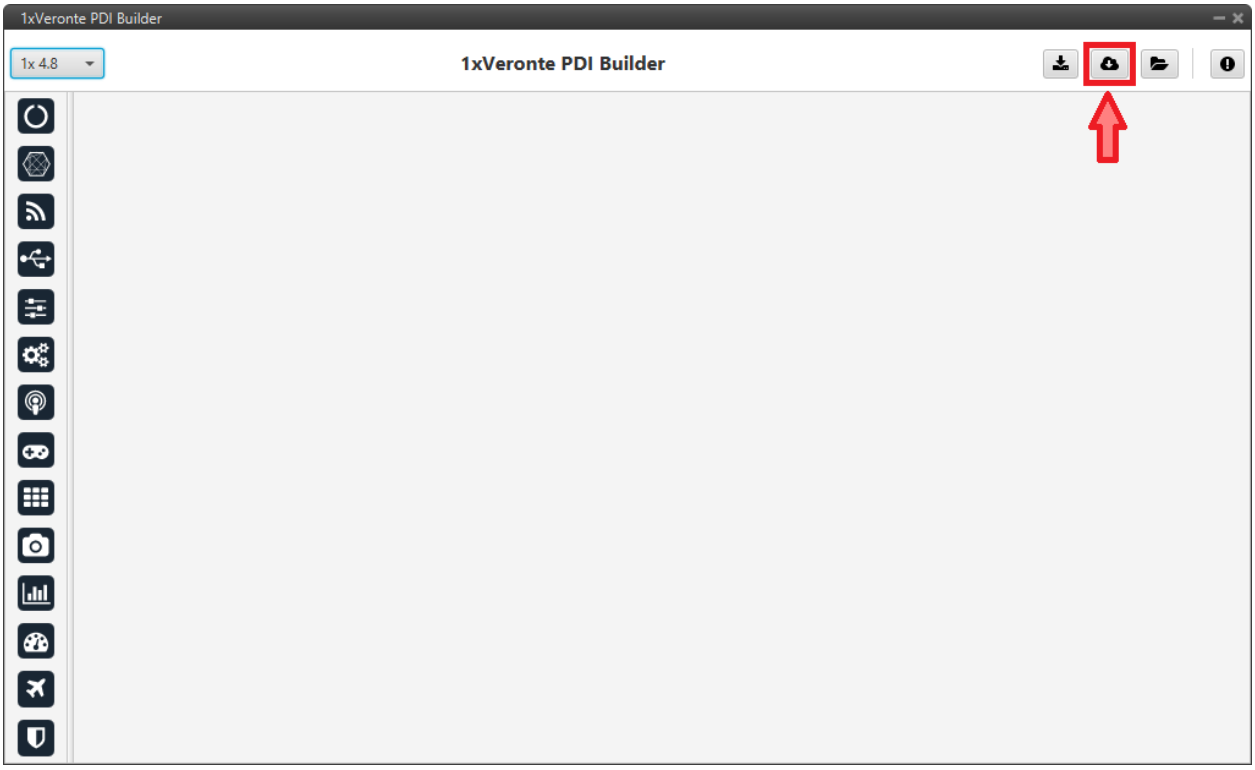


Fig. 218: **Veronte Gimbal - Import from repo**

The following window will appear while the templates are being downloaded:

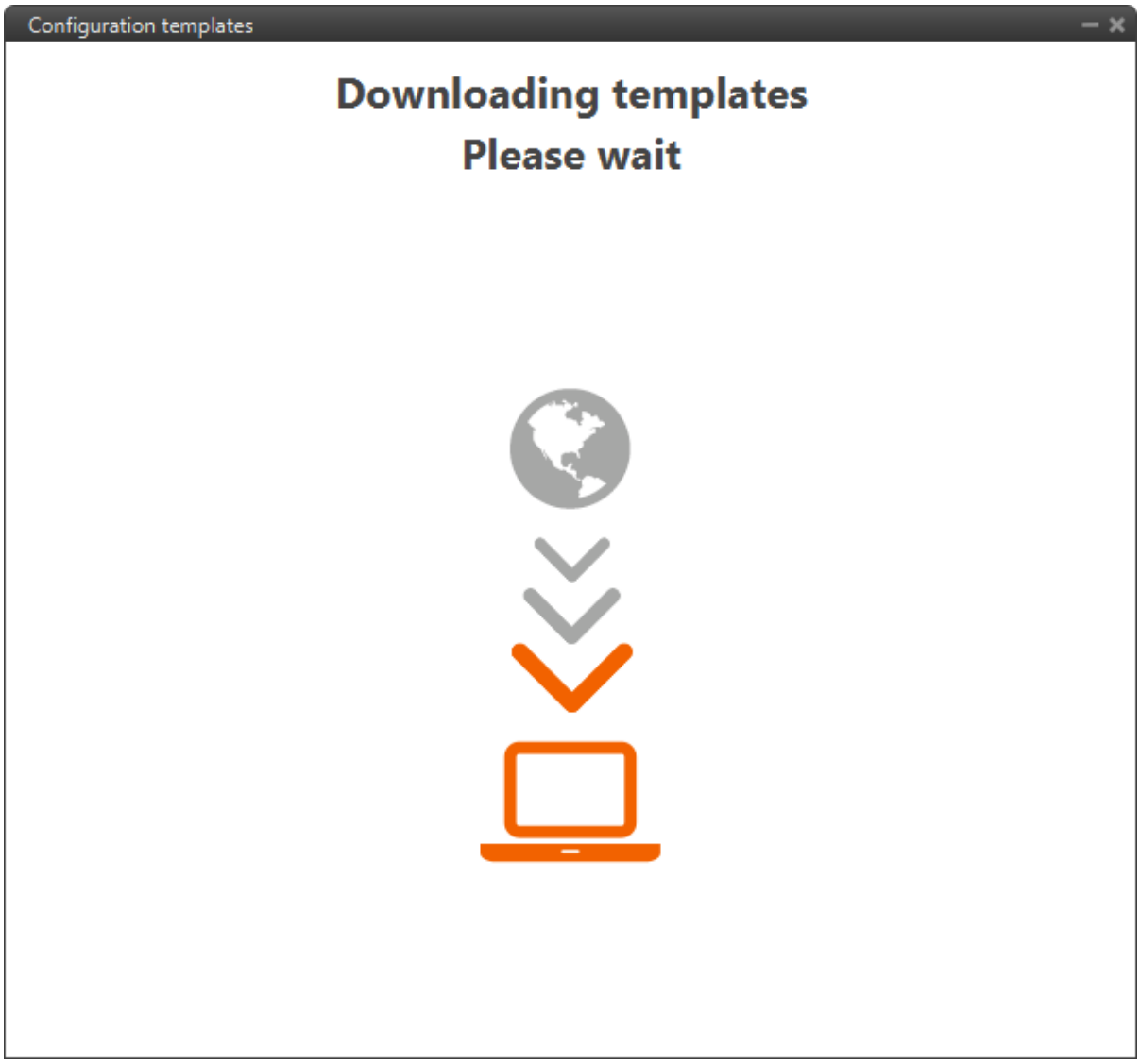


Fig. 219: **Veronte Gimbal - Downloading templates**

4. In the templates menu, select the **Veronte Gimbal** configuration template and press **Import** to import it to the app.

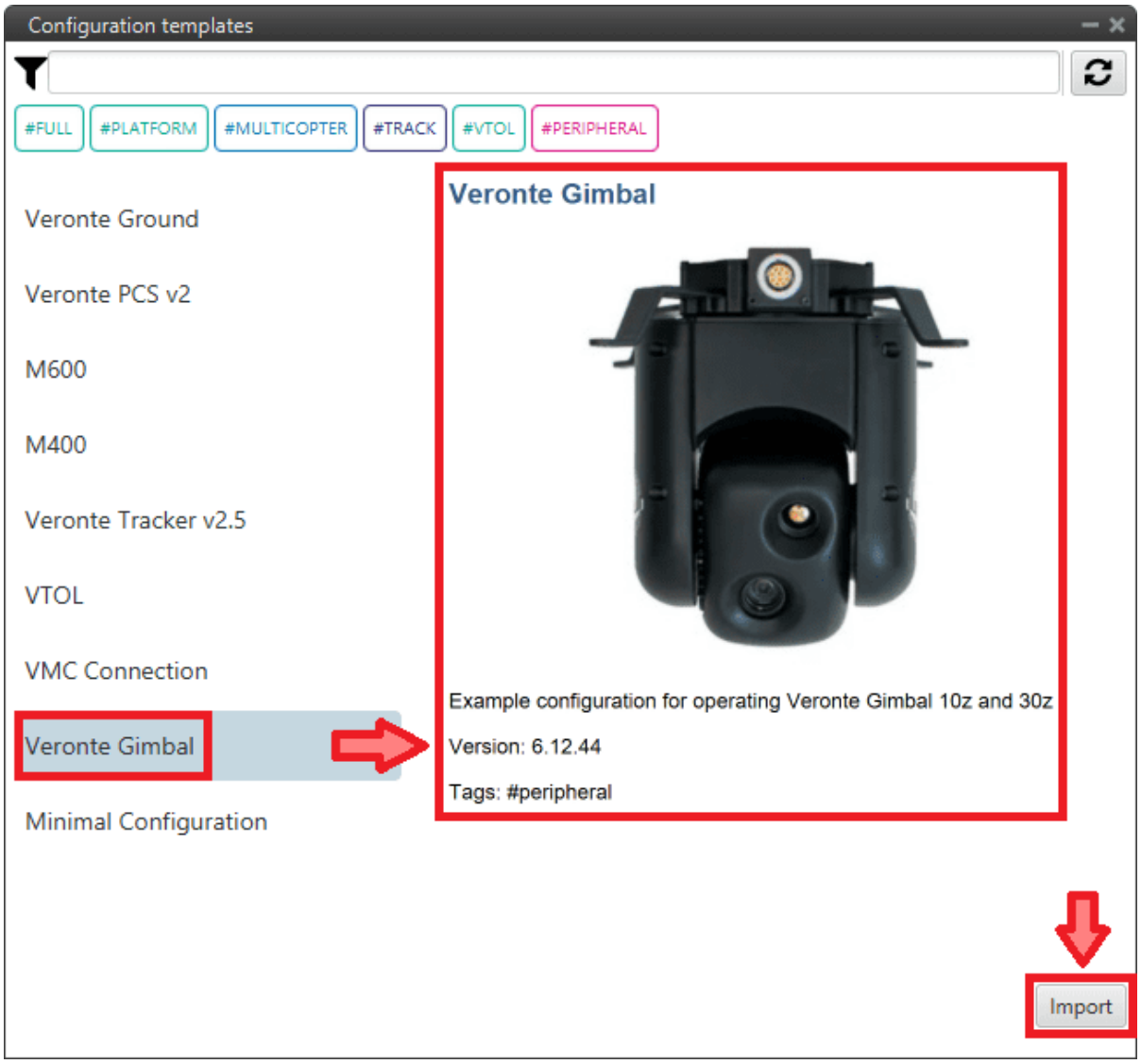


Fig. 220: **Veronte Gimbal - Veronte Gimbal configuration template**

Now, users have to **add** the Veronte Gimbal control aspects of this configuration template **to their own Autopilot 1x configuration**.

3.7.6.5.1 Controlling Veronte Gimbal movement

Concerning the movement control of the Veronte Gimbal, these are the relevant parts of the configuration:

- In the Input/Output menu → CAN Setup panel → **Configuration tab**.

  A **CAN custom message producer** must be connected to an **Output filter consumer**. In this example, *CAN custom message 0* is connected to *Output filter 2* and *CAN A* bus has been chosen.

