
1x PDI Builder

Release 6.8.65

Embention

2023-10-04

CONTENTS

1	Quick Start	3
1.1	Download	3
1.2	Installation	3
2	Configuration	7
2.1	Veronte	7
2.1.1	Unit name	7
2.1.2	Attitude	8
2.1.3	Frequencies	11
2.1.4	Operator position	12
2.1.5	GPIO	13
2.1.6	Status	14
2.2	Connections	14
2.2.1	ADC	15
2.2.2	Arbiter	19
2.2.3	FTS	20
2.2.4	GPIO	21
2.2.5	I2C	24
2.2.6	Others	25
2.2.7	PWM	26
2.2.8	Serial	30
2.2.9	USB	32
2.3	Sensors	32
2.3.1	Accelerometer	32
2.3.2	Gyroscope	39
2.3.3	Magnetometer	45
2.3.4	Dynamic Pressure	52
2.3.5	Static Pressure	57
2.3.6	RPM	62
2.3.7	Lidar	62
2.3.8	Interneer	64
2.4	Input/Output	65
2.4.1	I/O Setup	65
2.4.1.1	Tunnel	68
2.4.1.2	Serial Custom Messages	70
2.4.1.3	NMEA Parser	74
2.4.1.4	Unescape port	75
2.4.2	CAN Setup	77
2.4.2.1	Configuration	77
2.4.2.2	Custom Messages	81

	2.4.2.3	Mailboxes	84
2.4.3		Custom Messages types	85
	2.4.3.1	Variable	85
	2.4.3.2	Checksum (CRC)	87
	2.4.3.3	Matcher	91
	2.4.3.4	Skip	92
	2.4.3.5	Parse ASCII	92
	2.4.3.6	Position	94
2.4.4		Digital Input	95
2.5		Control	102
	2.5.1	Phases	102
	2.5.2	Envelope	106
	2.5.3	Modes	110
	2.5.4	Arcade axis	114
2.6		Automations	116
	2.6.1	New automation	118
	2.6.2	Other options	122
	2.6.2.1	Events	124
	2.6.2.1.1	Alarm	127
	2.6.2.1.2	Area	129
	2.6.2.1.3	Button	130
	2.6.2.1.4	Mode	131
	2.6.2.1.5	Phase	132
	2.6.2.1.6	Route	132
	2.6.2.1.7	Timer	133
	2.6.2.1.8	Variable	135
	2.6.2.2	Actions	136
	2.6.2.2.1	Atmosphere Calibration	137
	2.6.2.2.2	Change active sensor	139
	2.6.2.2.3	Command block	139
	2.6.2.2.4	Custom CAN TX	142
	2.6.2.2.5	Custom Serial TX	143
	2.6.2.2.6	DEM calibration	144
	2.6.2.2.7	Enable/Disable Wind Estimation	145
	2.6.2.2.8	Envelope	146
	2.6.2.2.9	FTS Activation	147
	2.6.2.2.10	Feature	148
	2.6.2.2.11	Format SD	150
	2.6.2.2.12	Go to	151
	2.6.2.2.13	Mode	152
	2.6.2.2.14	Navigation	152
	2.6.2.2.15	Obstacle avoidance	154
	2.6.2.2.16	Output	154
	2.6.2.2.17	Periodical	156
	2.6.2.2.18	Phase	157
	2.6.2.2.19	Ports	158
	2.6.2.2.20	Run block program	158
	2.6.2.2.21	Safety Bits	160
	2.6.2.2.22	Select Arcade axis	161
	2.6.2.2.23	Stick priority	162
	2.6.2.2.24	Terrain obstacle	163
	2.6.2.2.25	Track	164
	2.6.2.2.26	User Log	167
	2.6.2.2.27	Variable	167

	2.6.2.2.28 Yaw	168
2.7	Communications	169
2.7.1	Ports	170
2.7.2	4G	171
2.7.3	Comstats	173
2.7.4	Iridium	175
2.7.5	Veronte LOS	176
2.8	Stick	177
2.8.1	Transmitter (1-4)	177
2.8.1.1	PPM	177
2.8.1.2	Exponential	179
2.8.1.3	Trim	180
2.8.1.4	Output	181
2.8.2	Virtual Stick	183
2.9	Block Programs	184
2.9.1	Control blocks	196
2.9.1.1	PID	196
2.9.1.2	T-Sched PID	200
2.9.1.3	Total Energy Control	203
2.9.1.4	Fuzzy Logic Controller	205
2.9.1.5	ECU Control	208
2.9.1.6	Quaternion Control	211
2.9.1.7	Driver Control Filter	213
2.9.1.8	System Identification	215
2.9.1.9	Predictive Control Block	216
2.9.2	Data Source/Sink blocks	218
2.9.3	Devices blocks	219
2.9.3.1	Clock	219
2.9.3.2	Gimbal	220
2.9.3.3	Stick	223
2.9.4	Execution Flow blocks	226
2.9.5	Guidance blocks	229
2.9.5.1	Guidance blocks common configuration	229
2.9.5.2	Climb	236
2.9.5.3	Cruise	242
2.9.5.4	Guidance Computation	244
2.9.5.5	Landing	244
2.9.5.6	Rendezvous	249
2.9.5.7	Taxi	253
2.9.5.8	VTOL	256
2.9.5.9	Yawing current	259
2.9.5.10	Yawing heading	260
2.9.5.11	Yawing north	261
2.9.5.12	Navigation guidance blocks	262
2.9.6	Library blocks	263
2.9.7	Logic blocks	267
2.9.7.1	AND	267
2.9.7.2	OR	267
2.9.7.3	OR	268
2.9.8	Math blocks	269
2.9.8.1	$f(x)$	269
2.9.8.2	$f(x,y)$	270
2.9.8.3	Polynomial	271
2.9.8.4	Vectors	271

2.9.9	Mode/AP Selection blocks	273
2.9.9.1	AP Selection	274
2.9.9.2	Arcade	275
2.9.9.3	Arcade Bounce	277
2.9.9.4	Arcade Extend	279
2.9.9.5	Manual	281
2.9.9.6	Mix	281
2.9.10	Navigation blocks	282
2.9.10.1	EKF Adapters	282
2.9.10.2	EKF Split	296
2.9.10.3	Navigation	297
2.9.11	Positions blocks	301
2.9.11.1	Constant Position	301
2.9.11.2	Move	302
2.9.11.3	Relative Vector	302
2.9.11.4	Read Feature	303
2.9.11.5	Write Feature	303
2.9.12	Sensors blocks	304
2.9.12.1	Altimeter	304
2.9.12.2	GNSS sensor	306
2.9.12.3	Magnetic Field	322
2.9.12.4	Magnetometer	324
2.9.12.5	Relative position	324
2.9.12.6	SRTM height	326
2.9.12.7	Static Pressure	327
2.9.13	Servos blocks	328
2.9.13.1	Actuator	328
2.9.13.2	Arc Trim	336
2.9.13.3	PWM	337
2.9.14	Signals blocks	338
2.9.14.1	3D Table Interpolation	339
2.9.14.2	Bound	340
2.9.14.3	EWMA Tau filter	340
2.9.14.4	FFT	341
2.9.14.5	Hysteresis	342
2.9.14.6	IIR Filter	344
2.9.14.7	Interpolation Vector	345
2.9.14.8	Ramp	346
2.9.14.9	Rate limiter	347
2.9.14.10	Signal generator	348
2.9.15	Type Casting blocks	351
2.10	Devices	352
2.10.1	Transponder/ADS-B	353
2.10.2	Cameras	355
2.10.3	Board	359
2.11	Telemetry	362
2.11.1	Telemetry	362
2.11.2	Sniffer	370
2.12	UI	372
2.12.1	Operation elements	372
2.12.2	Variables	373
2.12.3	Geoid	376
2.13	HIL	378
2.14	Safety	380

2.14.1	Checklist	381
2.14.2	Config Manager	383
2.14.3	Safety bits	384
3	Integration examples	391
3.1	AP communication with PC	391
3.2	ArcTrim Button	392
3.3	CAN communication	396
3.4	Data transmission between Veronte 1x Autopilots	401
3.5	External devices	403
3.5.1	Altimeters	403
3.5.1.1	Lidar	403
3.5.1.1.1	ADC lidar	404
3.5.1.1.2	I2C lidar	405
3.5.1.1.3	Using lidar readings	406
3.5.1.2	Radar	408
3.5.1.2.1	Ainstein CAN Radar	408
3.5.1.2.2	Smartmicro CAN Radar	413
3.5.2	External sensors	418
3.5.2.1	LM335 with Autopilot 4x	418
3.5.2.2	Magnetometer Honeywell HMR2300	422
3.5.2.2.1	RS-232	422
3.5.2.2.2	RS-485	426
3.5.2.3	MEX as Magnetometer Honeywell HMR2300	427
3.5.2.4	OAT Sensor	432
3.5.2.5	Vectornav VN-300	434
3.5.2.5.1	Vectornav VN-300 configuration	437
3.5.3	Radios	438
3.5.3.1	Digi internal radio	438
3.5.3.1.1	Configuration	438
3.5.3.1.2	Operational range	444
3.5.3.2	Microhard internal radio	444
3.5.3.3	External radios	446
3.5.4	Servos	448
3.5.4.1	PWM	448
3.5.4.2	Serial	451
3.5.4.2.1	Volz DA26 - RS485	452
3.5.5	Stick	455
3.5.5.1	PPM Stick	455
3.5.5.1.1	General case: GND unit sends commands directly to the air unit	455
3.5.5.1.2	Simulation case (HIL)	462
3.5.5.1.3	On-board PPM receiver case	462
3.5.5.2	USB joystick	468
3.5.5.3	Virtual Stick	468
3.5.6	Veronte products	469
3.5.6.1	Autopilot 4x	469
3.5.6.2	CEX/MEX	477
3.5.6.3	MC01	486
3.5.6.4	VSE (Veronte Stick Expander)	491
4	Troubleshooting	493
4.1	Communication lost with internal Digi radio	493
4.2	Debug serial messages transmission	494
4.3	Maintenance mode	497

4.4	Maintenance mode (loaded with errors)	498
4.5	Radios paired but 1x air unit not showing	499
4.6	Reducing GNC Task frequency	500
4.7	Trajectory Overshoot	500
4.8	Unstable communication with CEX/MEX	501
5	FAQ	503
5.1	What does decimation mean?	503



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

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 1x autopilot has been detected on Veronte Link, install 1x PDI Builder.

1.1 Download

Once the **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 “1xPDIBuilder.exe” and follow the indications.

1. Click on Next:

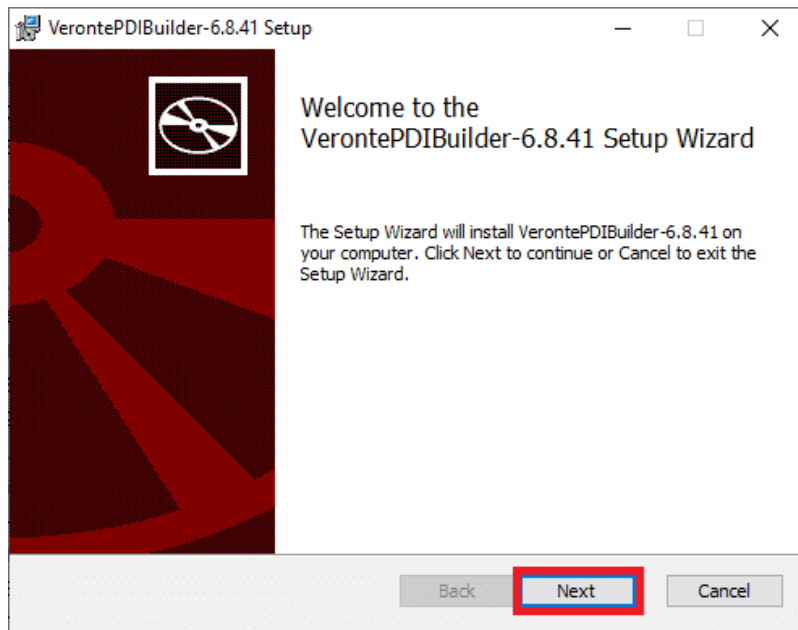


Fig. 1: Windows Installation Step 1

2. Select the desired directory where the software will be installed and click on next.

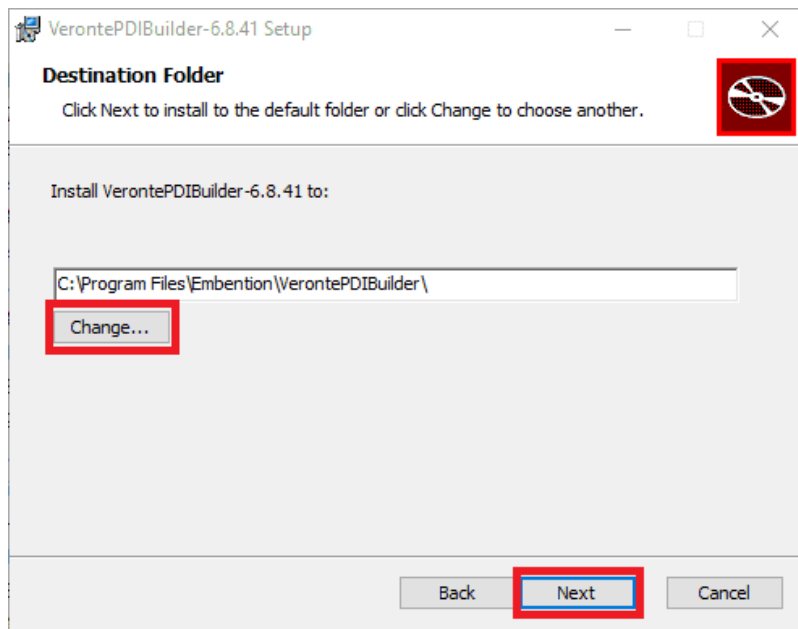


Fig. 2: Windows Installation Step 2

3. Finally, click on Install (administrator rights are needed):

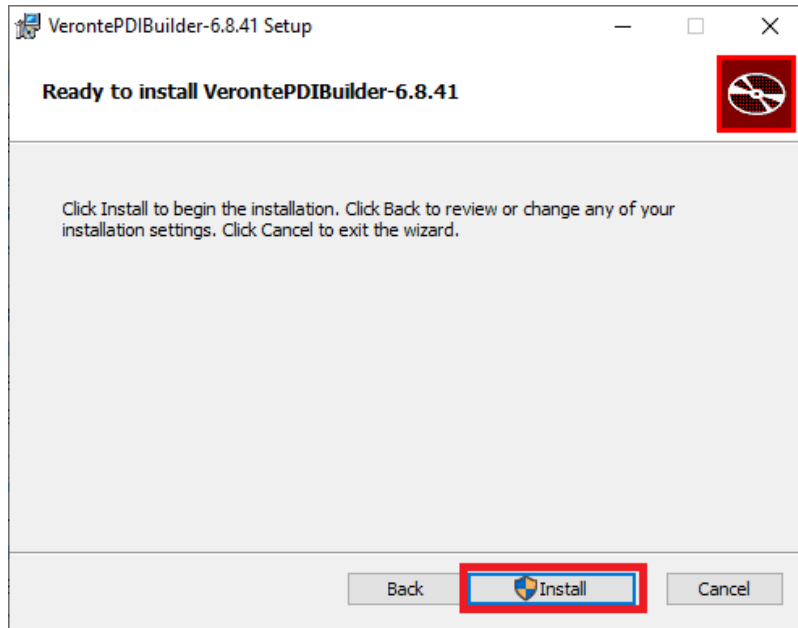


Fig. 3: Windows Installation Step 3

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” and then, click on “Turn windows Defender Firewall on or off”.