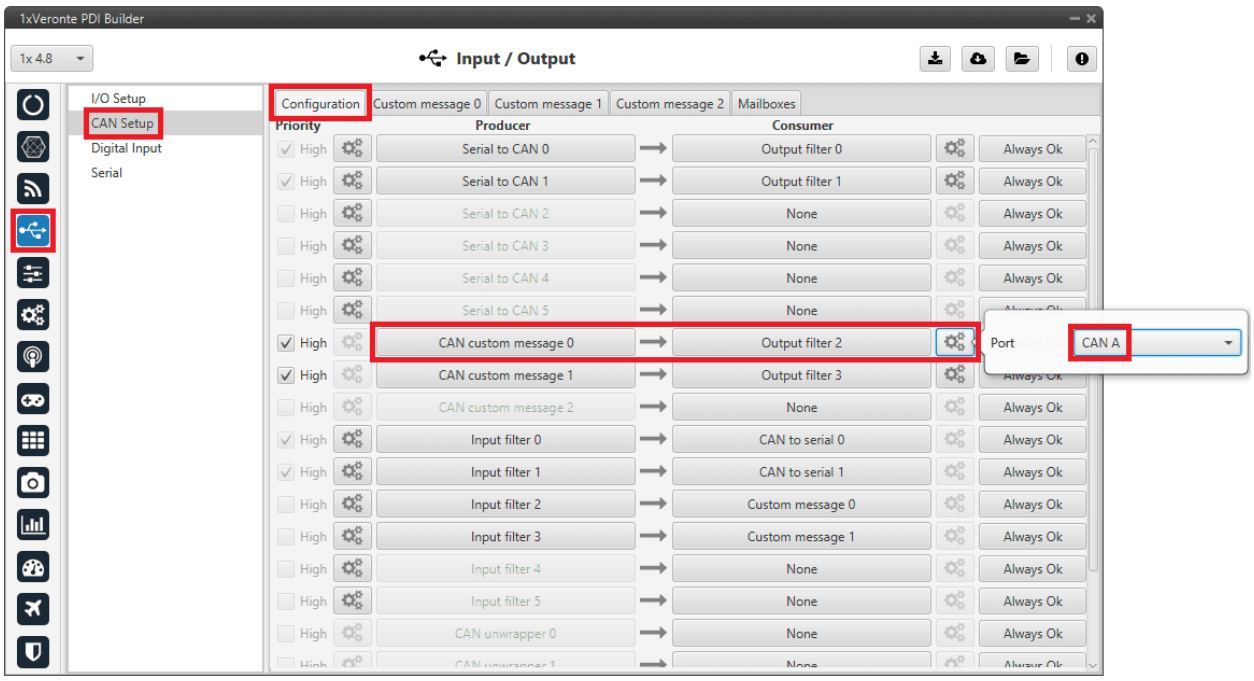


Fig. 221: **Veronte Gimbal - CAN Setup configuration panel**

- In the Input/Output menu → CAN Setup panel → **Custom message 0 tab** (since the producer *CAN custom message 0* has been connected to the output filter).

  The following parameters must be set for 2 CAN messages on **TX** (as they are for tranmission) with CAN ID 0 and 1. More information on the configuration of CAN messages can be found in the *TX/TX Ini Messages (Custom Messages) - Input/Output* section of this manual.

  Warning: These specific CAN IDs are entered because they have to match the ones configured in the Gimbal, which are configured by default with these ids.

  - **Can id**: 0 / 1
  - **Endianness**: Little endian
  - **Period**: 0.01 s

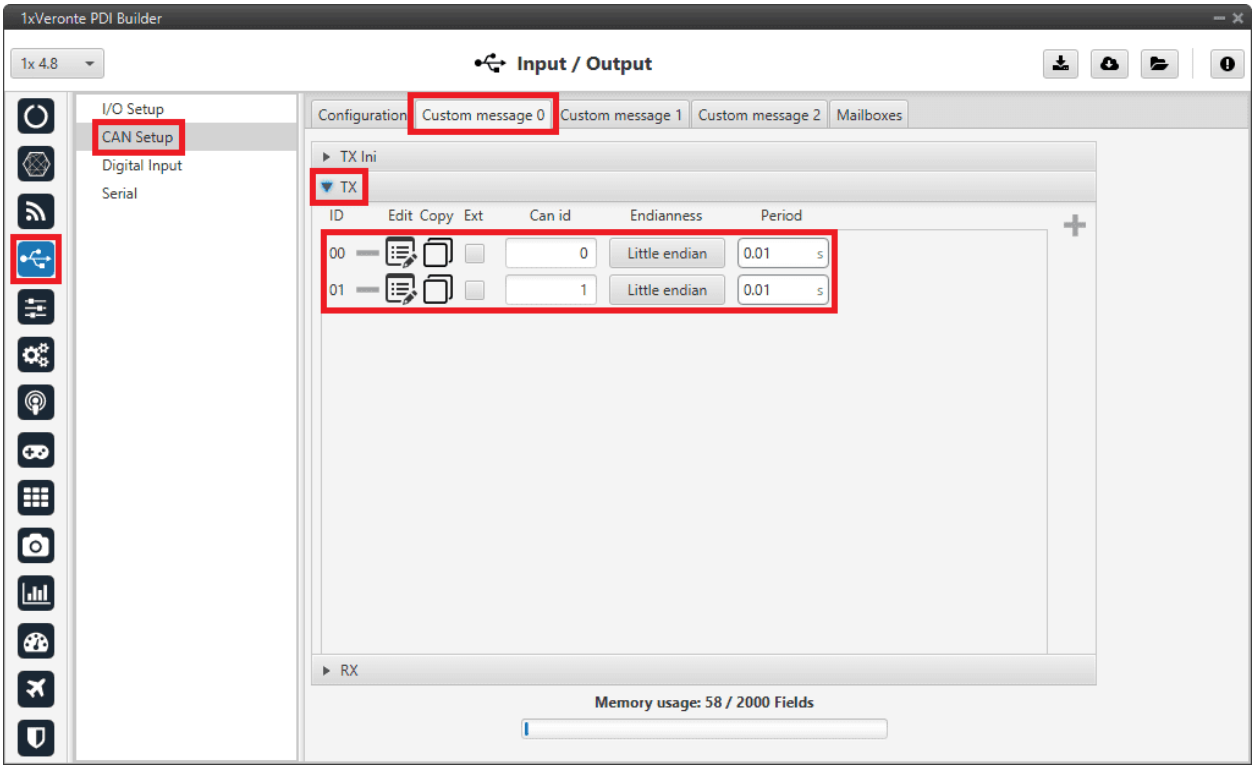


Fig. 222: **Veronte Gimbal - Custom message panel**

Click on  to access their configuration. This is almost the same for both messages but changing the variable:

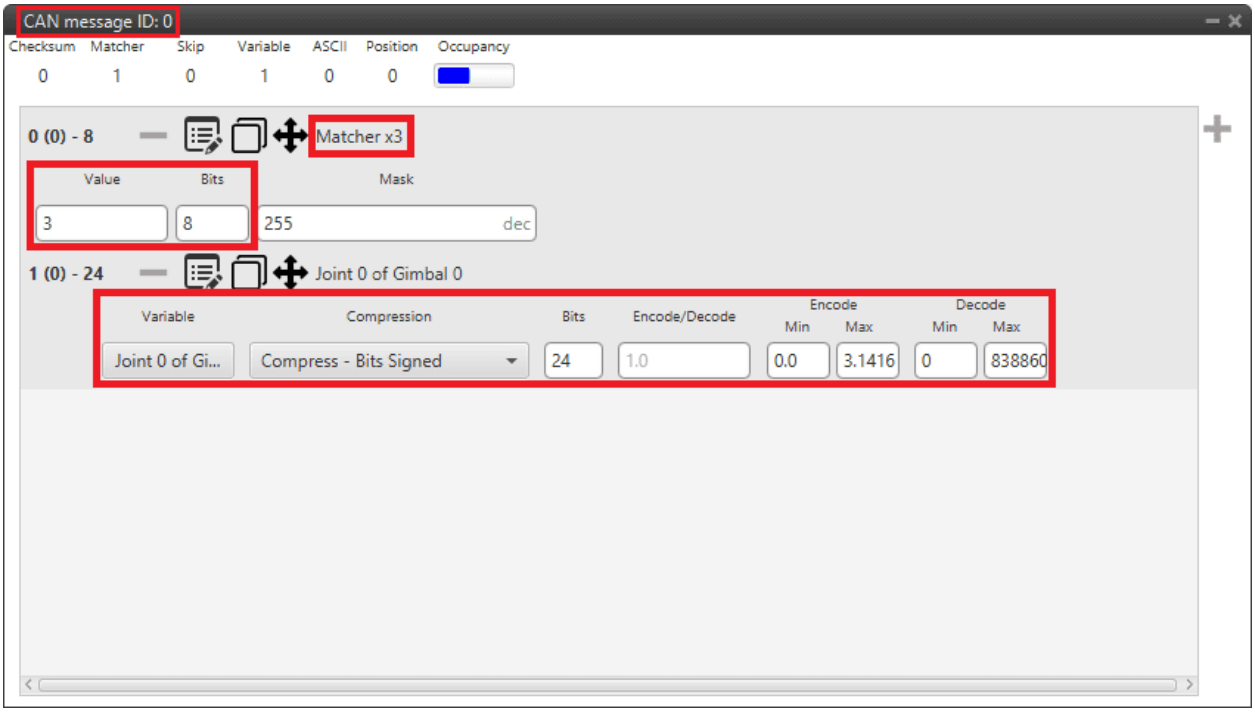


Fig. 223: **Veronte Gimbal - CAN custom message ID 0 configuration**

1. **Matcher**
   - **Value**: 3
   - **Bits**: 8
2. **Variable**
   - **Variable**: Joint 0 of Gimbal 0
   - **Compression**: Compress - Bits Signed
   - **Bits**: 24
   - **Encode - Min/Max**: 0.0/3.1416
   - **Decode - Min/Max**: 0/8388608

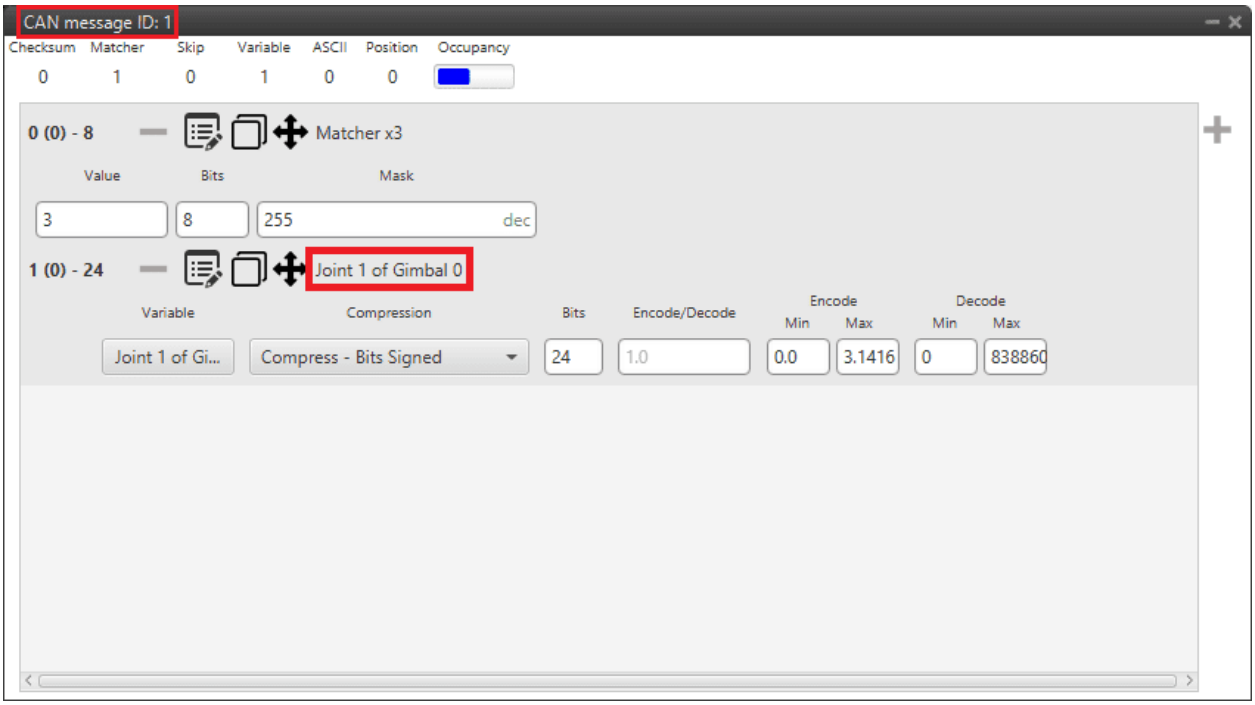


Fig. 224: **Veronte Gimbal - CAN custom message ID 1 configuration**

1. **Matcher**
   - **Value**: 3
   - **Bits**: 8
2. **Variable**
   - **Variable**: Joint 1 of Gimbal 0
   - **Compression**: Compress - Bits Signed
   - **Bits**: 24
   - **Encode - Min/Max**: 0.0/3.1416
   - **Decode - Min/Max**: 0/8388608

3.7.6.5.2 Communication with Veronte Gimbal camera video board

The following are the configuration aspects of the communication with the gimbal camera video board.

3.7.6.5.2.1 CAN commands sent by Autopilot 1x

- In the Input/Output menu → **I/O Setup panel**.

  A **RS custom message producer** is connected to a **Serial to CAN consumer**. In this example, *RS custom message 0* is connected to *Serial to CAN 1*.

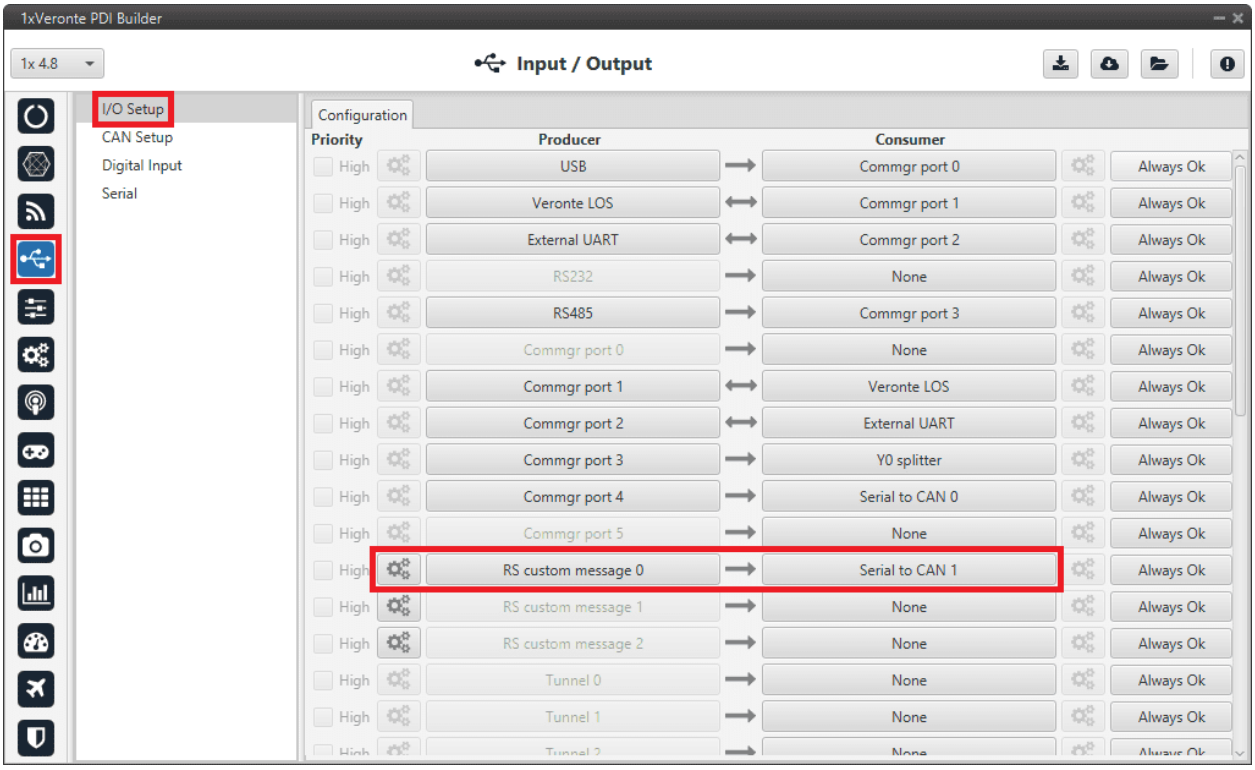


Fig. 225: **Veronte Gimbal - I/O Setup configuration panel**

Click on  to access the RS custom message configuration. It consists of 4 messages for **request variables**, i.e. they request information from the video board of the gimbal camera.

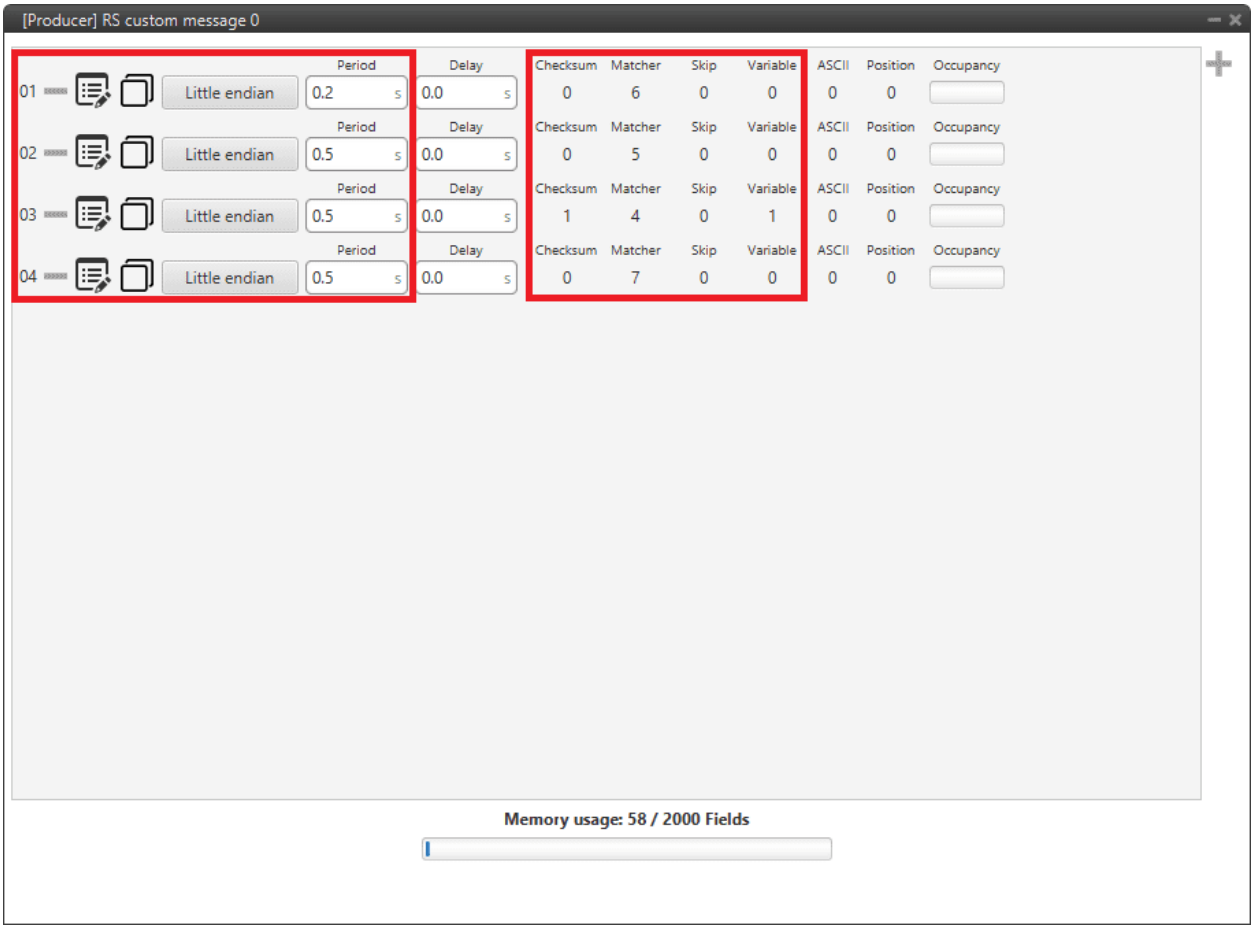


Fig. 226: **Veronte Gimbal - RS producer custom message configuration**

Message	Endianness	Period
01	Little endian	0.2 s
02	Little endian	0.5 s
03	Little endian	0.5 s
04	Little endian	0.5 s

- In the Input/Output menu → CAN Setup panel → **Configuration tab**.

  A **Serial to CAN producer** with **Id 1304** must be connected to an **Output filter consumer**. In this example, *Serial to CAN 1* is connected to *Output filter 1* and *CAN A* bus has been chosen.

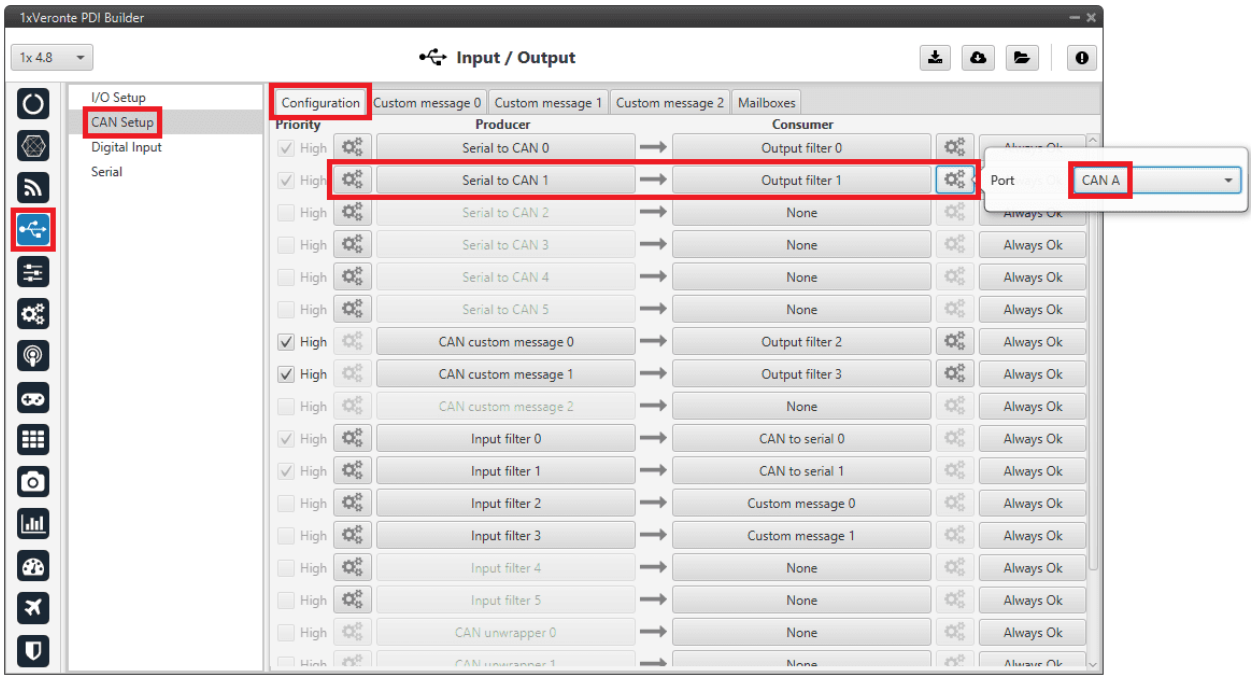


Fig. 227: **Veronte Gimbal - CAN Setup configuration panel**

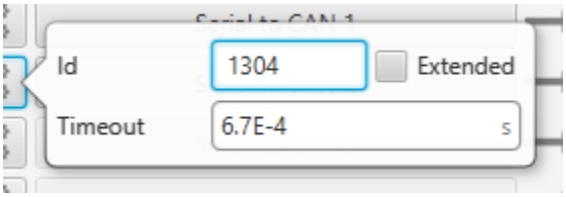


Fig. 228: **Veronte Gimbal - Serial to CAN configuration**

3.7.6.5.2.2 CAN commands received on Autopilot 1x

- In the Input/Output menu → CAN Setup panel → **Configuration tab**.

  An **Input filter producer** with **Id 1303** must be connected to a **CAN to serial consumer**. In this example, *Input filter 1* configured to *CAN A* bus, is connected to *CAN to serial 1*.