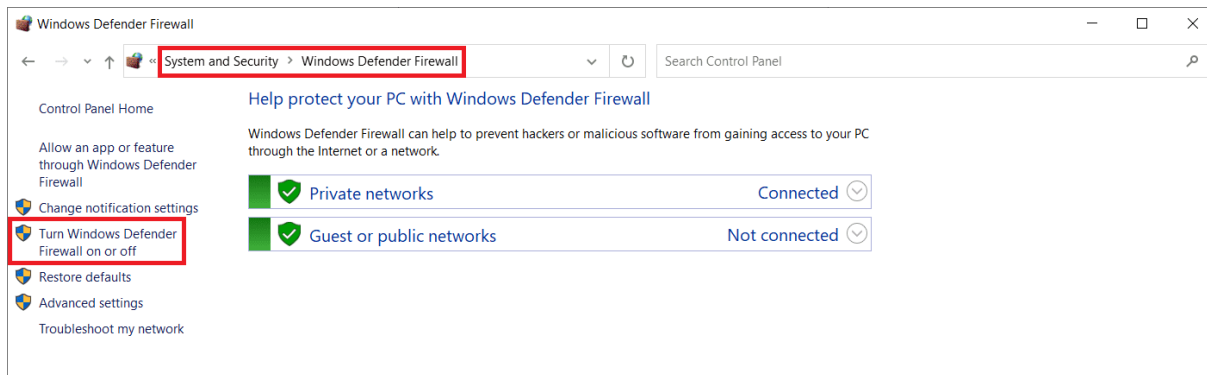


Fig. 4: Windows Defender Firewall

CONFIGURATION

2.1 Veronte

2.1.1 Unit name

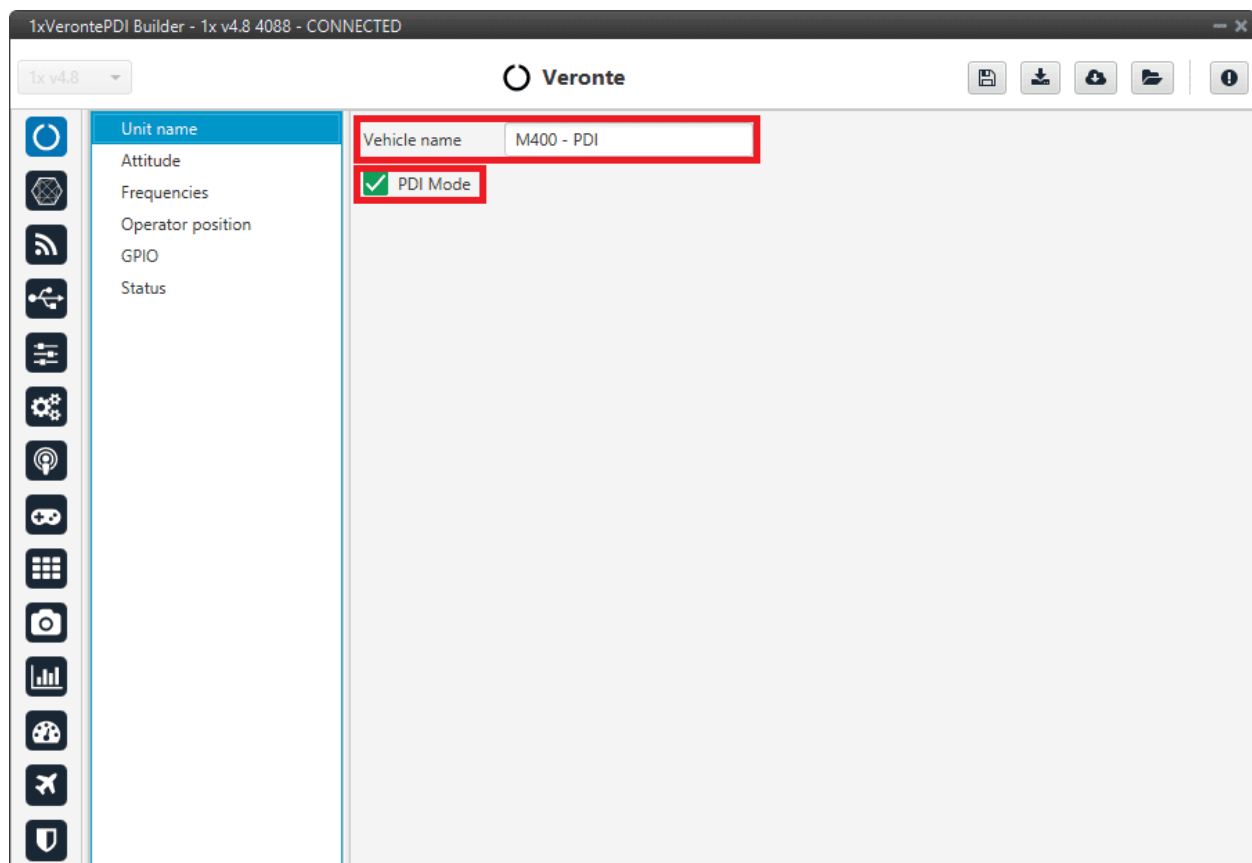


Fig. 1: Unit name section

- **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 1x autopilot is not in INI phase.

Warning:

- **Not** being in **PDI mode**, the user cannot do out of the INI phase:
 - Reboot 1x autopilot
 - 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 1x autopilot 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**.

Tip: It is recommended to use normal mode when the configuration is finished, while PDI mode is useful during the development process.

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. 1x autopilot axis are drawn on the autopilot's external case as defined in the [Hardware Installation](#).

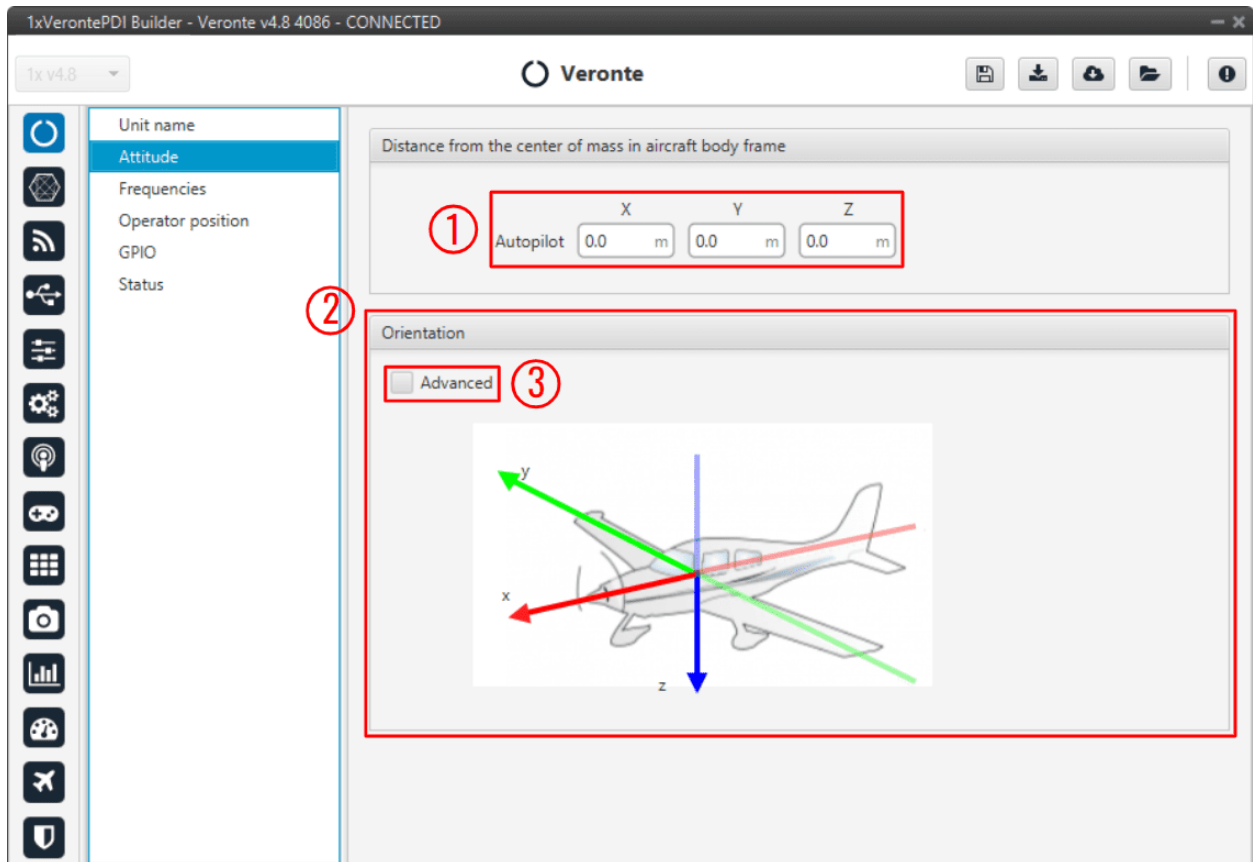


Fig. 2: Attitude section

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

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

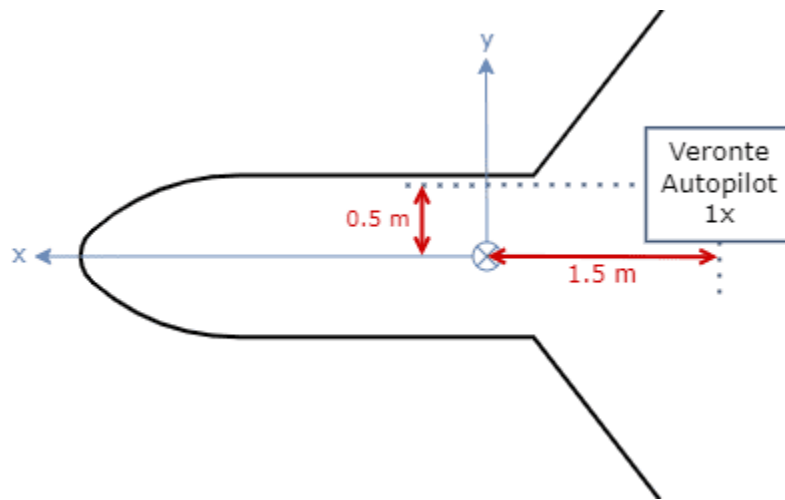
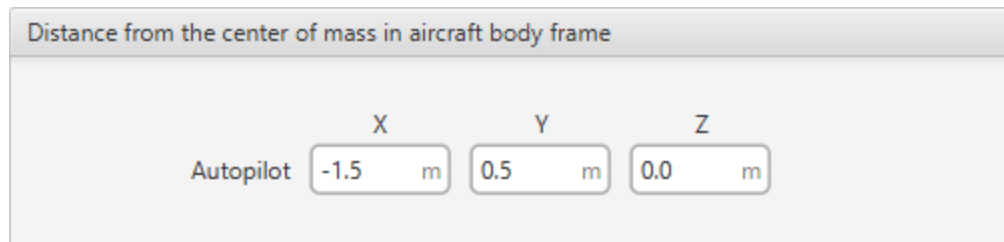


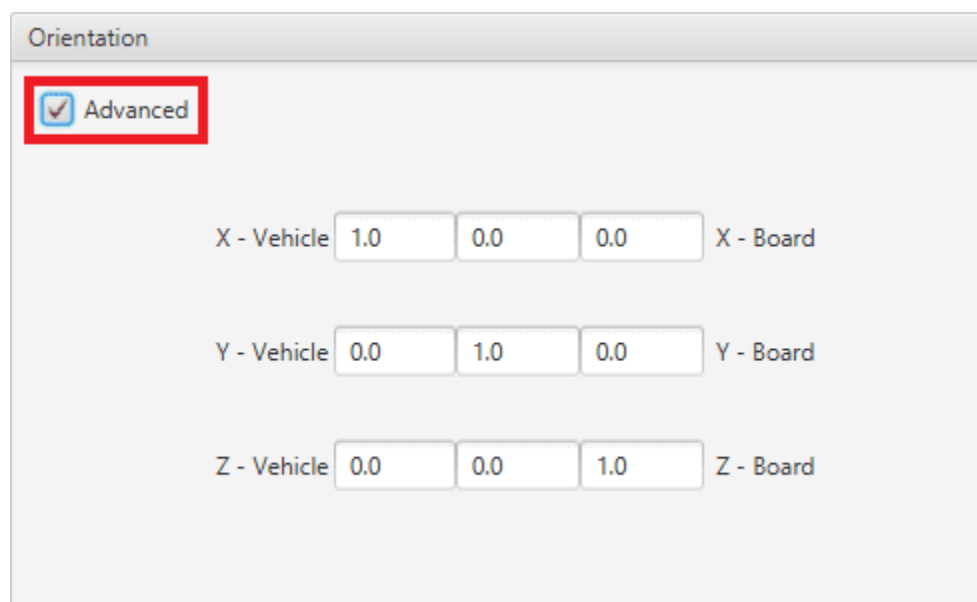
Fig. 3: Distance from the center of mass in aircraft body frame - Example



	X	Y	Z
Autopilot	-1.5 m	0.5 m	0.0 m

Fig. 4: 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.



	X - Vehicle	Y - Vehicle	Z - Vehicle	
X - Board	1.0	0.0	0.0	X - Board
Y - Board	0.0	1.0	0.0	Y - Board
Z - Board	0.0	0.0	1.0	Z - Board

Fig. 5: Advanced orientation

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

Fig. 6: 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**.

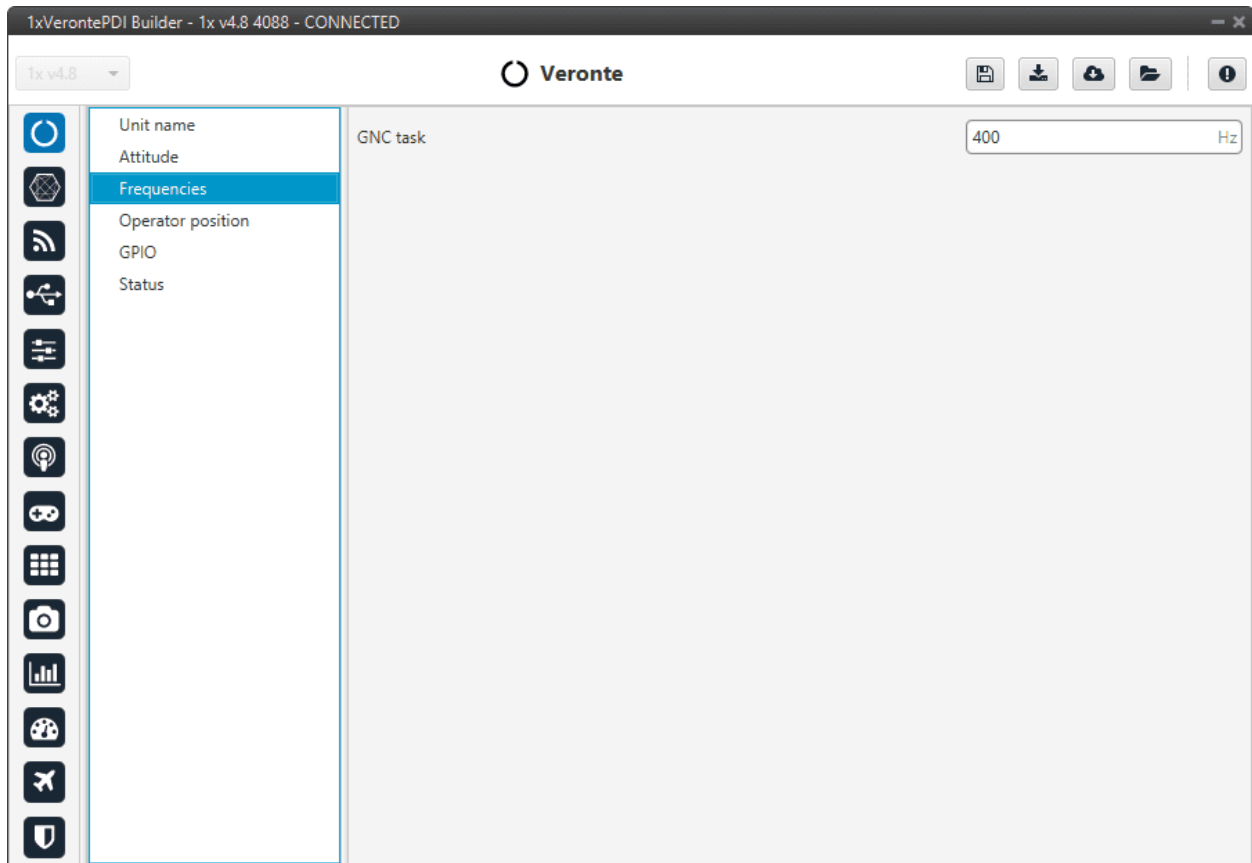


Fig. 7: Frequencies section

Warning: Only 400 Hz can be used 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

The operator position is the autopilot position from which the distance allowed by the licence is calculated.