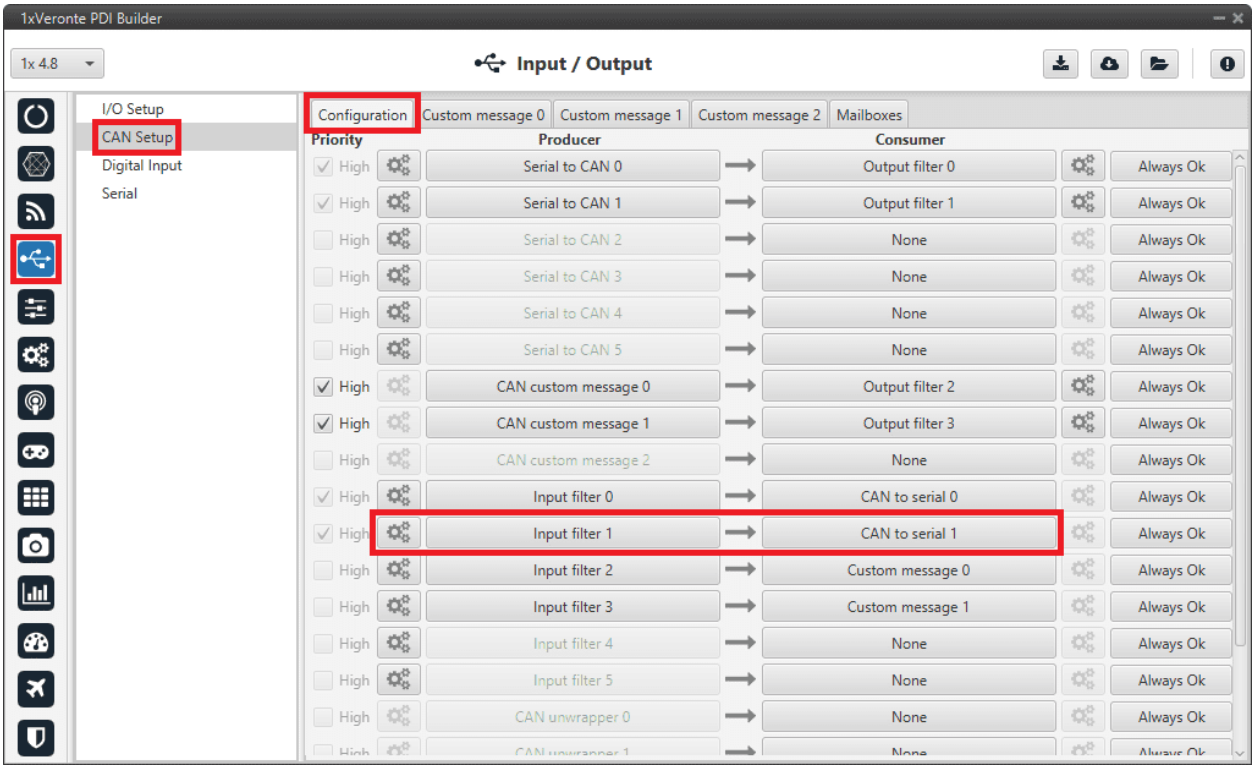


Fig. 229: **Veronte Gimbal - CAN Setup configuration panel**

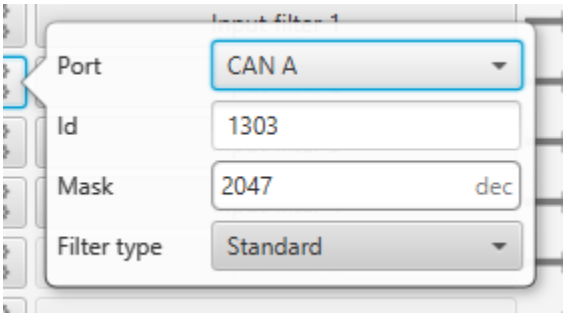


Fig. 230: **Veronte Gimbal - Input filter configuration**

- In the Input/Output menu → CAN Setup panel → **Mailboxes tab**.

  20 reception mailboxes with **ID 1303** are configured in **CAN A** bus (as the input filter has been configured to *CAN A*):

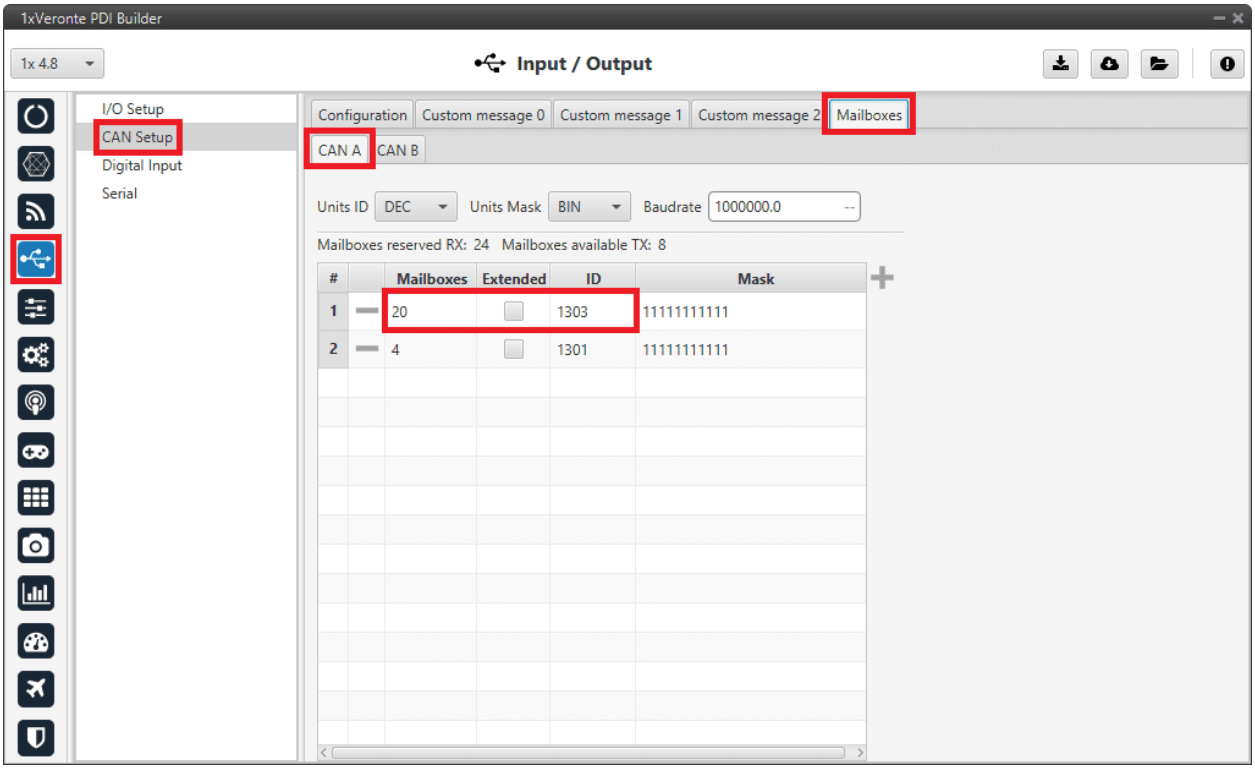


Fig. 231: **Veronte Gimbal - Mailboxes configuration**

- In the Input/Output menu → **I/O Setup panel**.

  A **CAN to serial producer** is connected to a **Y splitter consumer**. Then, a **Y splitter A producer** is connected to a **RS custom message consumer** and a **Y splitter B producer** is connected to a **Tunnel consumer**, configured to **Address 2** (App 2).

  This connection is made in order to read and process the information received from the video board (RS custom message consumer) while sending this information to Veronte Ops. So **Autopilot 1x** is acting as a tunnel between the video board and Veronte Ops.

  In this example, *CAN to serial 1* is connected to *Y1 splitter*, *Y1 splitter A* to *RS custom message 2* and *Y1 splitter B* to *Tunnel 0*.

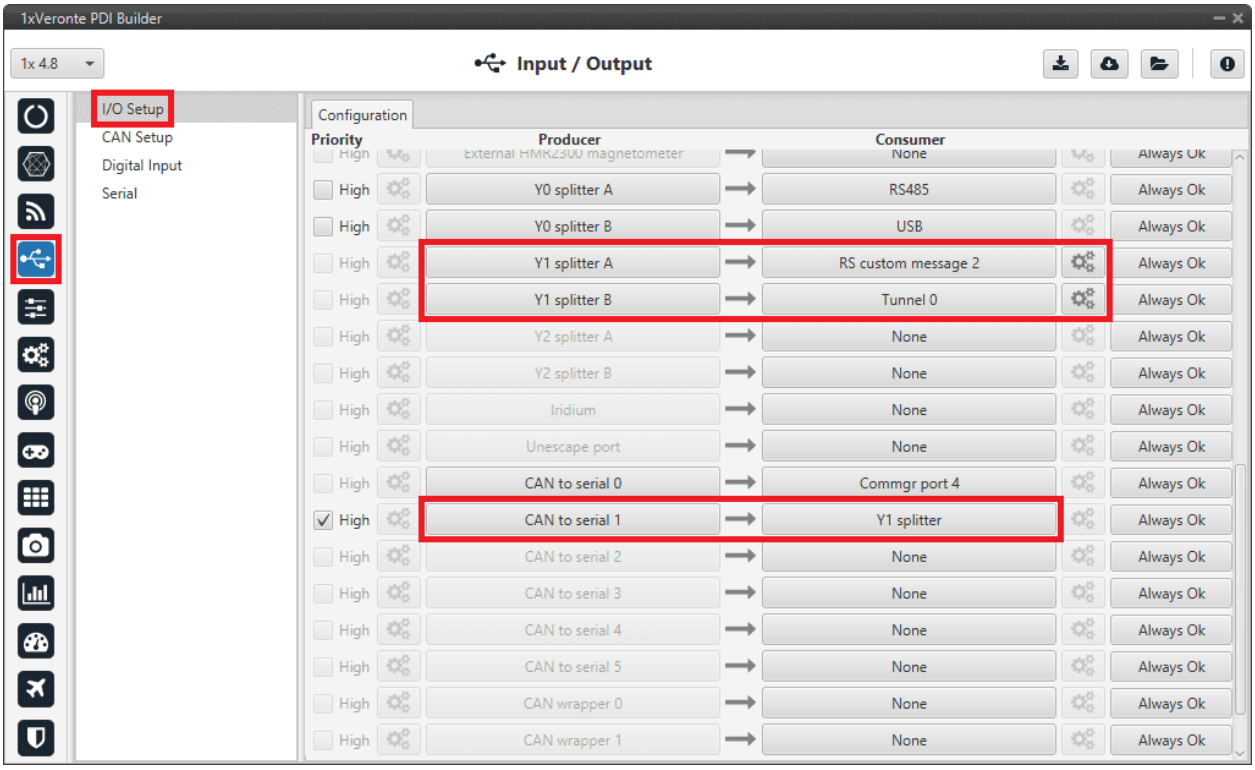


Fig. 232: **Veronte Gimbal - CAN Setup configuration panel**

Click on  to access the RS custom message configuration. It consists of 4 messages to read the video board information from the gimbal camera that has been previously requested.

Important: They must be configured as **Little endian**.

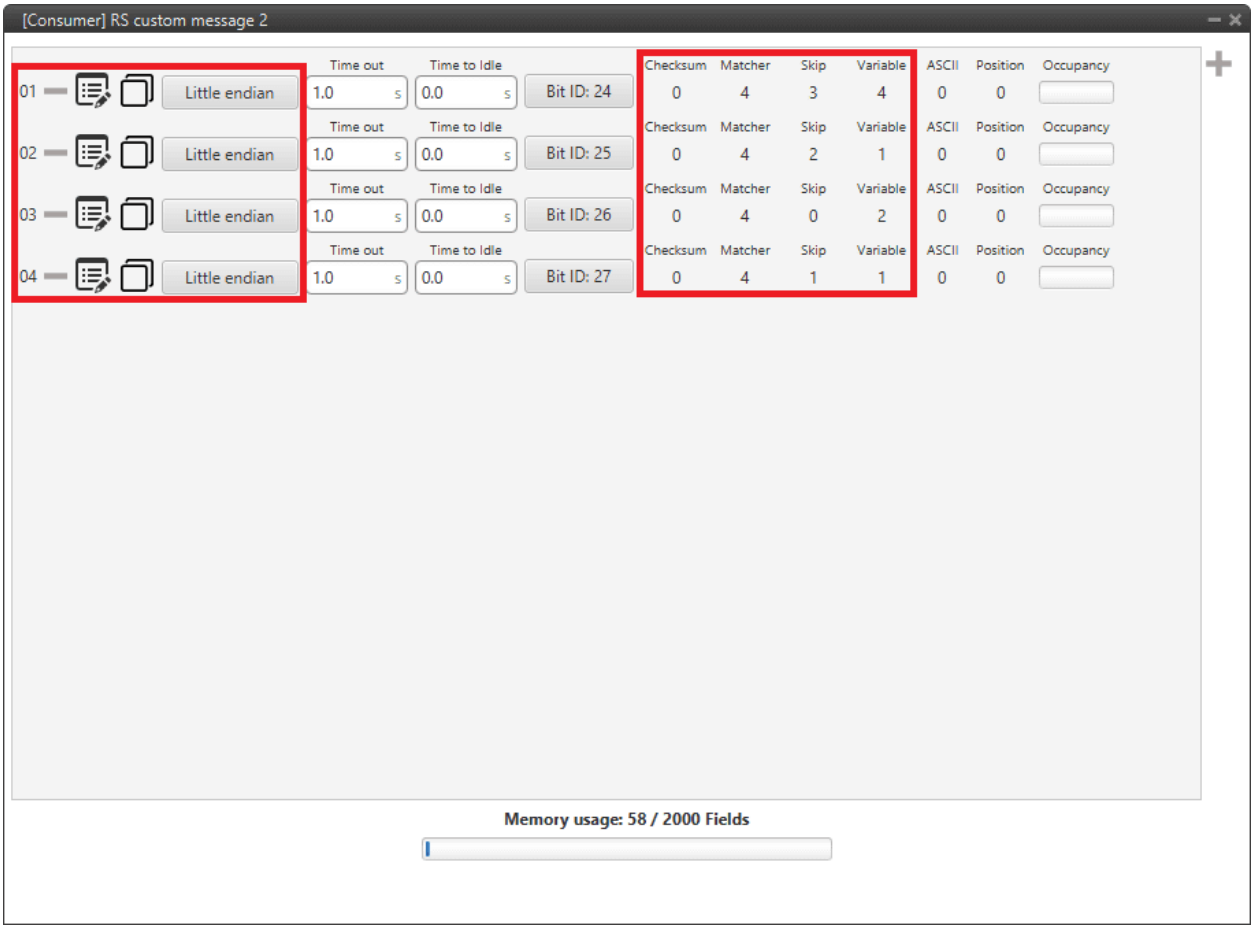


Fig. 233: **Veronte Gimbal - RS consumer custom message configuration**

Finally, click on  to access the Tunnel configuration:

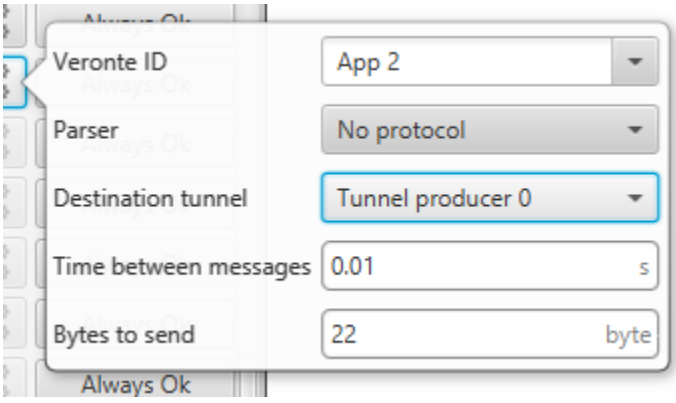


Fig. 234: **Veronte Gimbal - Tunnel configuration**

- **Veronte ID**: App 2
- **Parser**: No protocol
- **Destination tunnel**: Tunnel producer 0

- **Time between messages**: 0.01 s
- **Bytes to send**: 22 byte

3.7.6.5.2.3 Gimbal block program

Finally, in the **Block Programs menu**, a **Gimbal** program has also been created to allow a correct communication between **Veronte Autopilot 1x and the video board integrated in the Veronte Gimbal camera**.

Warning: Users must add it to their own configuration in exactly the same way.

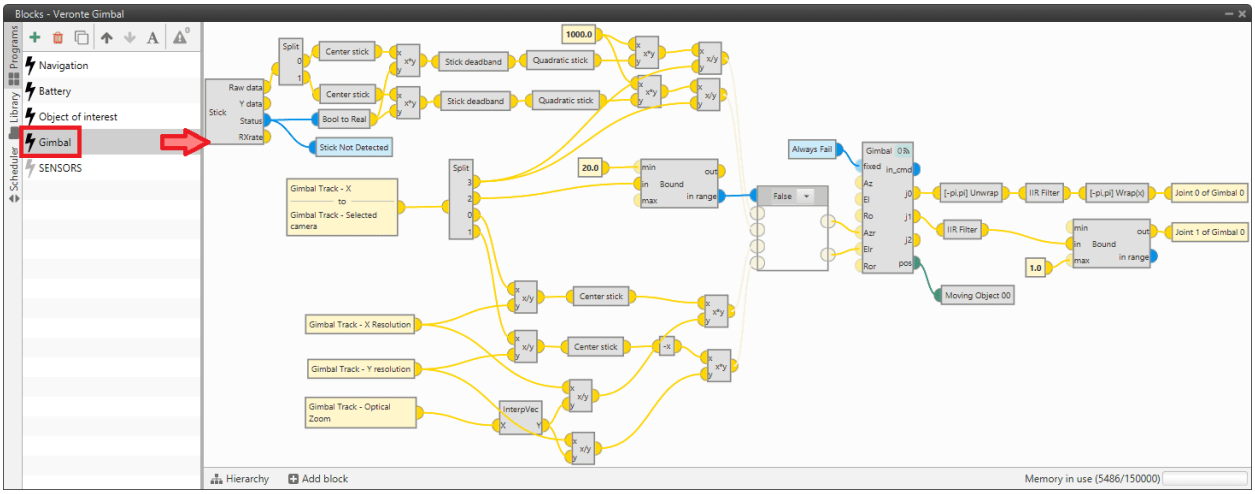


Fig. 235: **Veronte Gimbal - Block program**

3.7.6.6 VSE (Veronte Stick Expander)

To configure the VSE in **1x PDI Builder** it is only needed to follow the steps explained in the **Ground unit configuration** of the *General case - PPM Stick integration example*.

Important: In the **step 1** of that explanation, there is already a transmitter configured with the required VSE configuration, users will find it as **Brand: Embention** and **Model: Stick Expander**.

The **number of channels** configured here must match those set in the **VSE application**. For more information on this, refer to the Channels section of the **Stick Hardware Manual**.

Furthermore, as the number of channels is modified, the **Brand** name will change to **Customize**.

CHAPTER

FOUR

TROUBLESHOOTING

4.1 Communication lost with internal Digi radio

Most of the time, the **communication** between **Autopilot 1x** and **Digi radio** is **lost** due to a change in its **baudrate**.

In **1x PDI Builder** it is set to **115200** by default, however, in **Digi radios** the factory default baudrate at reset is **9600**.

To recover communication, try changing the baudrate on one of them to match.

1. Go to Input/Output menu → Serial panel → **Veronte LOS tab**.

   Set the **Baudrate** on Veronte LOS to **9600**.

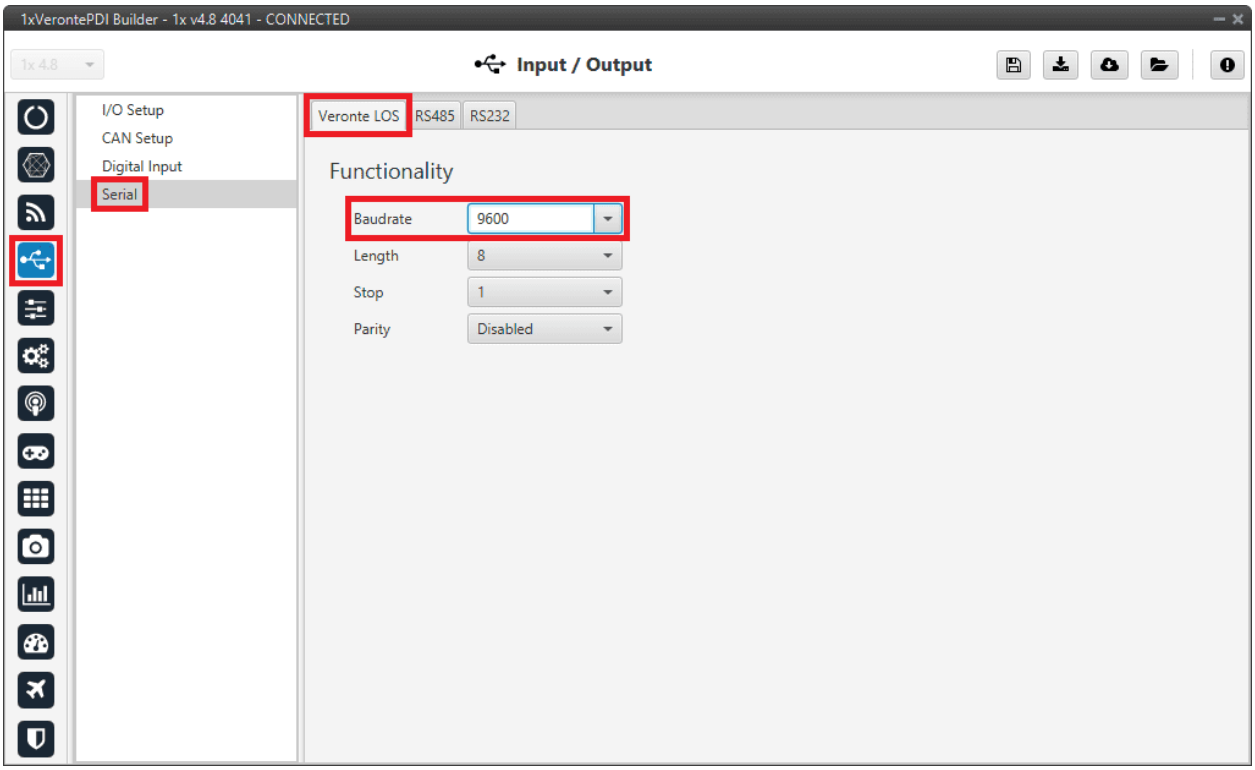


Fig. 1: **Veronte LOS baudrate**

2. Check the steps described in the *Digi internal radio - Integration examples* section to see if the module is now **detected** in **XCTU software**.

Then, if desired, the user can change the radio baudrate to 115200 and after that also change it for **Veronte Autopilot 1x**.

4.2 Debug serial messages transmission

To check that the transmission of serial messages is being carried out correctly, the user can view what is being sent in the 1x PDI Calibration software hyperterminal. To do this:

4.2.1 1x PDI Builder side

1. Go to Input/Output menu → **I/O Setup panel**.

   Connect the **RS custom message producer** (where the message is configured) to a **Tunnel** with **Address 2** (App 2). In this case, the message is configured in the *RS custom message 0* producer and sent through *Tunnel 0*.

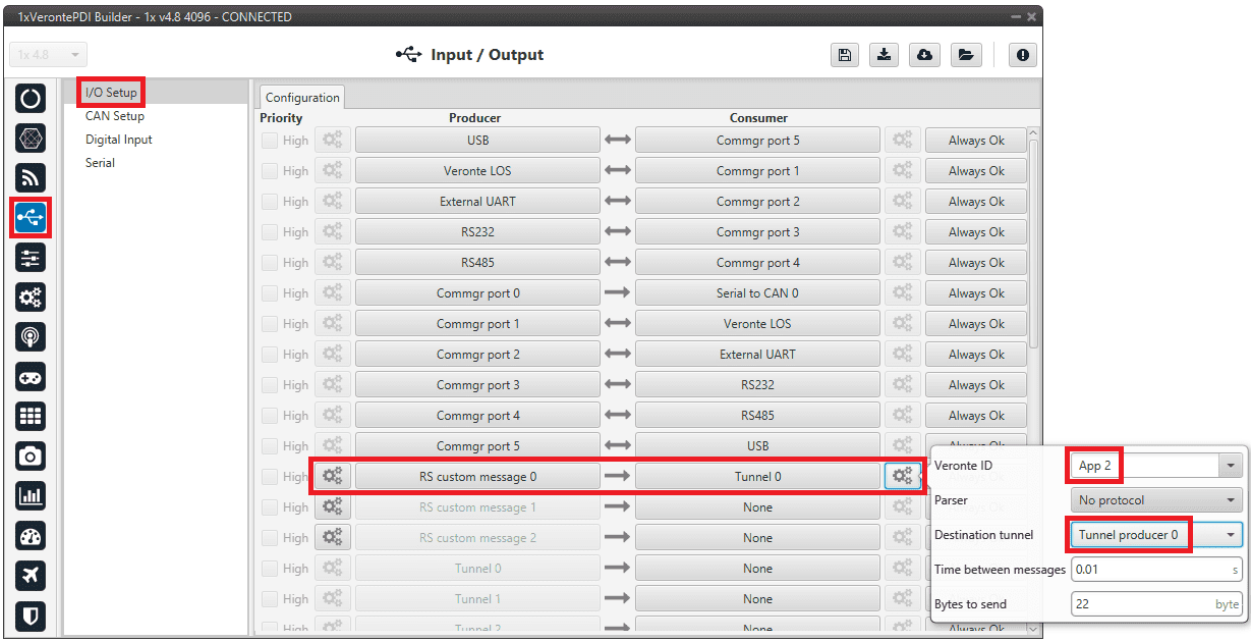


Fig. 2: **RS custom message → Tunnel**

4.2.2 1x PDI Calibration side

2. Go to **Terminal tab**.

   Click on **Agree**:

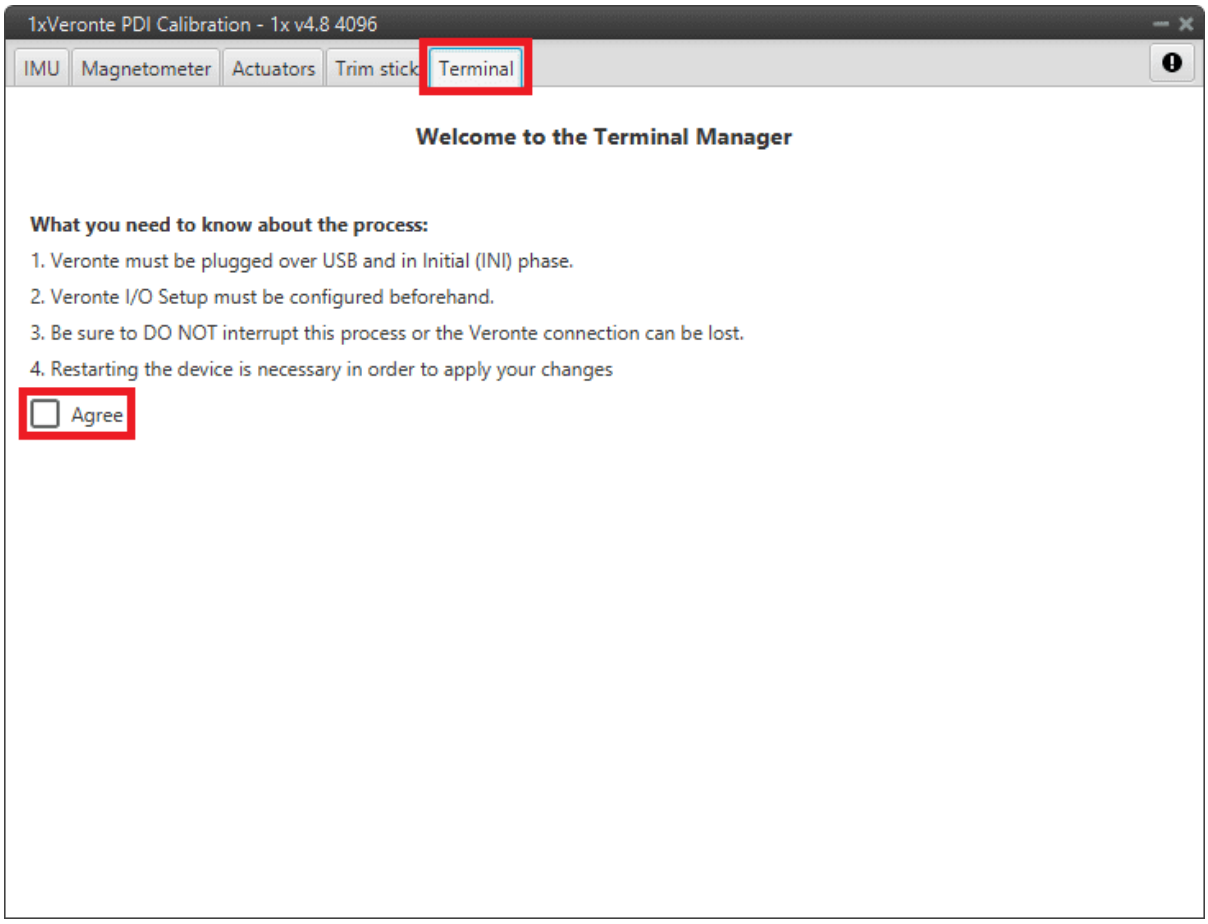


Fig. 3: **Terminal tab**

3. Next, select the **Tunnel 1** (this corresponds to the *Tunnel 0* that has been configured in the **1x PDI Builder**, as the numbering here starts at 1 and not 0) and click on **Launch**:

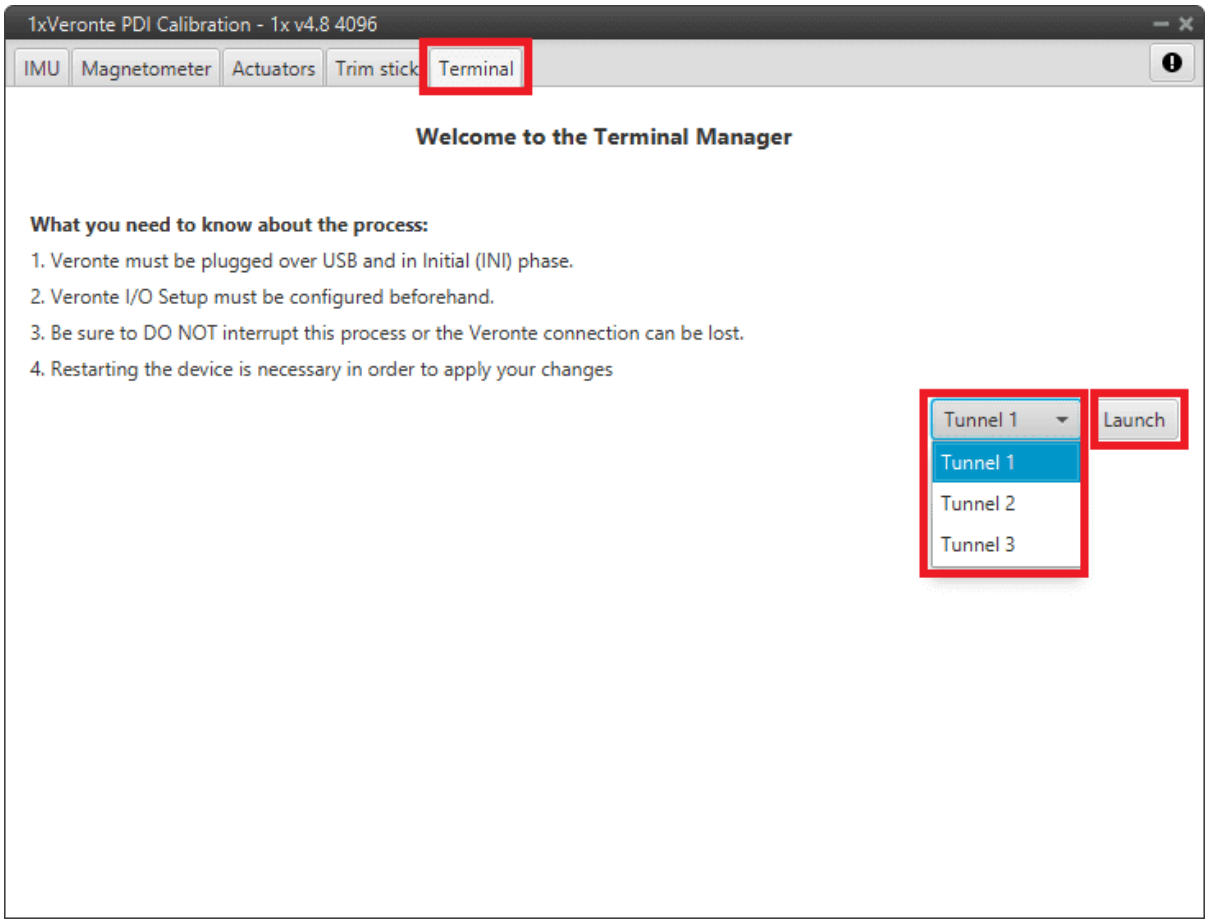


Fig. 4: **Terminal tab - Tunnel selected**

The tunnel console should open and the user will be able to view the message being sent:

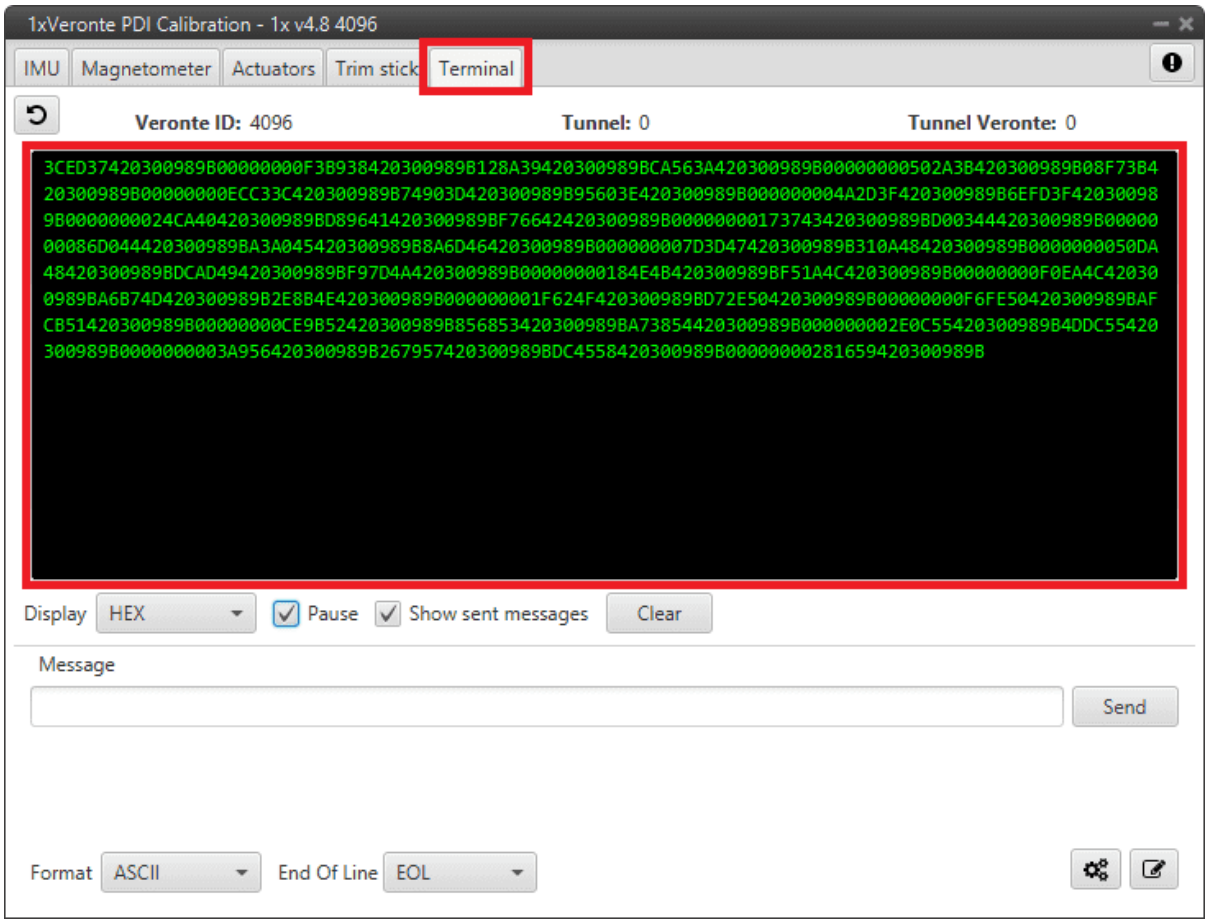


Fig. 5: **Tunnel console**

For more information on the **Terminal configuration**, please refer to the Terminal section of the **1x PDI Calibration** user manual.

4.3 Maintenance mode

The user can simply enter maintenance mode via **1x PDI Builder** by clicking on the "Normal mode" button in the initial menu. In addition, exiting maitenance mode is the same process.

Below is an example of how to do this:

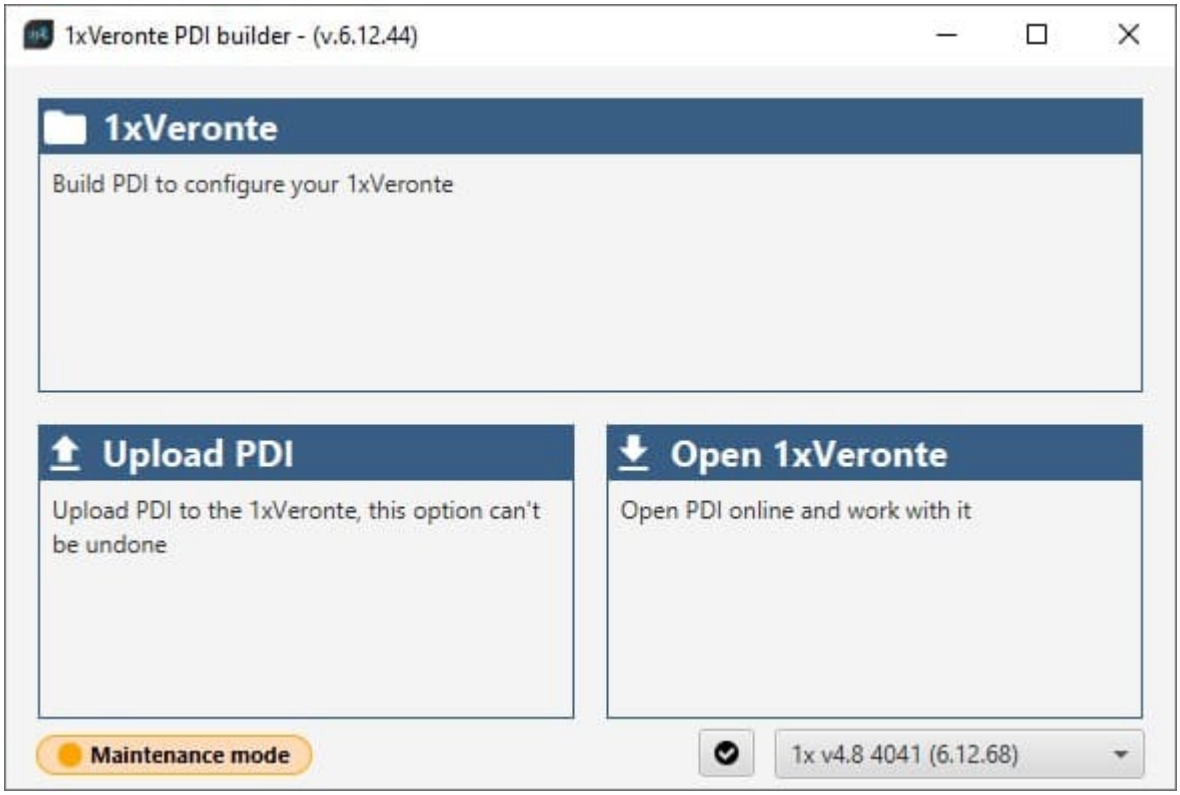


Fig. 6: **Enter/Exit maintenance mode**

4.4 Maintenance mode (loaded with errors)

The following error message may appear when trying to save a change or import a configuration.

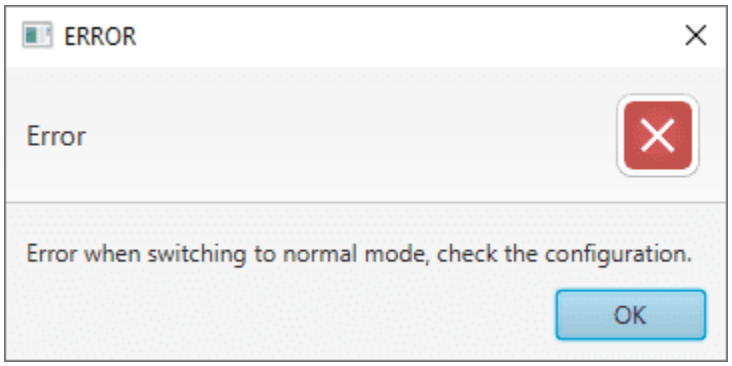


Fig. 7: **Error message**

Therefore, Veronte Autopilot 1x will be in 'Maintenance mode (loaded with errors)':

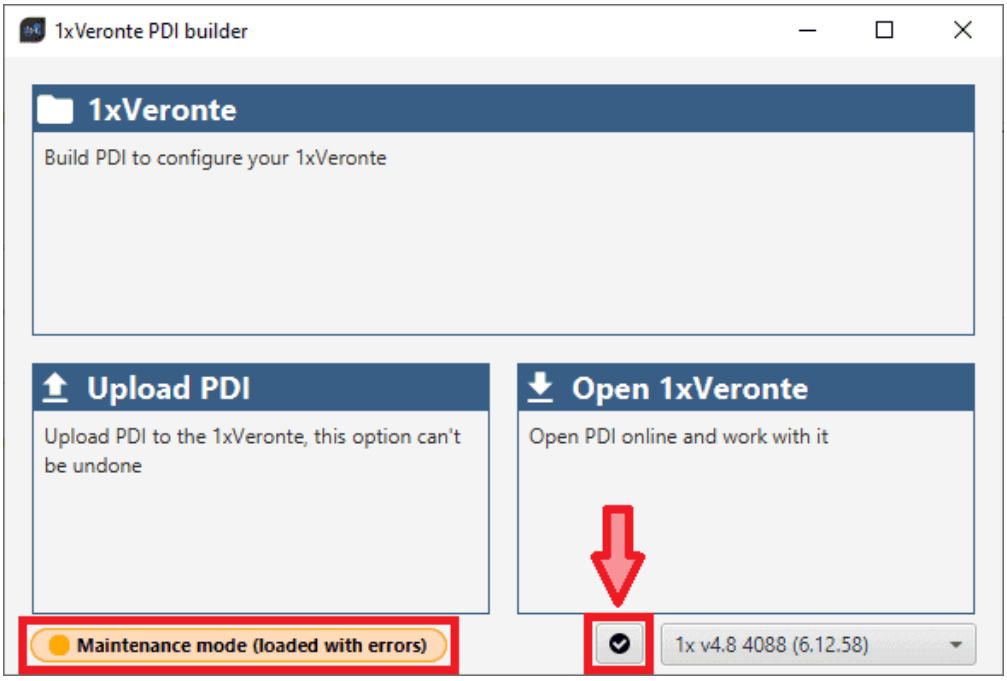


Fig. 8: **Maintenance mode (loaded with errors)**

To check what the source of the problem is, the user can simply click on the **PDI Error button**  , which will show what the PDI Error is:

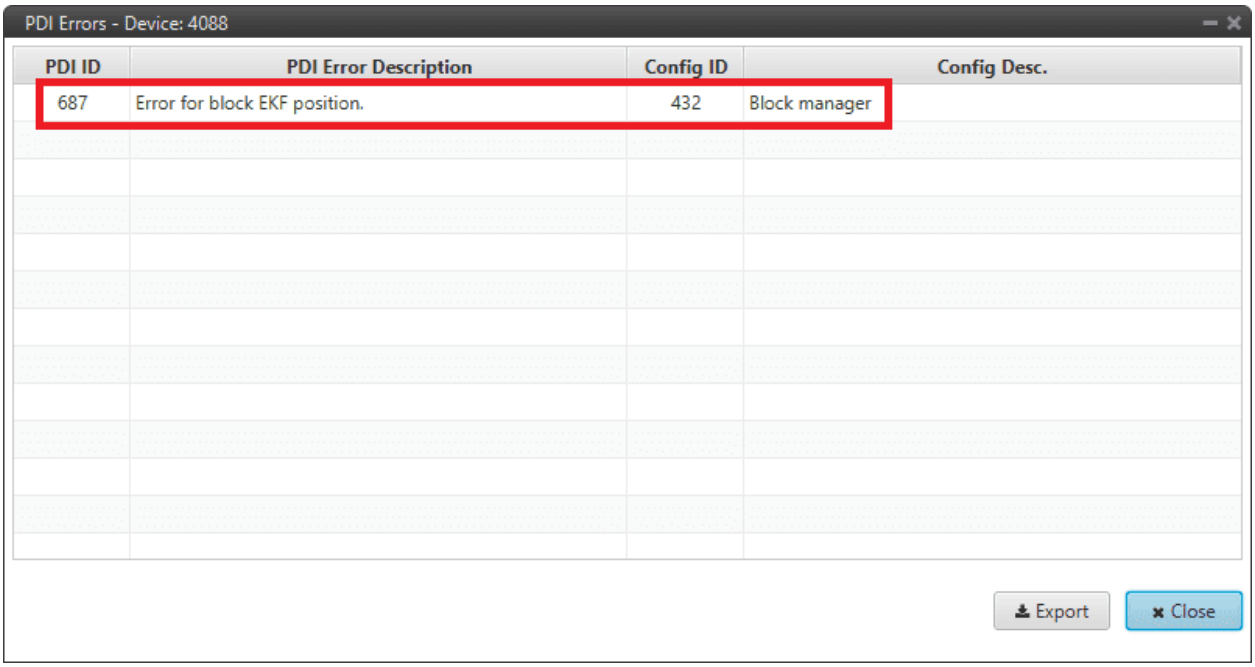


Fig. 9: **Maintenance mode (loaded with errors) - PDI Errors panel**

- **PDI ID**: ID of the PDI Error.
- **PDI Error Description**: Description of this PDI Error. A list of all PDI Errors can also be accessed in the List

of PDI errors section of the **1x Software Manual**.

- **Config ID**: ID of the configurable (.xml file) containing the data in which the PDI Error has been caused.
- **Config Description**: Description of the configurable (.xml file) containing the data in which the PDI Error has been caused.

Clicking the **Export** button will export a `.csv` file with the same information shown in this PDI Errors panel.

This is useful while the configuration is in progress, however, if the user encounters this situation during the operation, it is also possible to look up the cause of the PDI Error directly on the **Platform panel** of **Veronte Ops**. For more information about this panel, see Platform panel section of the **Veronte Ops** user manual.

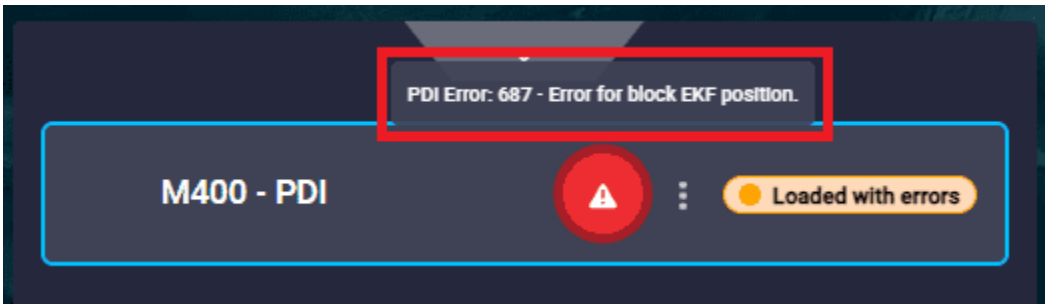


Fig. 10: **PDI Error - Veronte Ops**

Then, it is possible to access the Autopilot 1x configuration to fix this error.

Tip: If the PDI error is related to a **migration**, it is usually caused by the **selection of accelerometer, gyroscope and magnetometer sensors**.

In addition, a list of all PDI Errors can be accessed in the List of PDI errors section of the **1x Software Manual**.

4.5 Migrate configuration

Warning: When performing automatic migration from a previous version to the current version of the software, errors may occur.

It is then the responsibility of the user to check the subsequent result.

4.6 Radios paired but 1x air unit not showing

If the radios of both Autopilots 1x, air and ground units, are paired but the 1x air unit does not appear connected in **Veronte Link**, it may be because the **routing on the 1x ground unit is not correctly configured**. Therefore, users should check the **Ports** configuration on the 1x **ground unit**. To do this:

Go to Communications menu → **Ports panel**, and the routing of **address 2** (address by which Autopilots 1x ground and air units **communicates with all Veronte applications**, such as Veronte Link) must be set to the port that is connected to the USB (PC connection), in this case it is **PORT 0**.

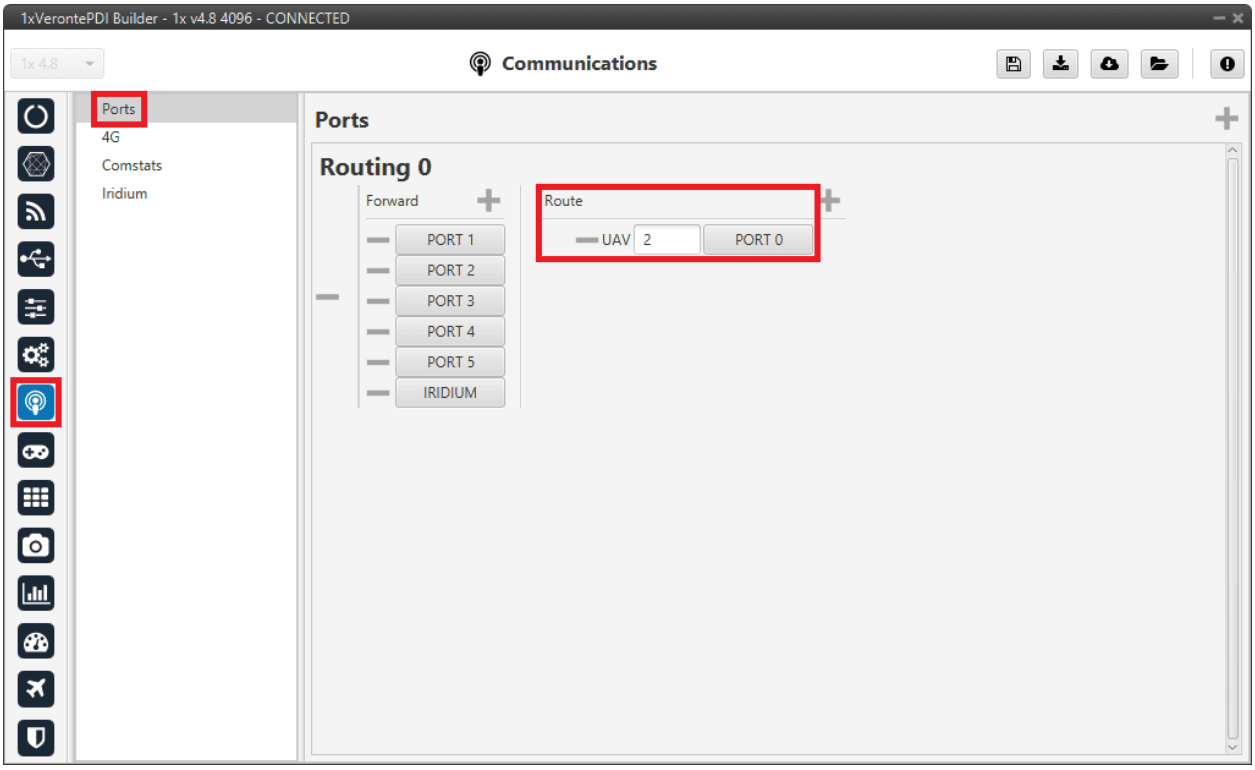


Fig. 11: **1x ground unit - Ports configuration**

4.7 Reducing GNC Task frequency

400 Hz is the **maximum** possible **frequency**, but can only be used in **simple configurations**, in other cases it is advisable to **reduce** it to **250-300 Hz**.