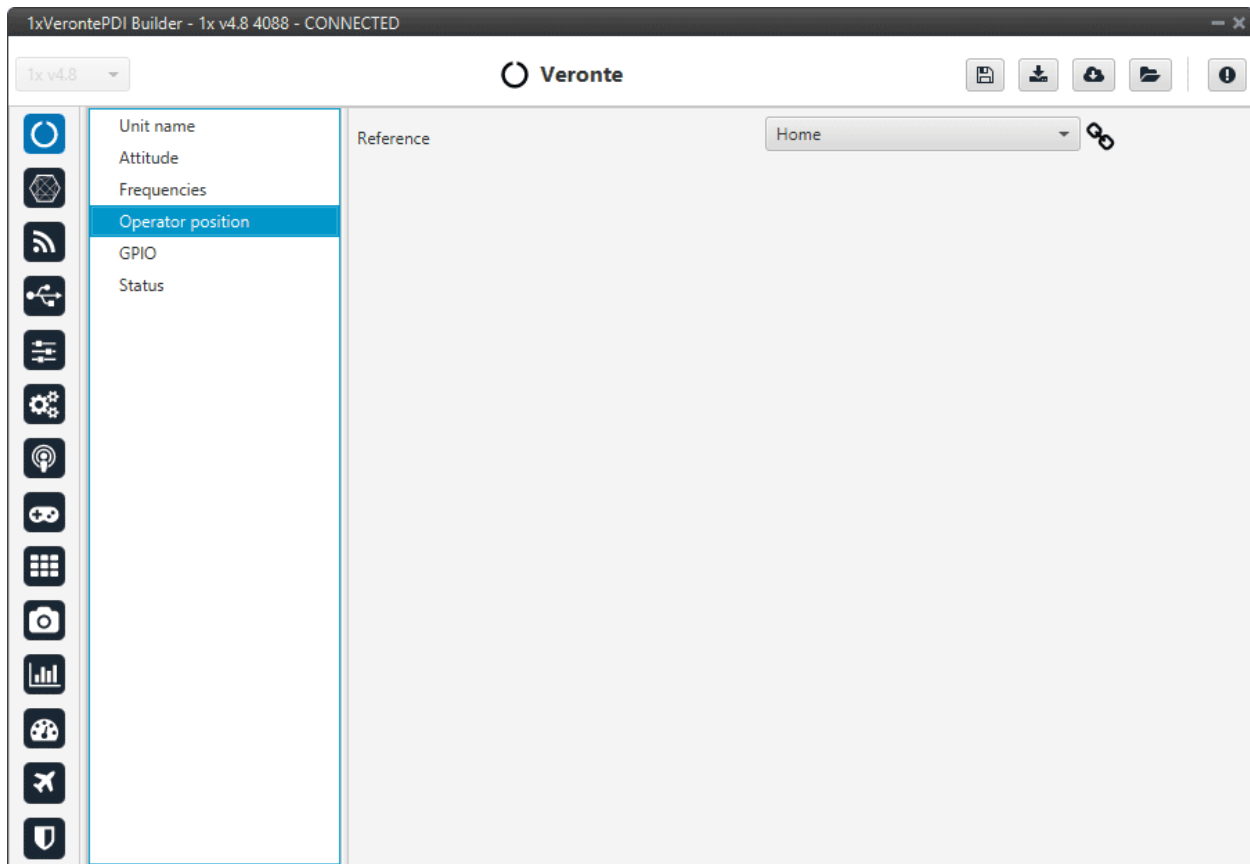


Fig. 8: Operator position section

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

2.1.5 GPIO

In this tab each individual GPIO behavior can be configured:

Signal	GPIOId	IO	Pull-up	Function	Qsel
I/O1	GPIO 0	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O2	GPIO 1	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O3	GPIO 147	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O4	GPIO 3	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O5	GPIO 4	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O6	GPIO 150	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O7	GPIO 6	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O8	GPIO 152	GPIO as output	Pull-up disabled	Mux 1	Sync
I/O9	GPIO 8	GPIO as input	Pull-up disabled	Mux 0	ASync
I/O10	GPIO 9	GPIO as output	Pull-up disabled	Mux 0	Sync
I/O11	GPIO 10	GPIO as output	Pull-up disabled	Mux 0	Sync
I/O12	GPIO 11	GPIO as output	Pull-up disabled	Mux 0	Sync
I/O13	GPIO 12	GPIO as output	Pull-up disabled	Mux 0	Sync
I/O14	GPIO 13	GPIO as output	Pull-up disabled	Mux 0	Sync
I/O15	GPIO 159	GPIO as output	Pull-up disabled	Mux 0	Sync
I/O16	GPIO 160	GPIO as output	Pull-up disabled	Mux 0	Sync
EQEP_A	GPIO 100	GPIO as input	Pull-up disabled	Mux 0	ASync
EQEP_B	GPIO 101	GPIO as input	Pull-up disabled	Mux 0	ASync
EQEP_S	GPIO 102	GPIO as input	Pull-up disabled	Mux 0	ASync
EQEP_I	GPIO 103	GPIO as input	Pull-up disabled	Mux 0	ASync

Fig. 9: GPIO section

1. **Signal:** Pin ID as described in [Hardware installation - Electrical](#) section of the **1x user 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.
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 recognise the 1x autopilot.

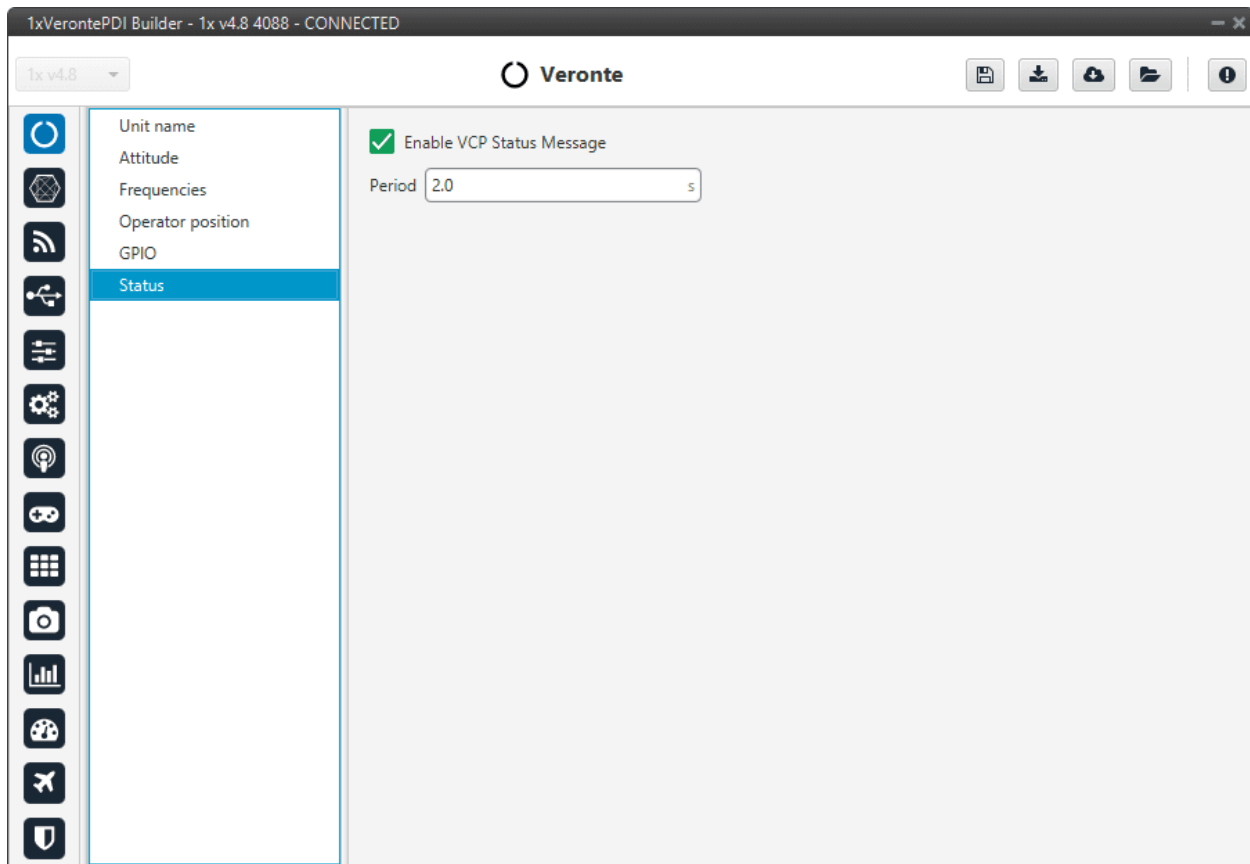


Fig. 10: Status section

- **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 section [Hardware Installation - Electrical](#).

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.

1x autopilot 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**.

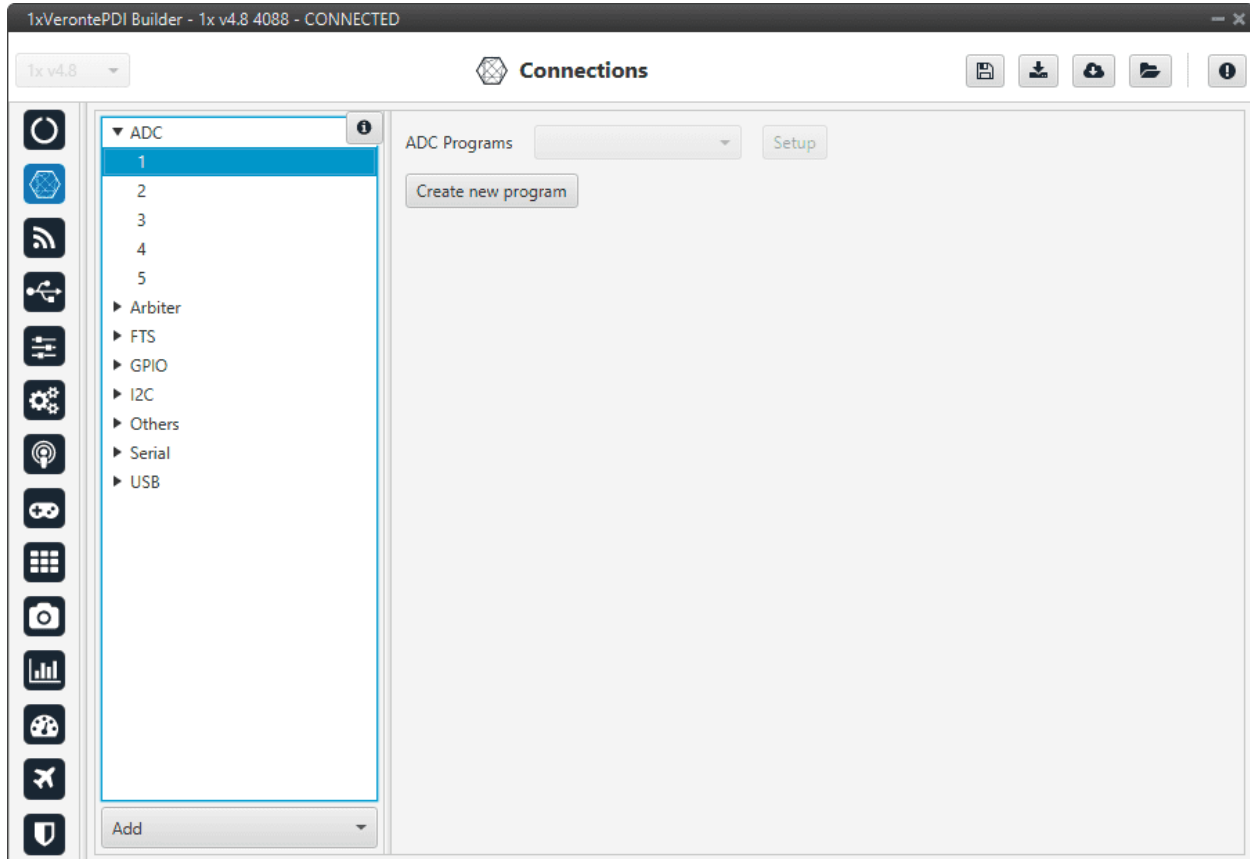


Fig. 11: ADC menu

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.

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 $y = -0.0498x^4 + 0.3002x^3 - 0.3083x^2 + 1.2423x + 0.15$, 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.

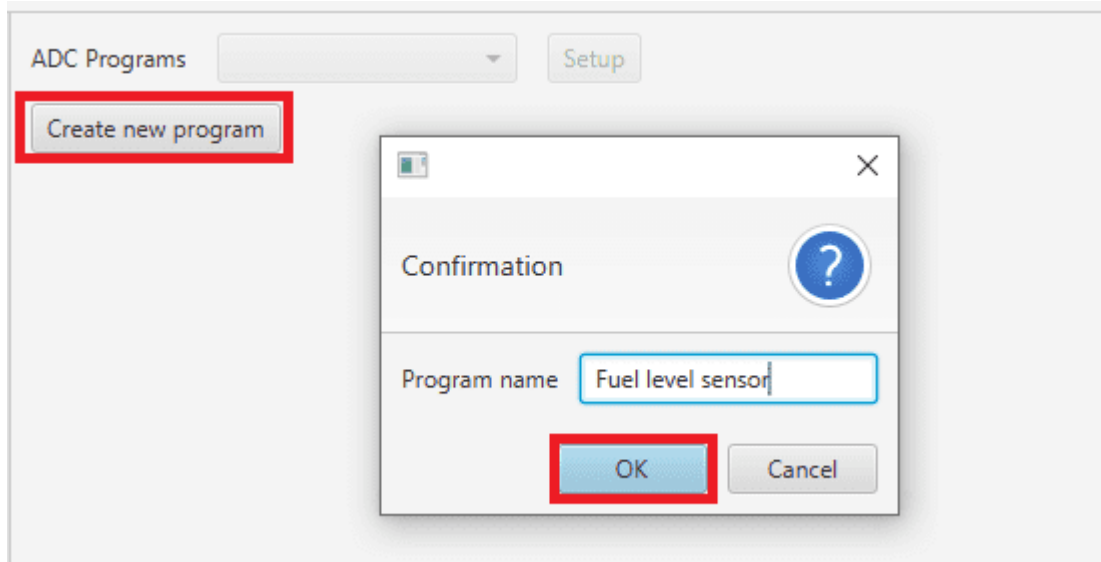


Fig. 12: Create ADC program

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

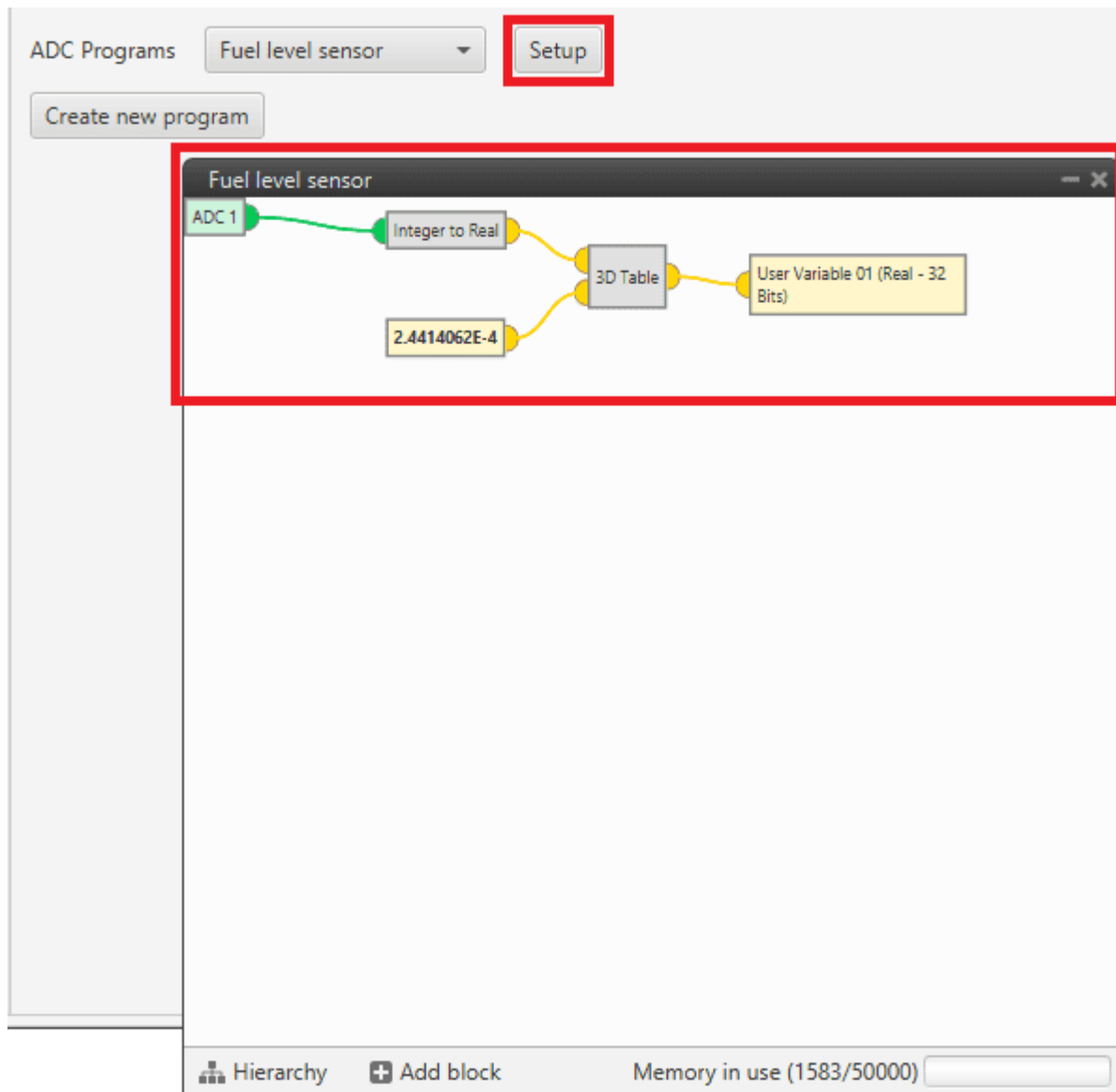


Fig. 13: ADC default program

Now, the program has to be customised for this application. See more information about programs in the [Block Programs](#) section.

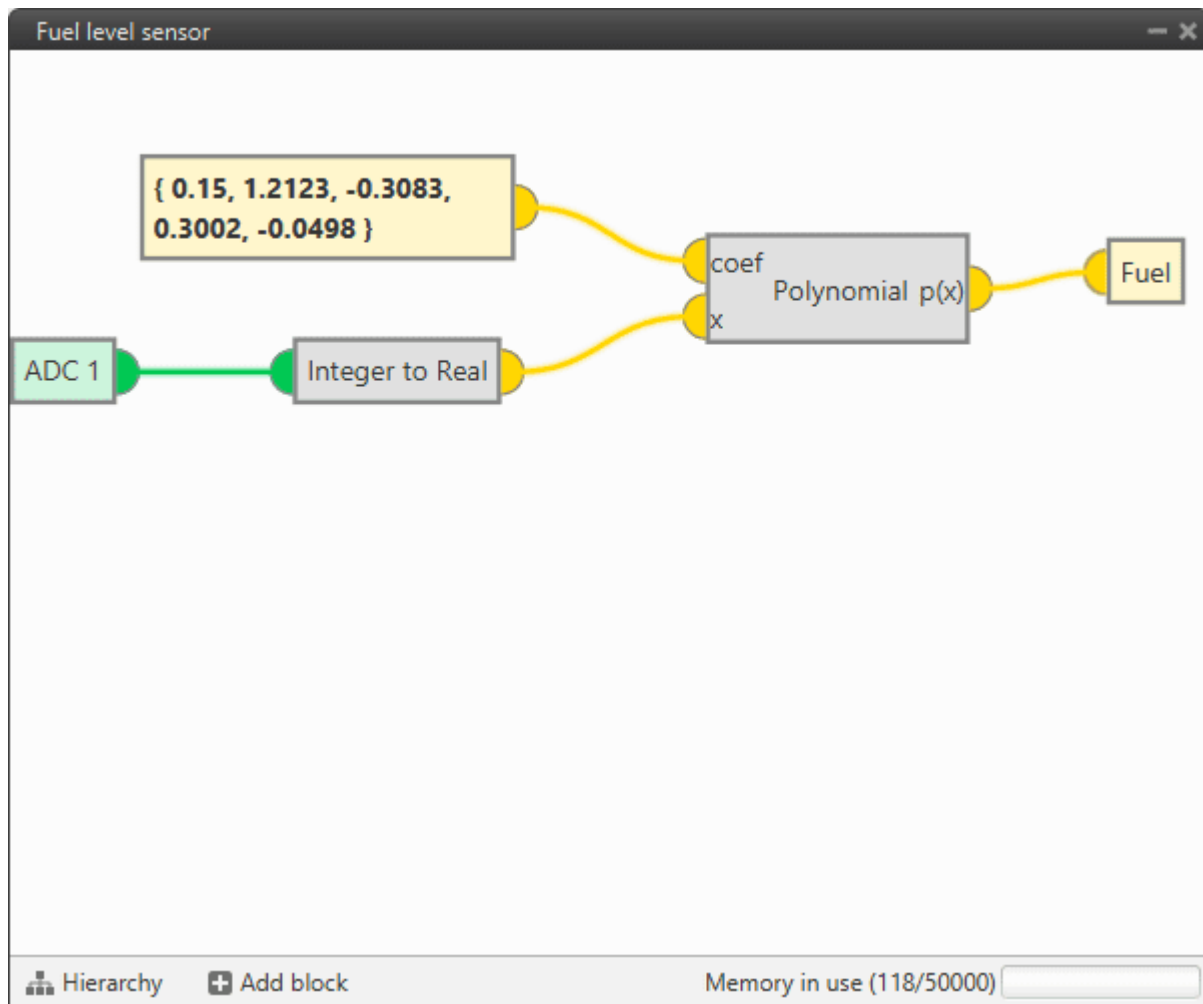


Fig. 14: ADC conversion for a Fuel Level Sensor

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* section.

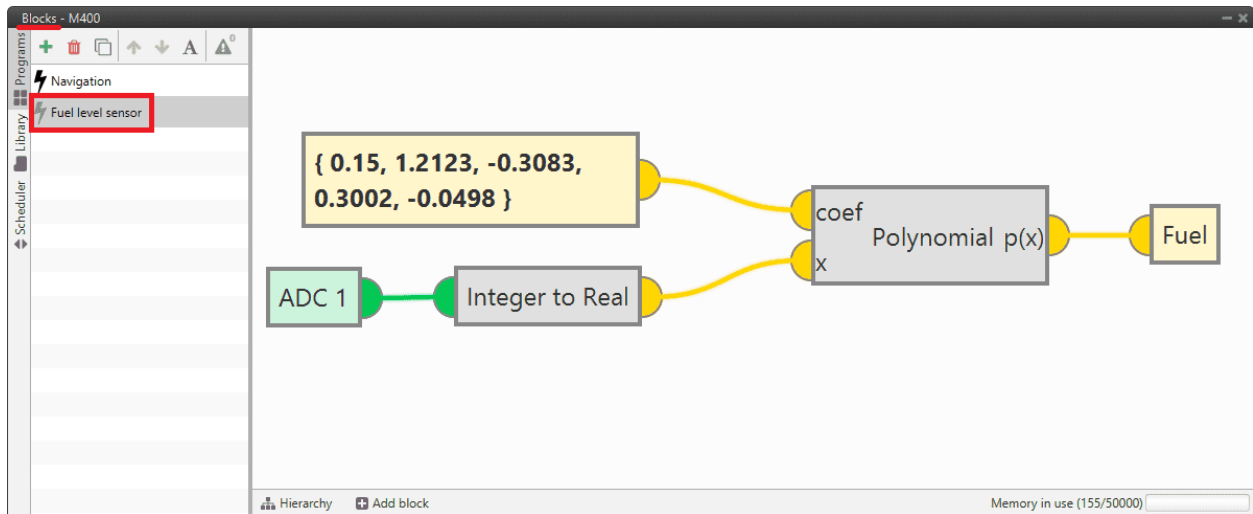


Fig. 15: ADC program in Block Programs

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).

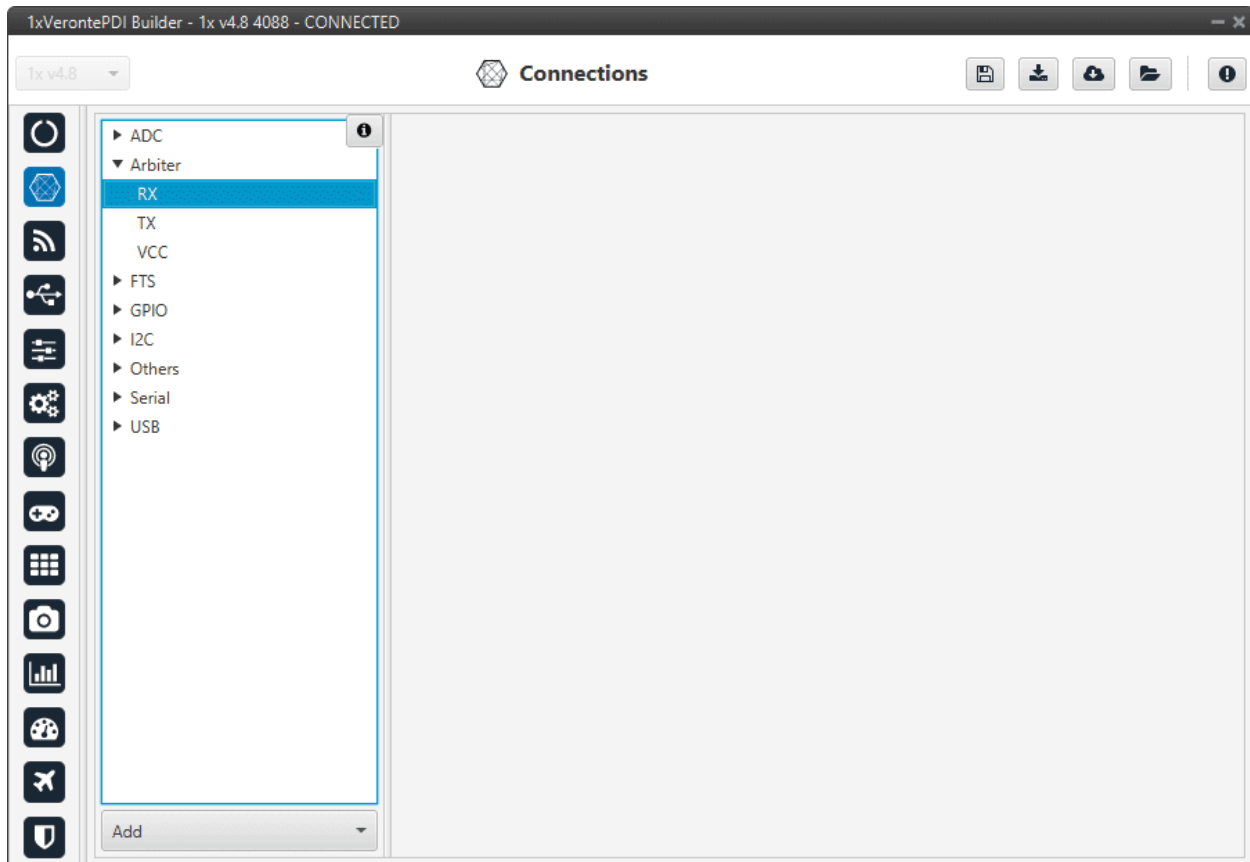


Fig. 16: Arbiter menu

2.2.3 FTS

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

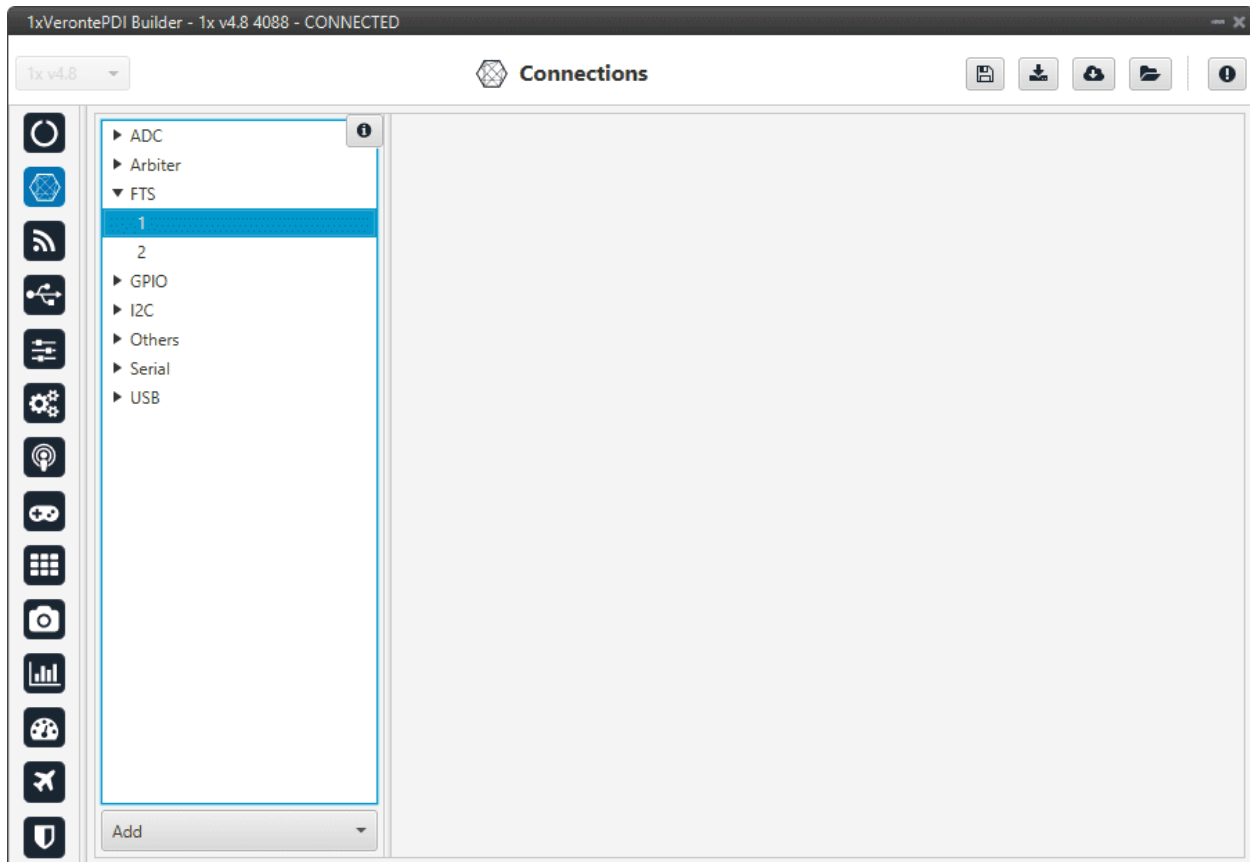


Fig. 17: FTS menu

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.