To find out if the frequency needs to be reduced in the user configuration, check the **GNC Task Average CPU Ratio variable**.

For **correct operation**, this variable should be at approximately **60-70%**. If it reports a **higher value**, the **frequency must be lowered**.

4.8 Trajectory Overshoot

If the user observes significant **meandeling** or overshoot in the mission path, this can be reduced by modifying the gains of the guidance PIDs:

- **Reducing** the **proportional** gain.
- Ensure that the **integral** gain is **0**.

Guidance error accumulates and leads to increasing overshoot, as can be seen in the following example:

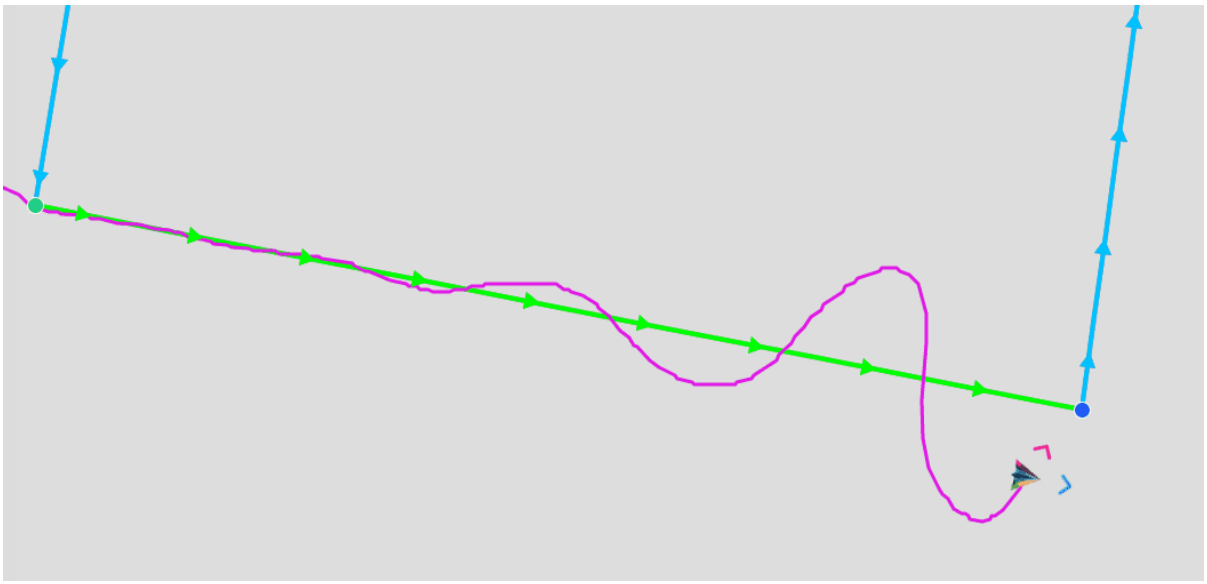


Fig. 12: **Trajectory overshoot**

CHAPTER

FIVE

FAQ

5.1 How to calculate a mask

This section attaches a python program that allows users to easily calculate their mask in standard or extended frame format by simply entering the CAN Ids as a **vector**. In addition, this program also converts each Id entered into binary.

maskCalculator.py

An example of the execution of this program is shown below:

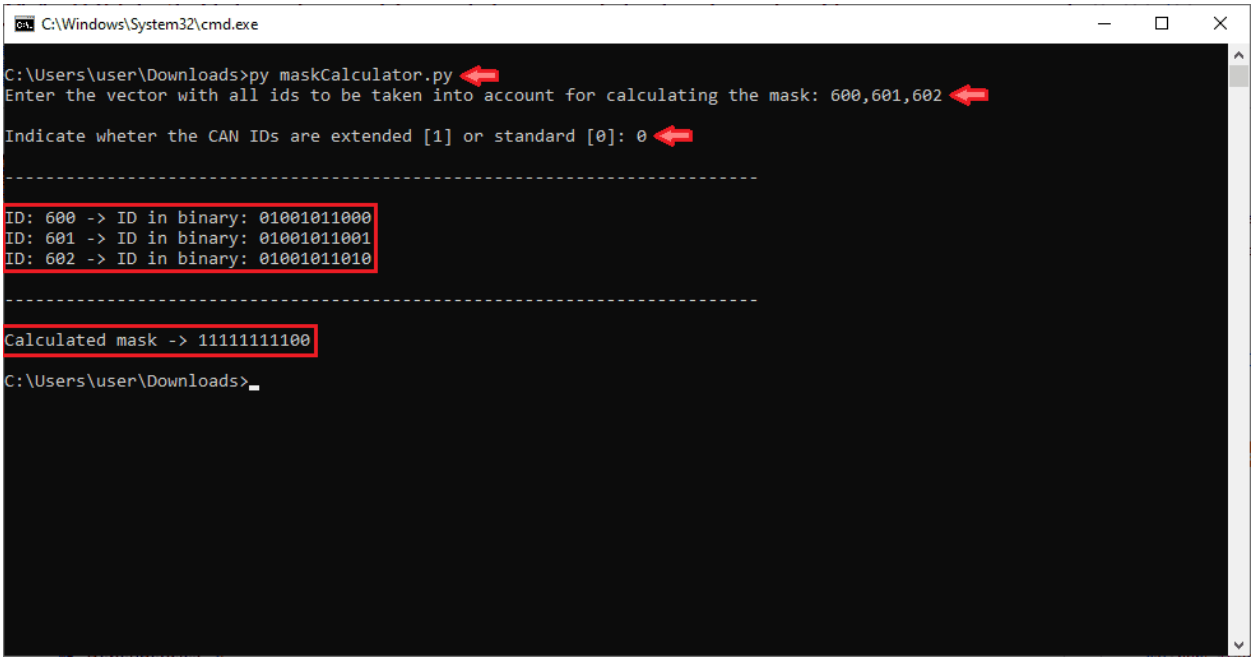


Fig. 1: **Example of maskCalculator program**

5.2 What does decimation mean?

EKF implementation in Veronte Autopilot 1x algorithm means that only one sensor can enter per run step.

Therefore, if more than one sensor is read in the same GNC step, then **the sensor with highest priority is the one introduced to the EKF**. The rest of the sensor measurements will be introduced to the EKF in subsequent GNC steps according to their priority order.

The **priority order of the sensors from highest to lowest priority** is as follows:

- GNSS position
- GNSS velocity
- Relative position sensor
- GNSS compass
- Magnetometer
- Static pressure
- Altimeter
- Velocity down
- Terrain mesh

Consequently, if there is a sensor with a high priority and it has a high refresh rate it may cause other sensors to never enter.

To avoid this, the parameter **decimation** has been introduced to discard a certain number of new measurements. That is, with decimation 10, only 1 out of 10 new measurements is entered.

It is recommended **not to change the default values** if users are not sure what they are doing.

5.3 Automations evaluation and execution

- **What is the approximate rate of evaluation of all automations?**

  The evaluation rate of these automations depends on the type of event in terms of priority: high and low events.

  High events are those that check whether a system variable is within the defined limits or those which check if the selected bits are correct. **Low events** are the rest of events of the system.

  Considering that automations run at **Core 2 (C2) frequency**, these are the evaluation rates of each type of event:

  – **High events**: Evaluated in all steps of C2.

  – **Low events**: The evaluation rate is given by the following equation:

$$\text{Evaluation rate} = \frac{\text{nº low events}}{\text{floor}(\frac{\text{nº of low events}+35}{36})} + 3$$

  Example

56 low events → low events are evaluated each **31 C2 steps**

- **Are the automations evaluated in sequential order from top to bottom (similar to block programs that are evaluated in sequential order)?**

  No, they are evaluated in the order of the automation ID, which may not necessarily coincide with the order in which they appear in the list of automations.

  For instance, considering the following automations configuration, the automation with ID 9 (*Auto Mode*), will be executed before the automation with ID 27 (*RDZ Button*), regardless of whether the automation *RDZ Button* is first in the "sequential order".

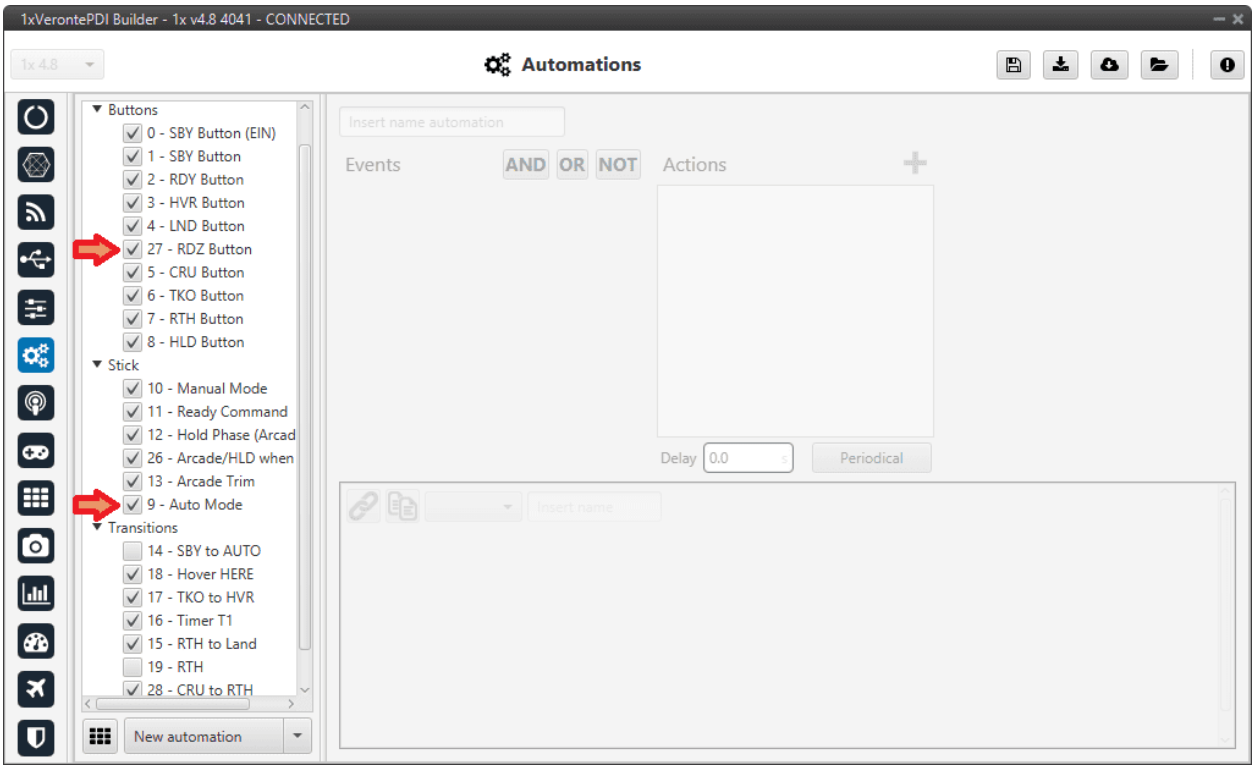


Fig. 2: **Automations evaluation order example**

- **Is it possible for one automation action to be executed before the complete list of automations is evaluated?**

  No, this is not possible because the Veronte Autopilot 1x system first checks all automation events and then executes the corresponding actions triggered by the evaluated events.

- **Could a block program use a variable set by an action of an automation before the complete list of automations is evaluated?**

  No, because block programs are executed before automations.

- **Could a CAN message with a variable value set by an automation be sent before the complete set of automations is evaluated?**

  No, because CAN is evaluated in Core 1, while automations are executed in Core 2. For more information on the tasks and functioning of each Core, please refer to the Core architecture section of the **1x Software Manual**.

CHAPTER

SIX

SOFTWARE CHANGELOG

This section presents the changes between the previous software version (**v.6.12.44**) and the current (**v.6.12.54**).

Added

- Search bar in the creation panels of Phases, Events, Actions and Automations

Improved

- Operator position interface
- To avoid user errors:
  - Default hwv option is now 4.8 instead of 4.0
  - CAN Terminator checkbox is only available if applicable
- To avoid undesirable behavior, editing the number of columns and rows of the SU and US matrices is now only available from the “Dimensions” parameter of the Actuator block
- Migration between versions procedure

Changed

- All item ids start at 0 instead of 1