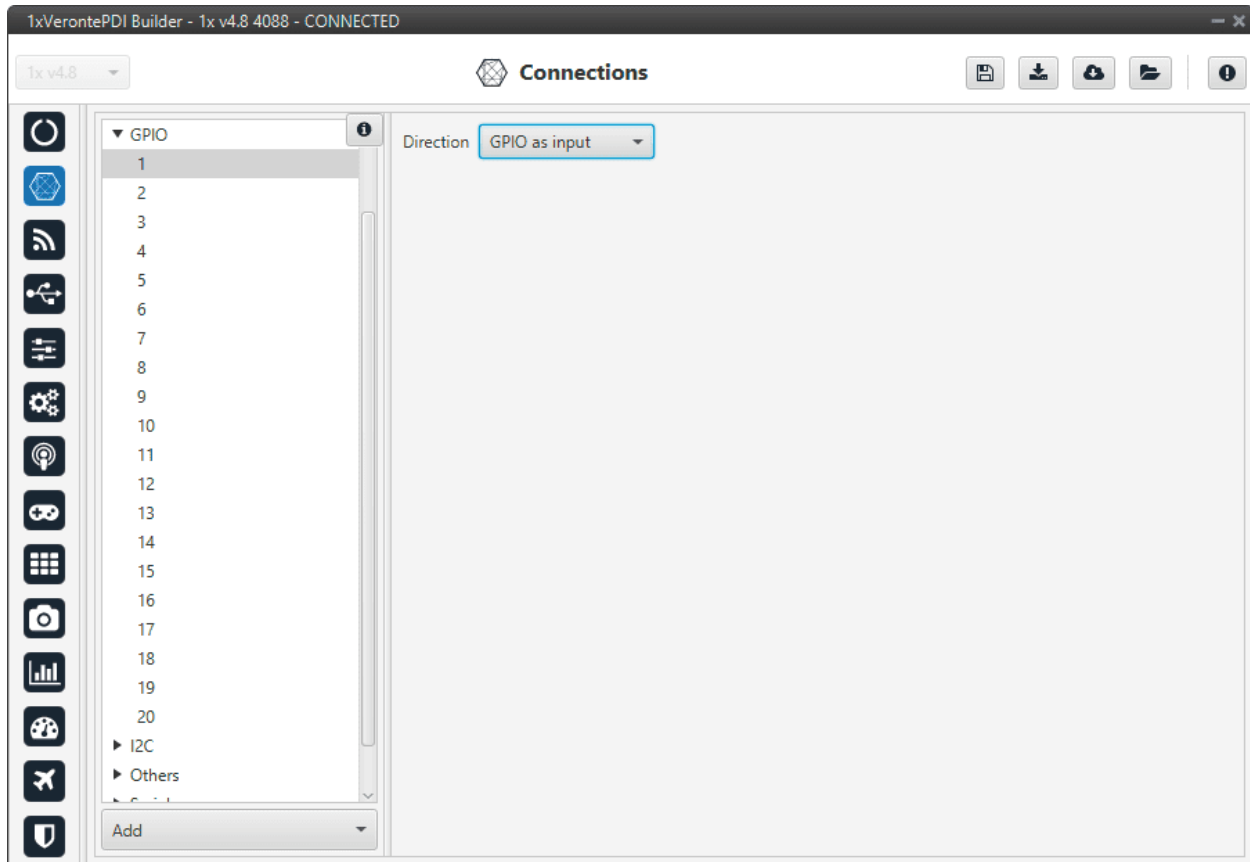
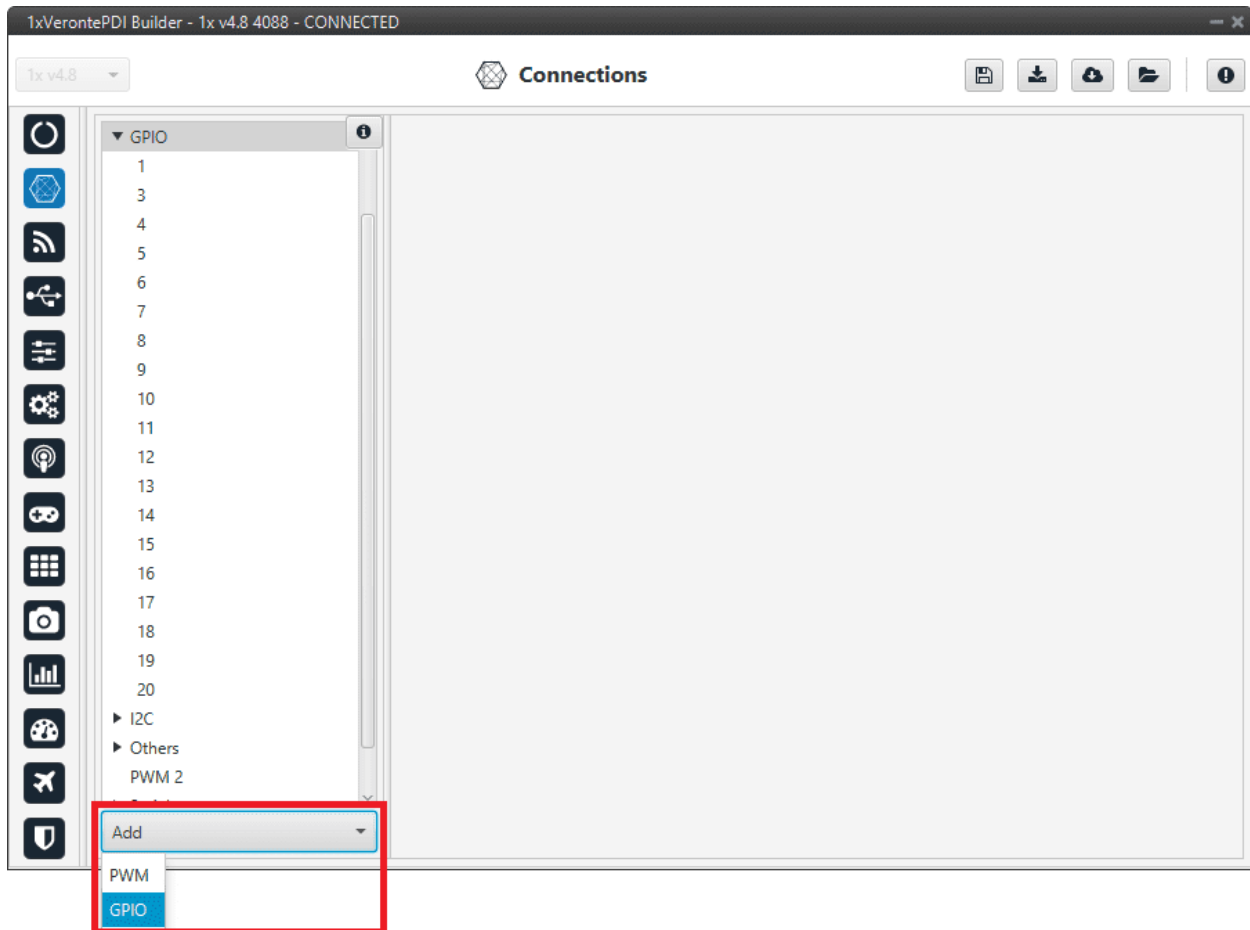


Fig. 18: GPIO menu

1x autopilot 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:



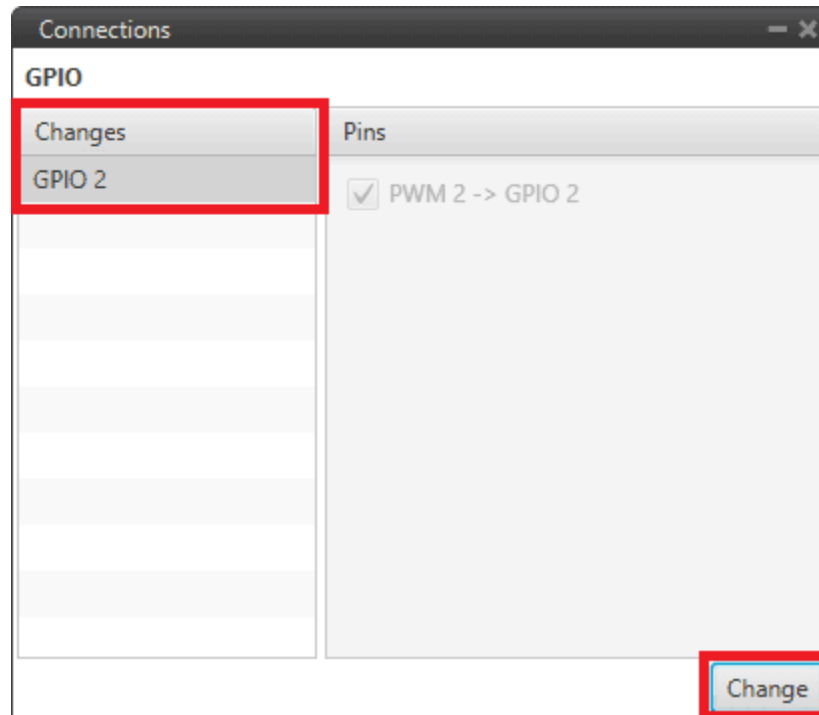


Fig. 19: Add GPIO

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.

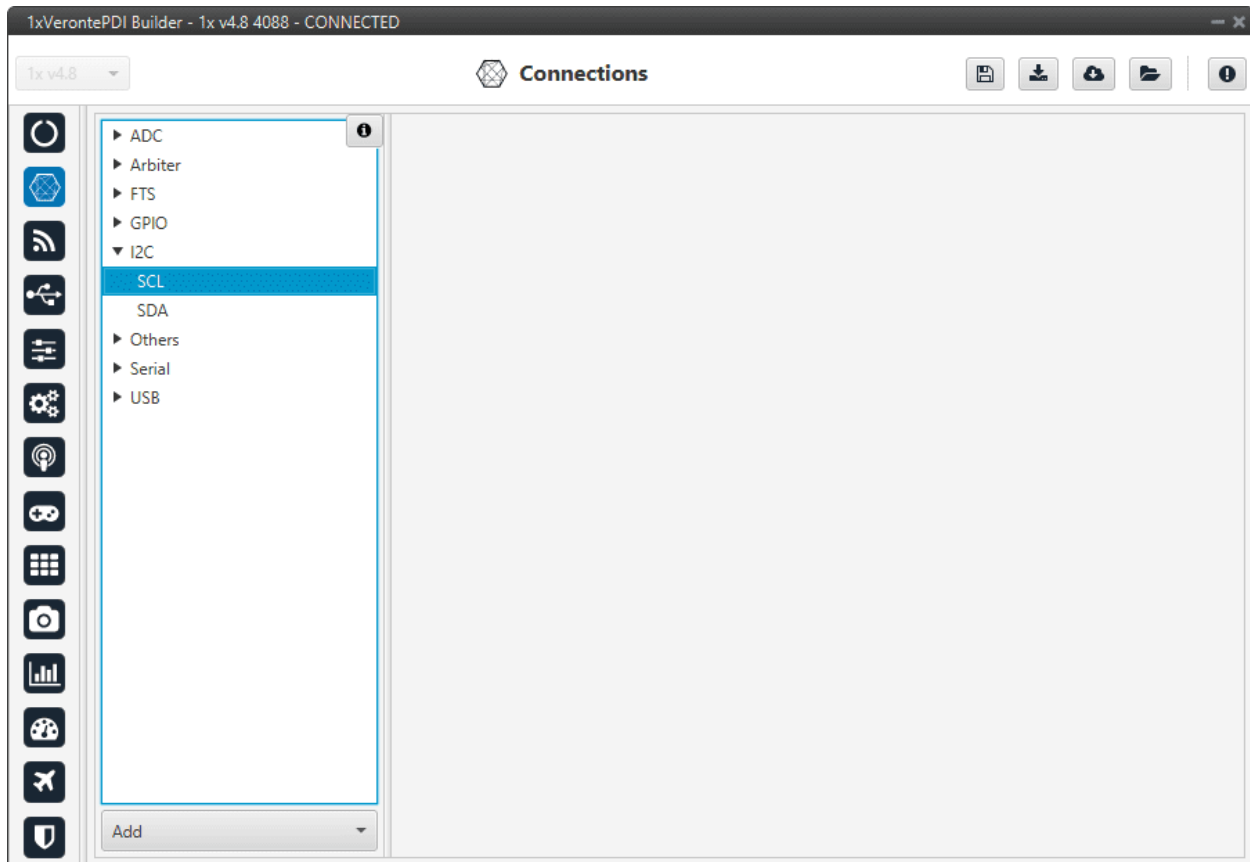


Fig. 20: I2C menu

2.2.6 Others

- GND: Ground.
- Power.

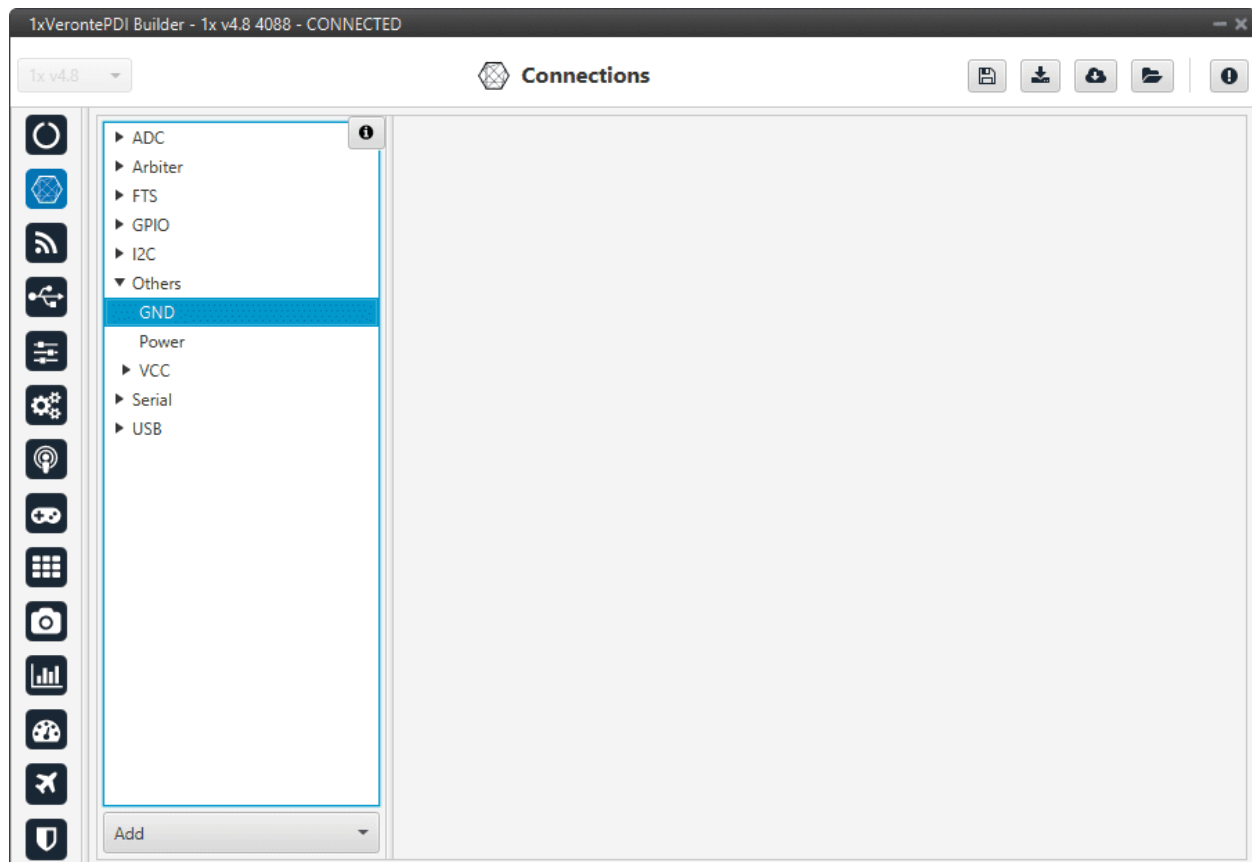


Fig. 21: Others menu

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 behaviour. 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*:

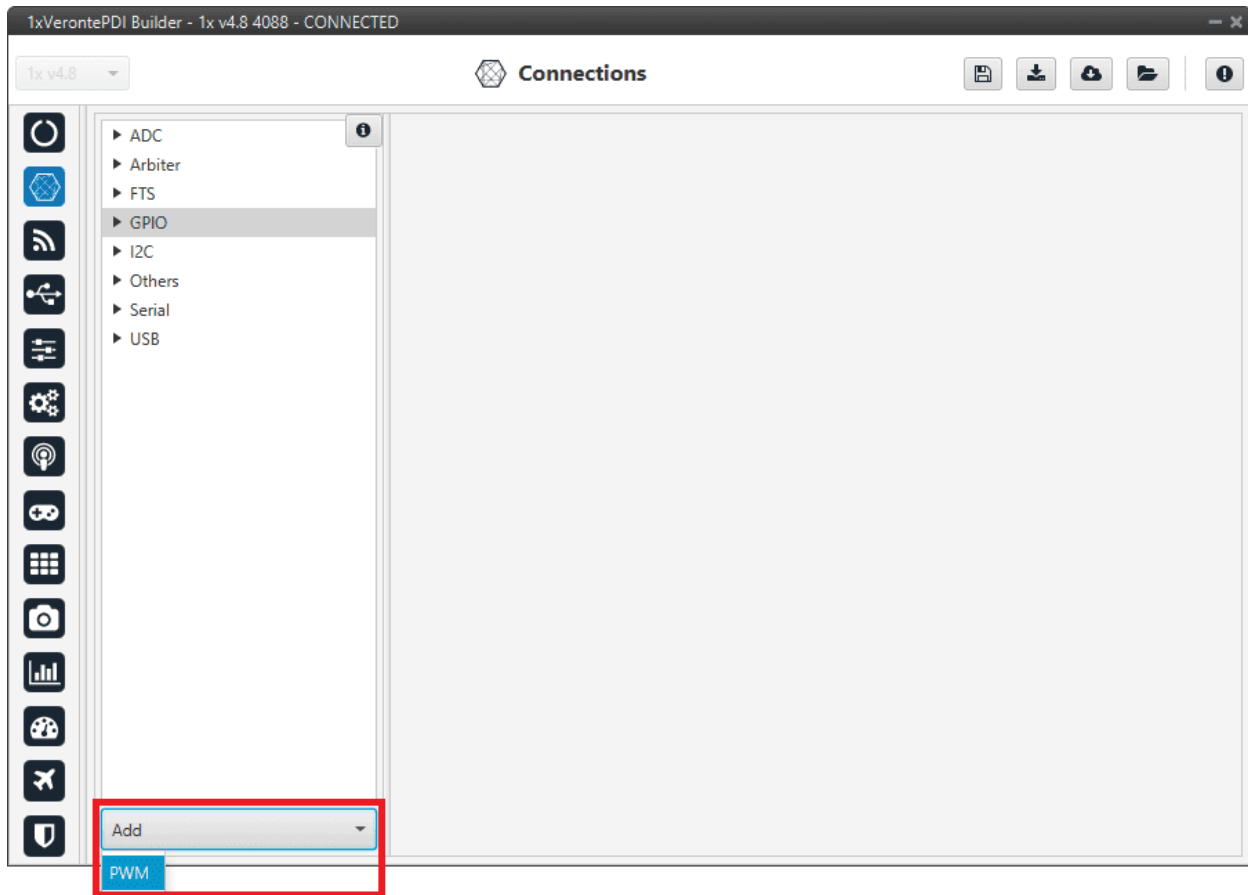


Fig. 22: Add PWM

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

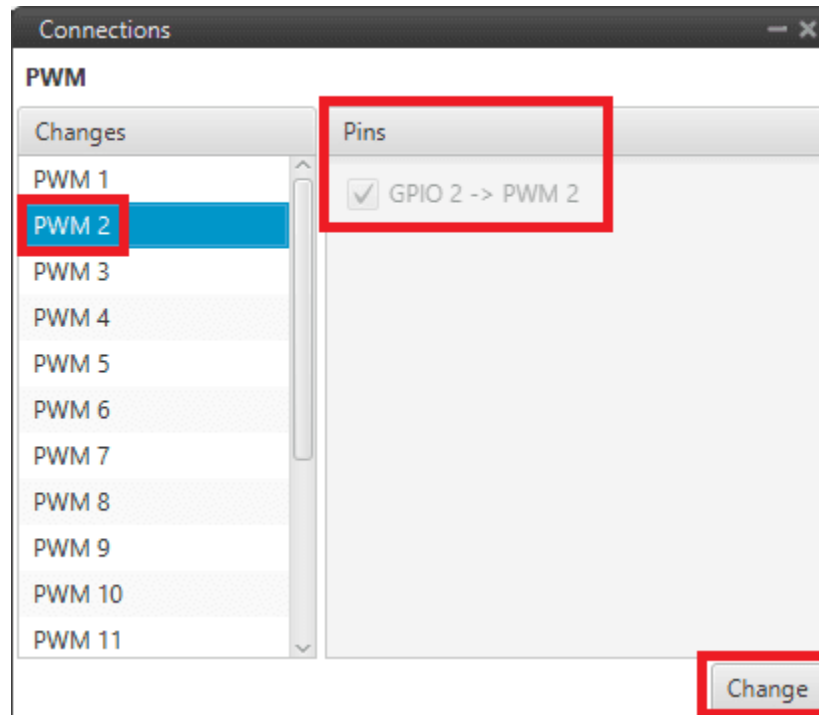


Fig. 23: PWM change

As shown in the image below, the GPIO 2 output is now missing as it has been changed to a PWM output.

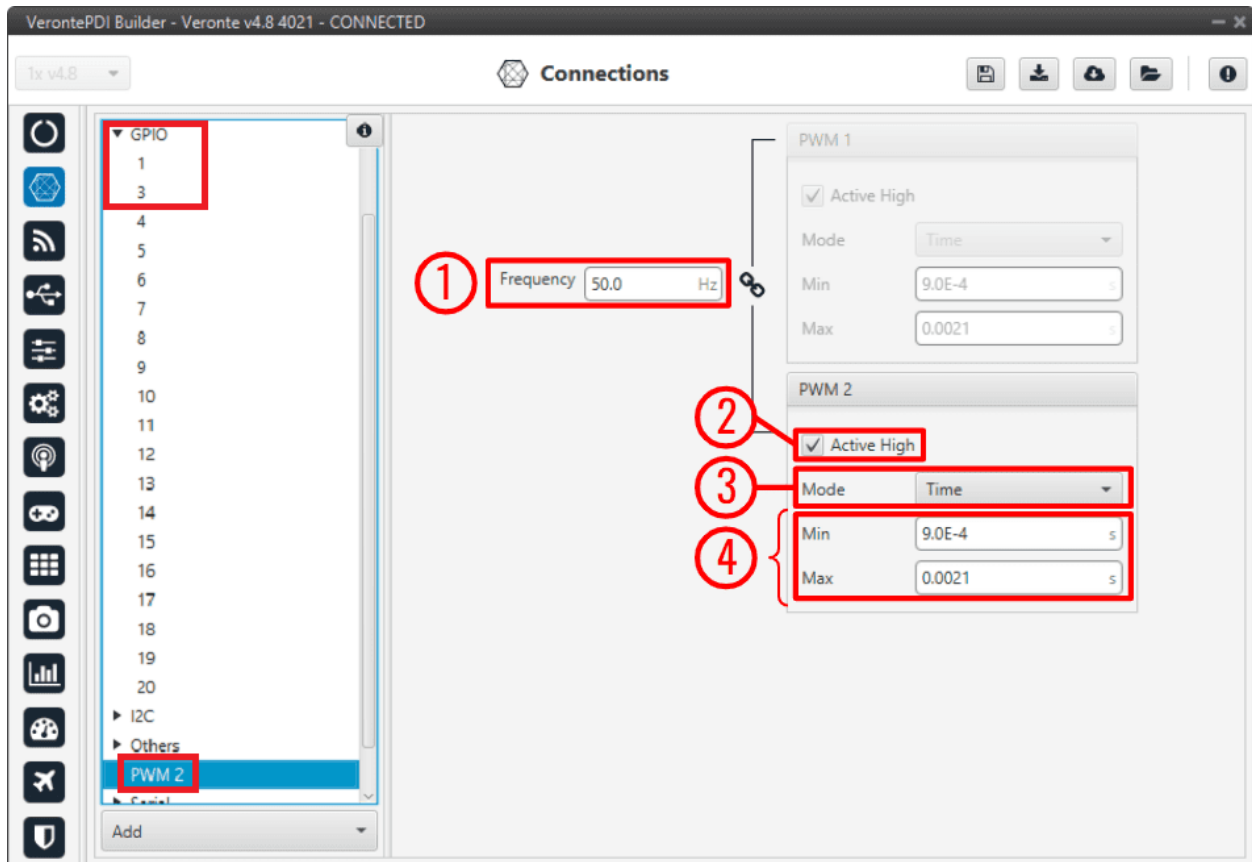


Fig. 24: PWM menu

In this menu the following parameters can be configured:

1. **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 1 is changed, the one of PWM 2 also changes. The following table shows the PMW pairs as configured in 1x autopilot.

PWM Pairs	
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	PWM 20

2. **Active High:** Enable/disable. Polarity high or low.
3. **Mode:** The options available are **Time** and **Duty cycle**. The second option is a different way of indicating the pulse width. Now the value indicated 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 these two extremes, the pulse width can vary as in the examples shown in the following figure.

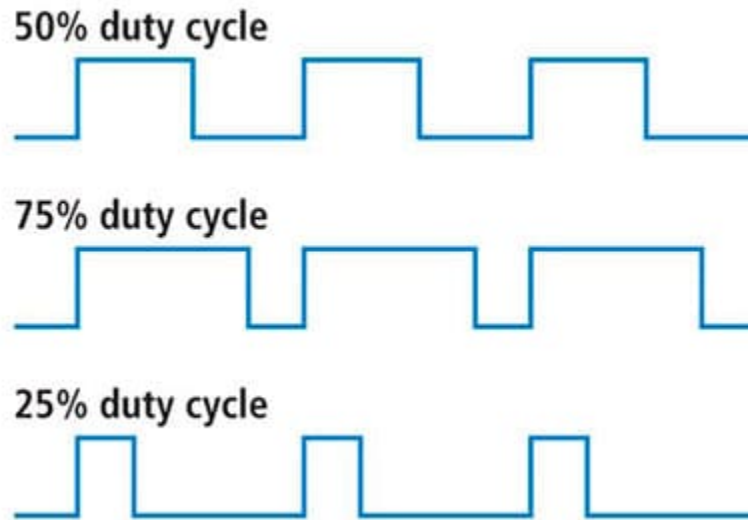


Fig. 25: Duty Cycle

4. **Min/Max:** These parameters are the pulse width values that will make the servo/actuator go to its lowest and highest position. As an example let's consider the servo of an aircraft elevator, a pulse sent by 1x autopilot 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 1x autopilot, 1 port RS-232 and 1 port RS-485, however more can be added by using a CEX. Each one of the serial interfaces is associated with a set of pins.

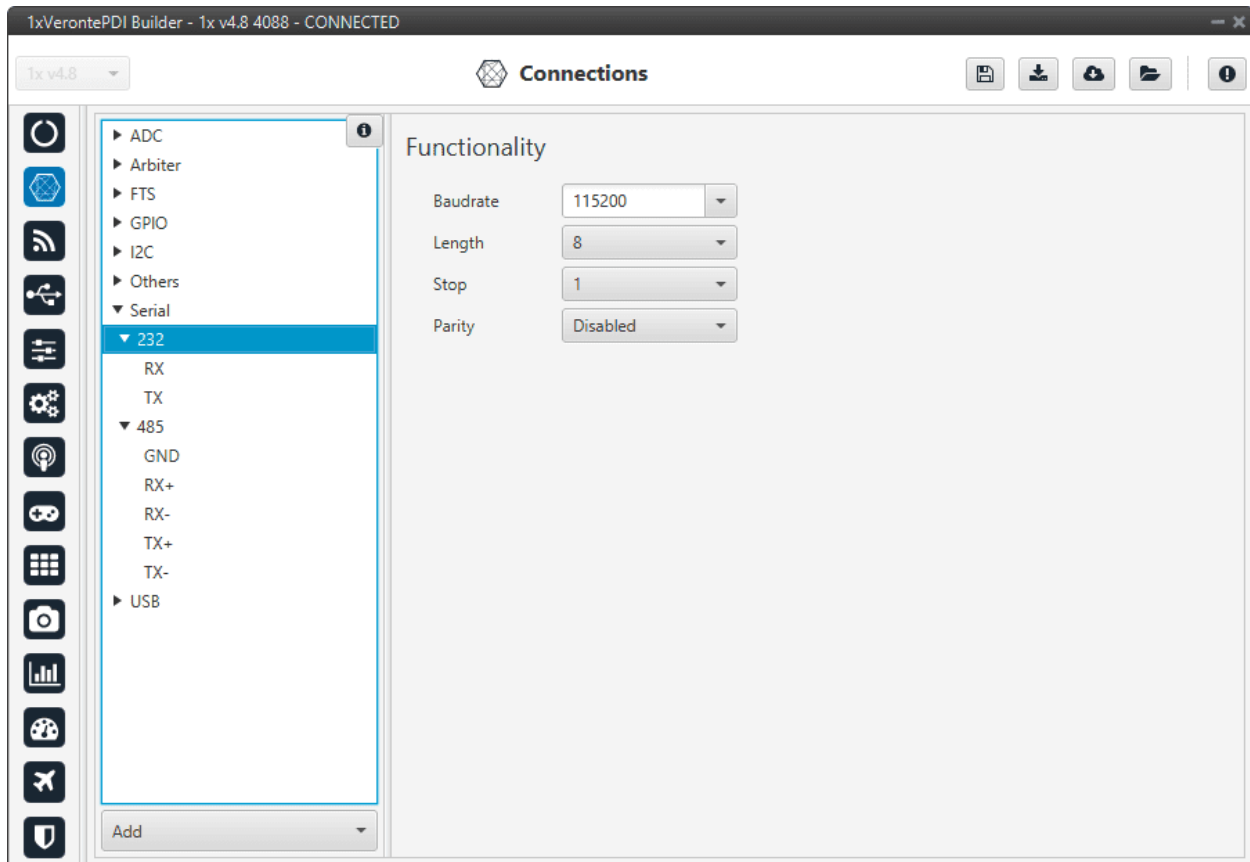


Fig. 26: Serial menu

The following fields can be configured:

- **Baudrate:** This specifies how fast data is sent over a serial line.
- **Length:** This defines the number of data bits in each character: 4 to 8.
- **Stop:** 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, 1x autopilot should match with them in order to be able to communicate.

Compatibility table:

Port name	RS-232	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

2.2.9 USB

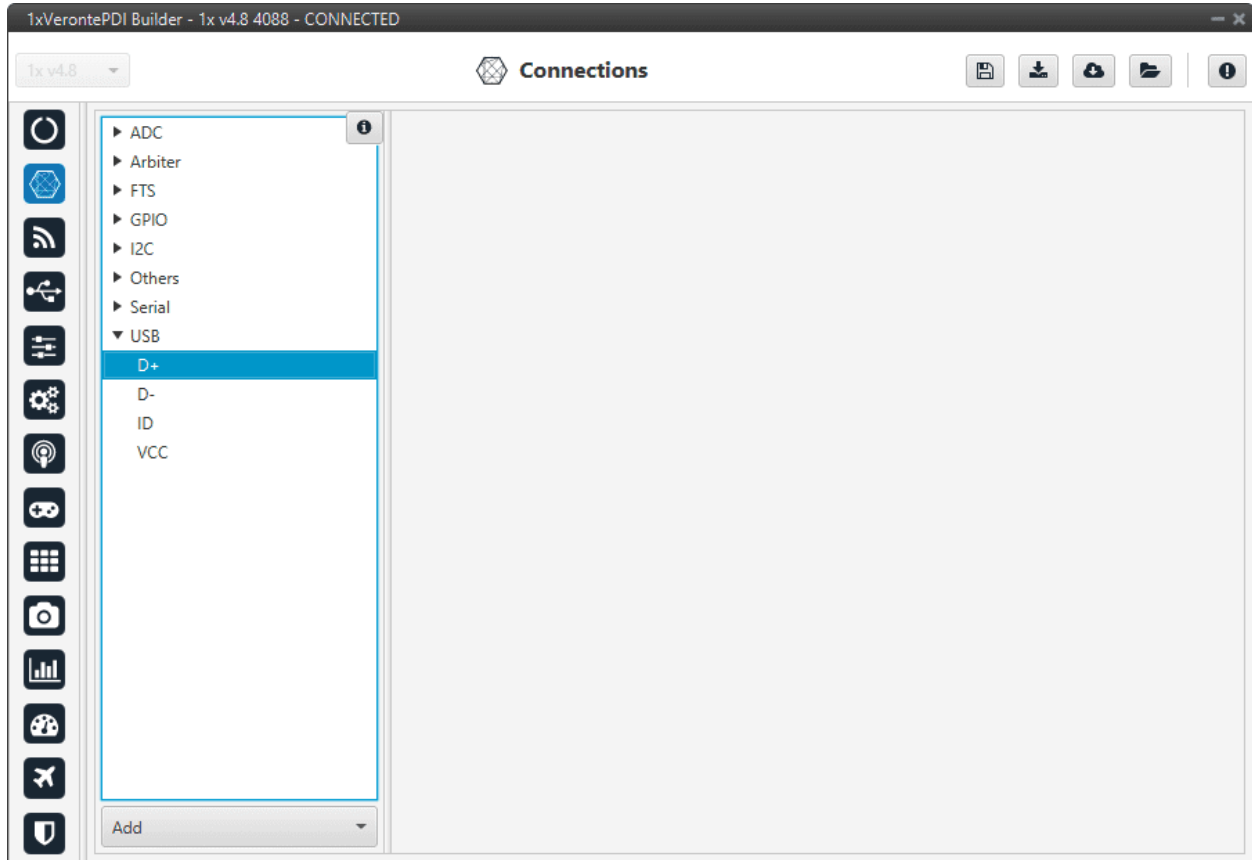


Fig. 27: USB menu

2.3 Sensors

Sensors section allows to configure any sensor connected or internal from 1x autopilot.

2.3.1 Accelerometer

1x autopilot incorporates 3 Inertial Measurement Units (IMUs) that allows the 1x system to measure different variables and that are the main navigation data source. 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):** 1x autopilot uses the internal sensor.

Warning: If the user has a 1x autopilot 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.

- **Integer var sensor 1-2:** 1x autopilot uses a integer value provided by an external sensor.
- **Decimal var sensor 1-2:** 1x autopilot uses a decimal value provided by an external sensor.

Suggestion

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

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

Secondary Accelerometer

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

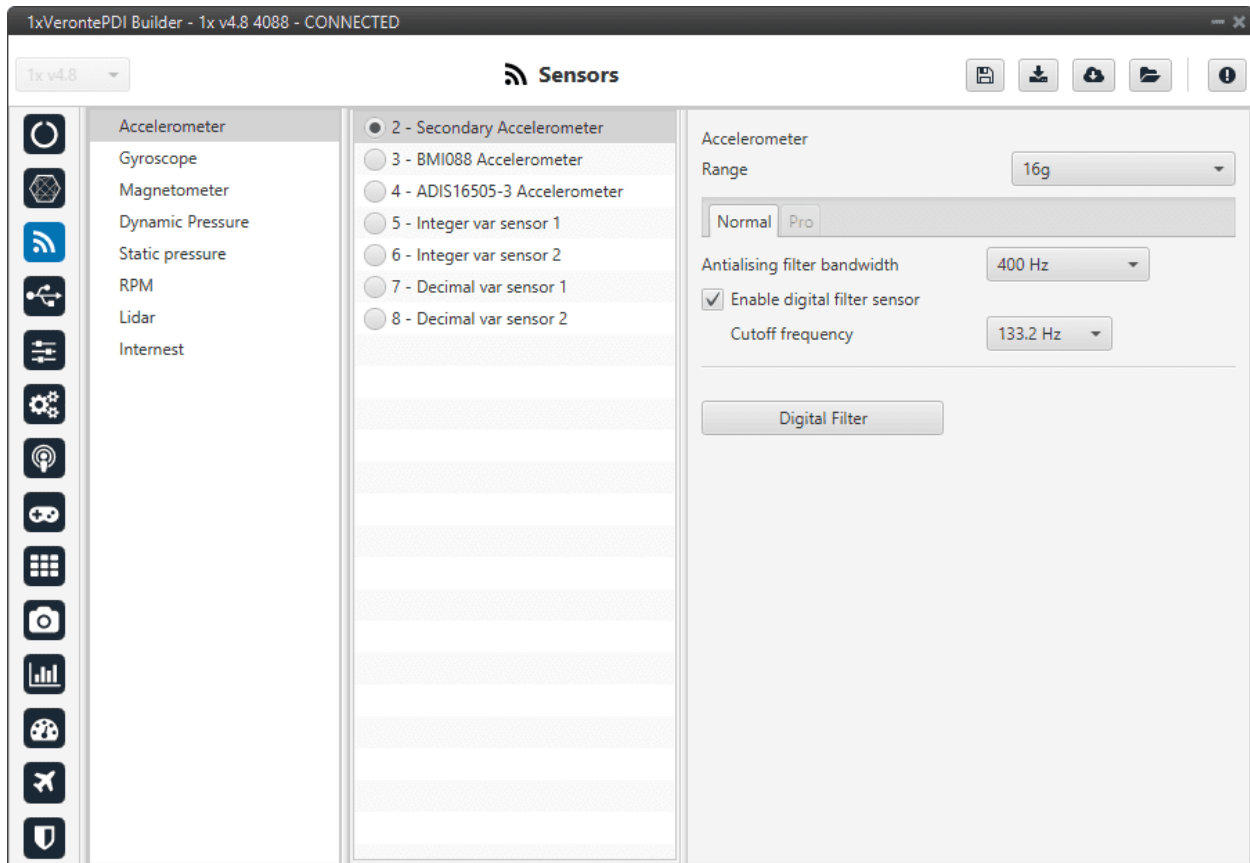


Fig. 28: Secondary Accelerometer menu

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

- **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**.
- **Antialiasing filter bandwidth:** 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.
- **Digital 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**, applied after the hardware filter from the point before.

BMI088 Accelerometer

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

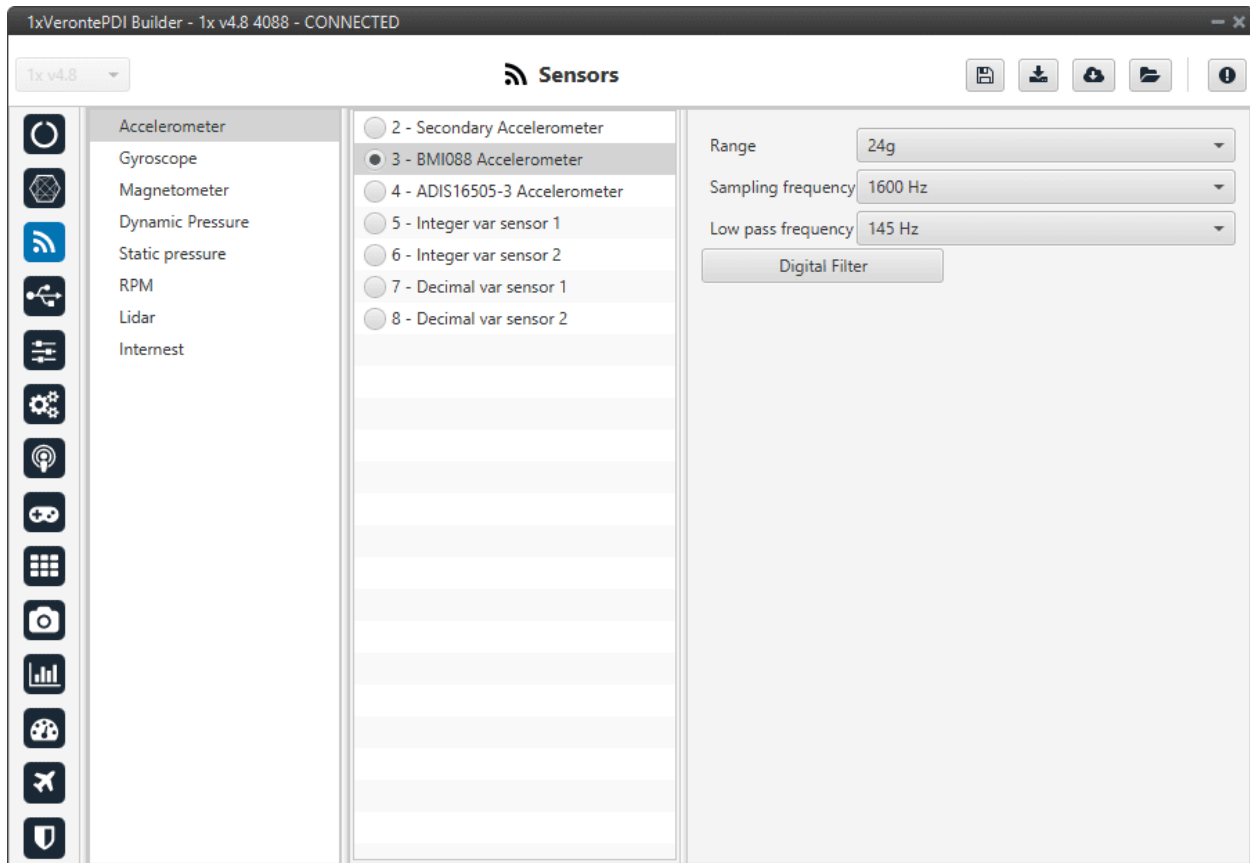


Fig. 29: BMI088 Accelerometer menu

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

- **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, 800Hz 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).

- **Digital 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**, applied after the hardware filter from the point before.

ADIS16505-3 Accelerometer

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

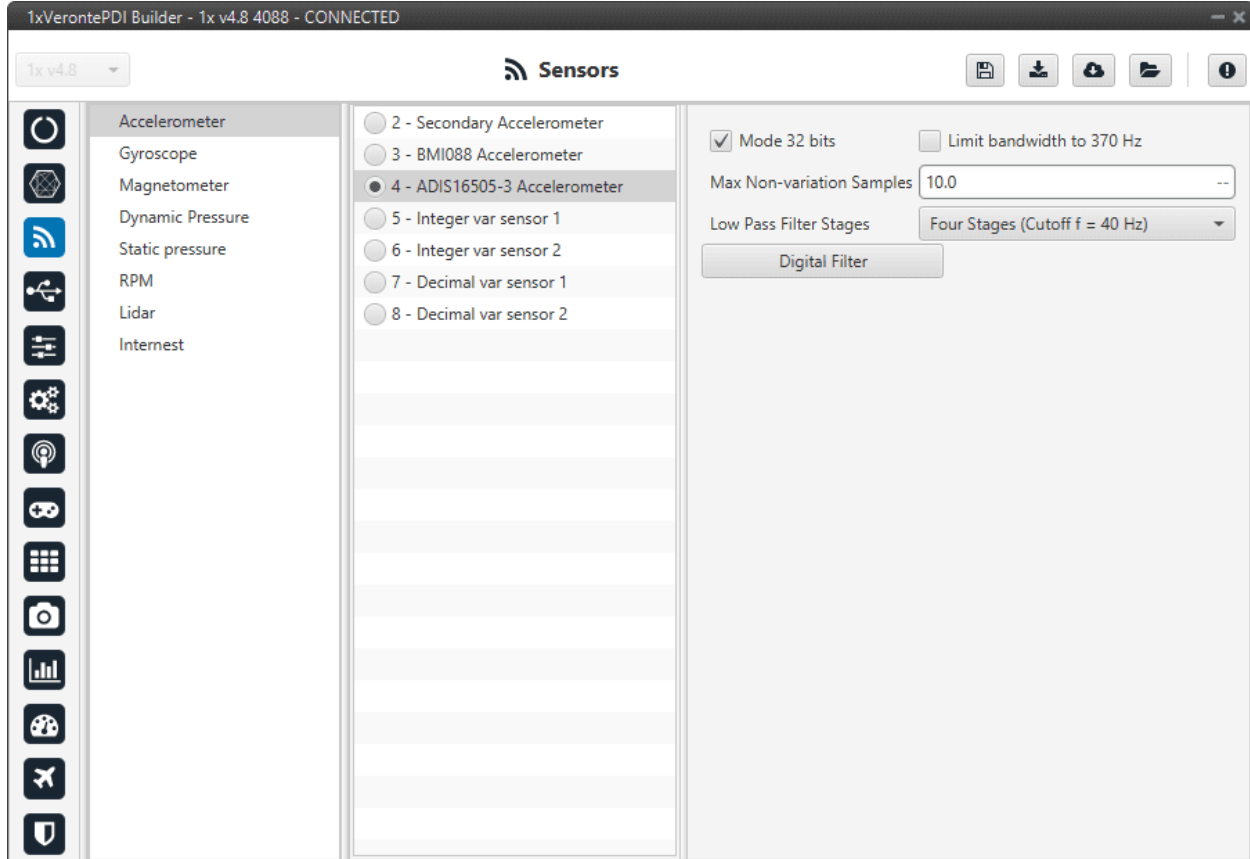


Fig. 30: ADIS16505-3 Accelerometer menu

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

- **Mode 32 bits:** Enable or disable. With 32 bits of precision. We recommend **enabling** it.
- **Limit bandwidth 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 $f=364\text{Hz}$)

- 2 stages (Cutoff $f=165\text{Hz}$)
- 3 stages (Cutoff $f=80\text{Hz}$)
- 4 stages (Cutoff $f=40\text{Hz}$)
- 5 stages (Cutoff $f=20\text{Hz}$)
- 6 stages (Cutoff $f=10\text{Hz}$)

We recommend **4 stages (Cutoff $f=40\text{Hz}$)** option.

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

Warning:

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

Integer var sensor 1-2

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

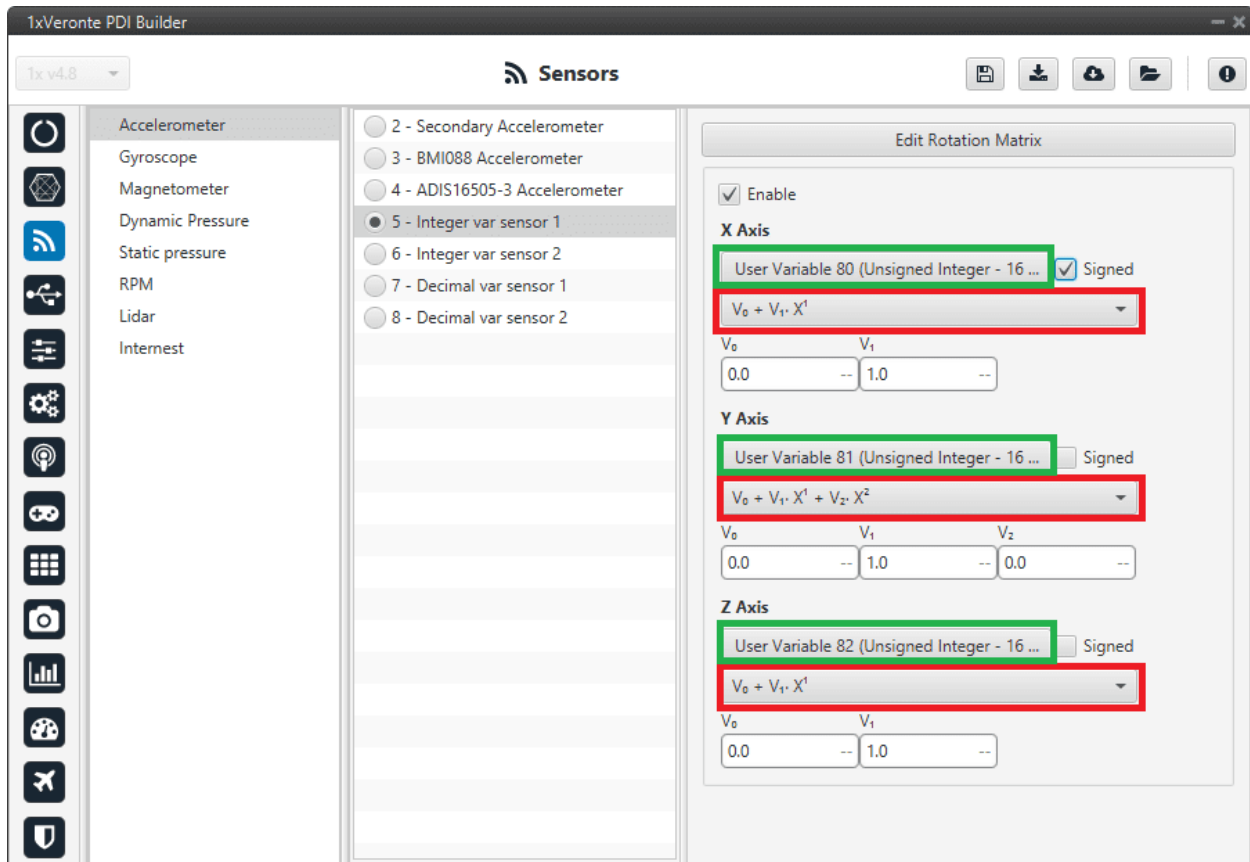


Fig. 31: Integer var accelerometer sensor menu

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:

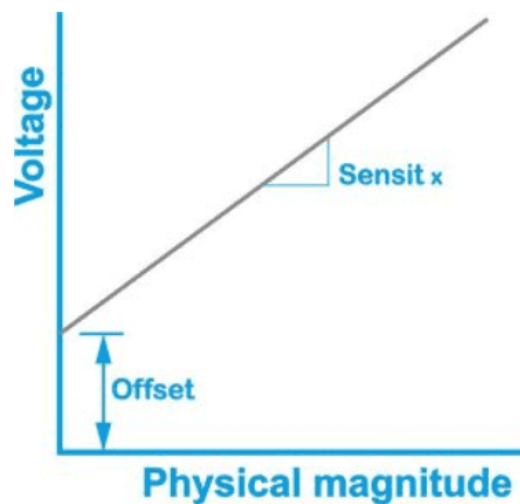


Fig. 32: Linear relation of 2 variables

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 menu of the I/O section*. The configuration will depends on the device in use and its communication protocol.

Warning: **Edit Rotation Matrix** brings the position of the accelerometer inside the 1x autopilot, it must NOT be changed under any circumstance.

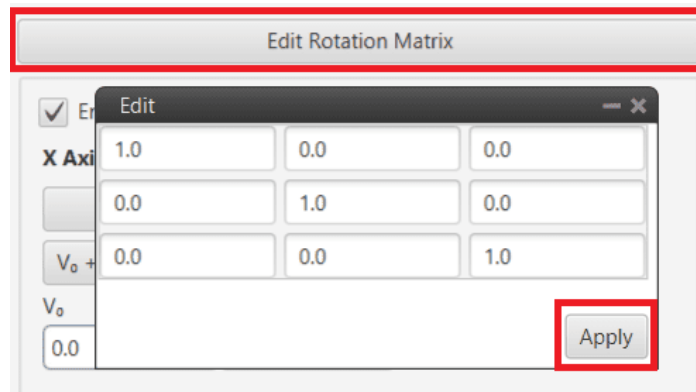


Fig. 33: Rotation matrix

Decimal var sensor 1-2

In this menu, the user selects real variables for each axis (X,Y,Z), these do not require a signal treatment. The process of configuration is similar to the one carried out when configuring a Integer Variable.

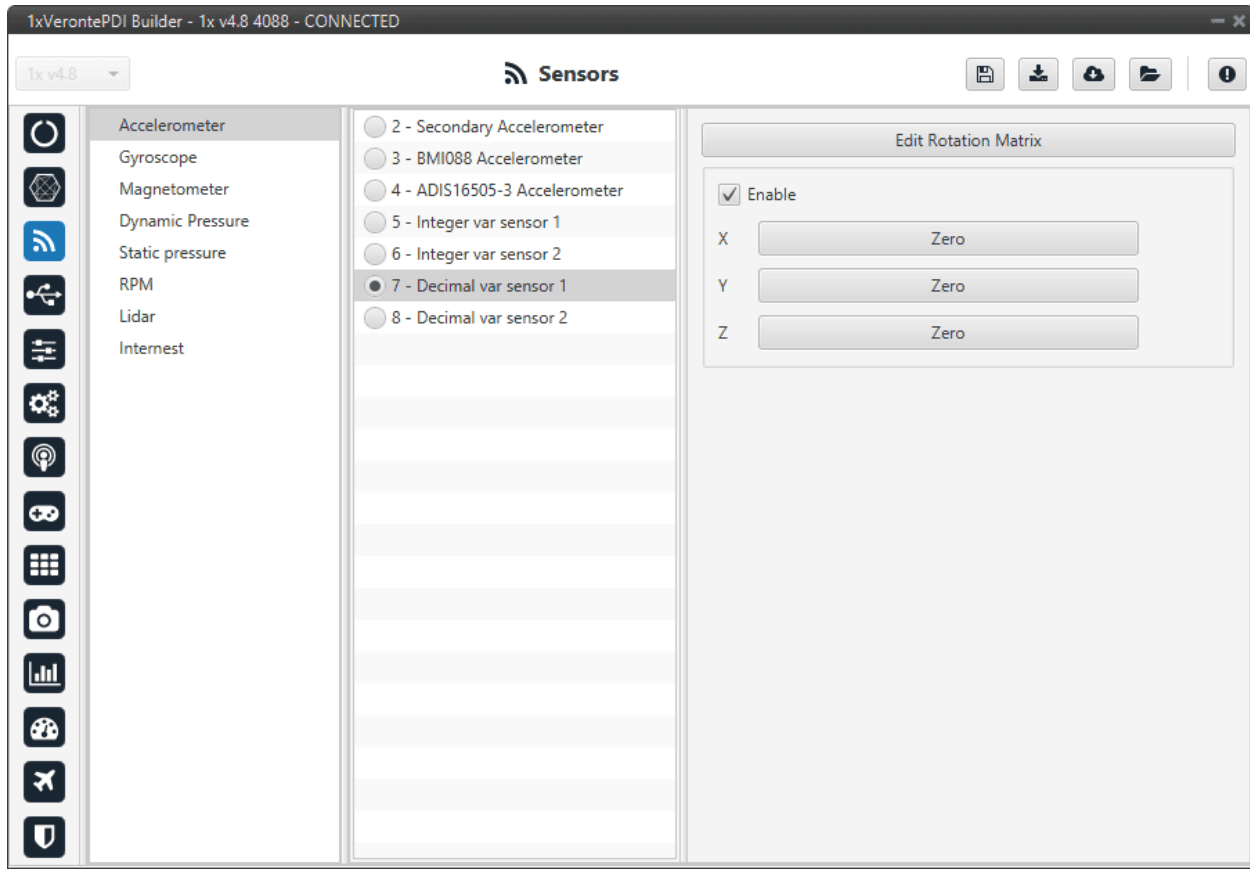


Fig. 34: Decimal var accelerometer sensor menu

2.3.2 Gyroscope

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

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

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

Warning: If the user has a 1x autopilot 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.

- **Integer var sensor 1-2:** 1x autopilot uses a integer value provided by an external sensor.
- **Decimal var sensor 1-2:** 1x autopilot uses a decimal value provided by an external sensor.

Suggestion

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

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

Secondary Gyroscope

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

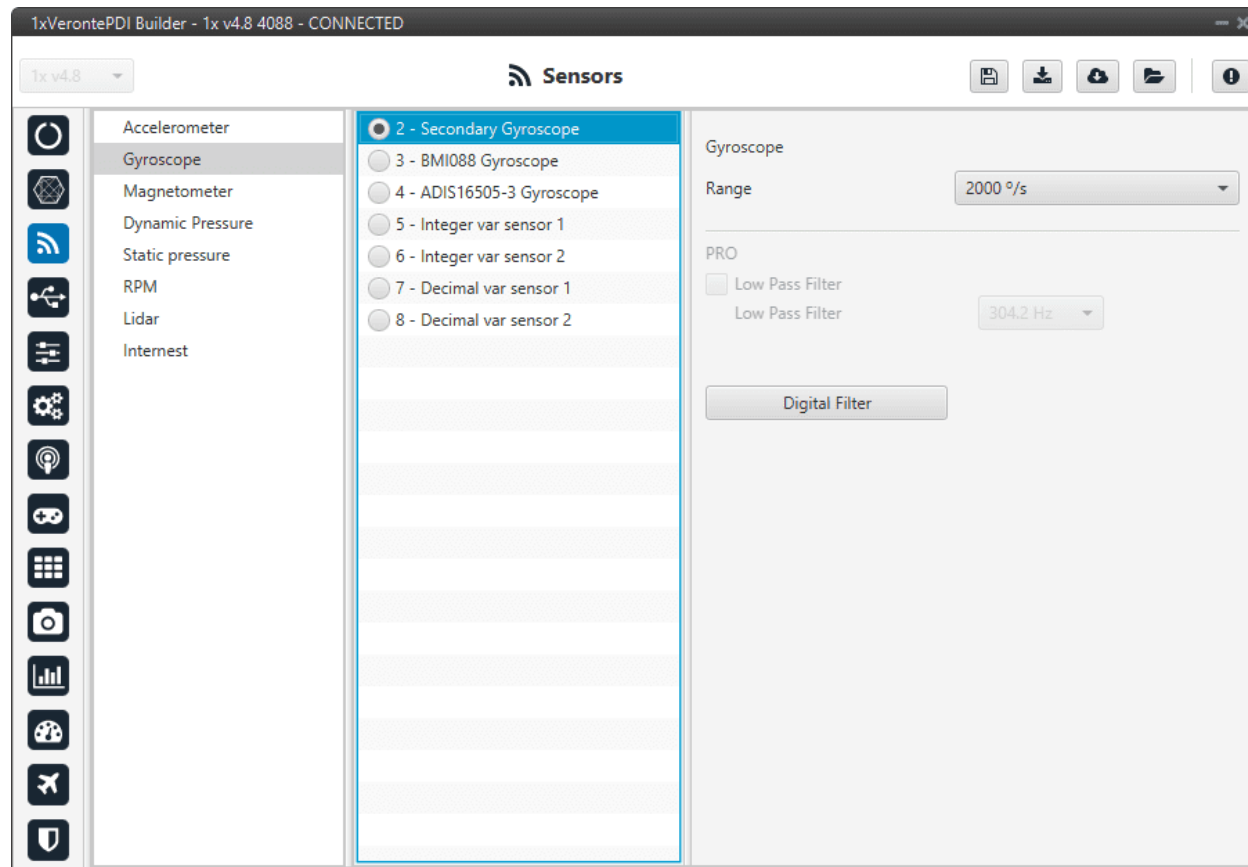


Fig. 35: Secondary Gyroscope menu

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

- **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**.
- **Digital 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**.

BMI088 Gyroscope

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

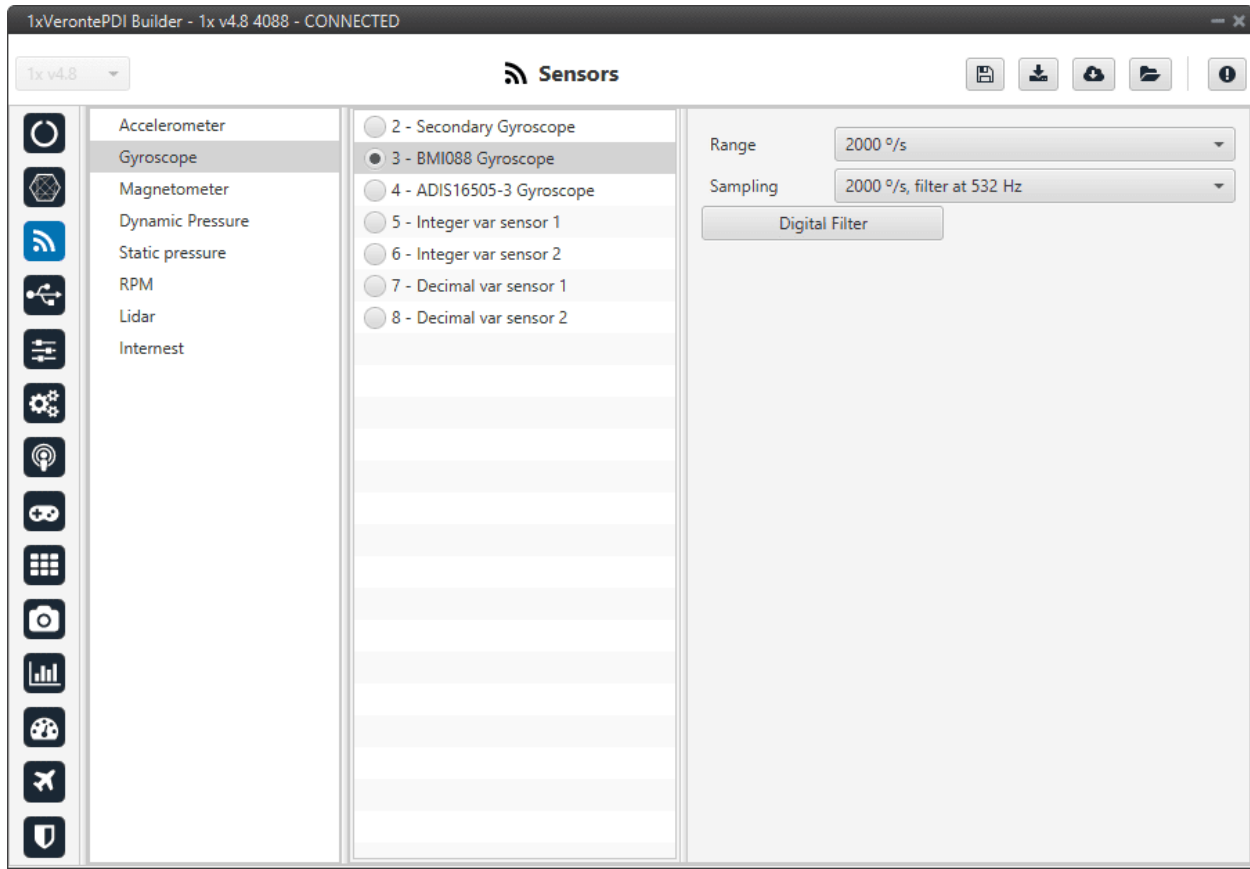


Fig. 36: BMI088 Gyroscope menu

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

- **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.
- **Digital 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**.

ADIS16505-3 Gyroscope

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

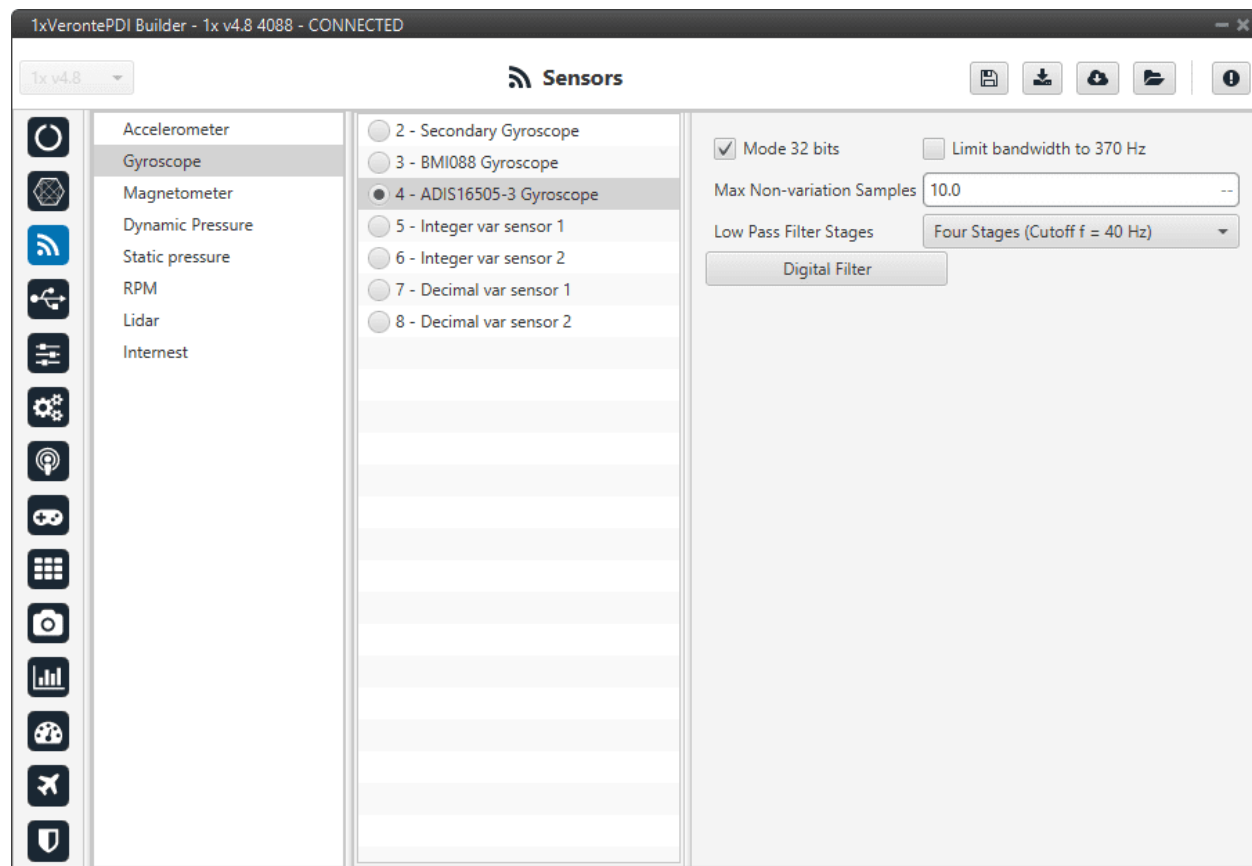


Fig. 37: ADIS16505-3 Gyroscope menu

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

- **Mode 32 bits:** Enable or disable. With 32 bits of precision. We recommend **enabling** it.
- **Limit bandwidth 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:** The options available are No filter, 1 stage (Cutoff $f=364\text{Hz}$), 2 stages (Cutoff $f=165\text{Hz}$), 3 stages (Cutoff $f=80\text{Hz}$), 4 stages (Cutoff $f=40\text{Hz}$), 5 stages (Cutoff $f=20\text{Hz}$) and 6 stages (Cutoff $f=10\text{Hz}$). We recommend **4 stages (Cutoff $f=40\text{Hz}$)**.
- **Digital filter:** Enables a low pass filter which its cutoff frequency is configured manually, allowing the user to input any desired value in Hz. We recommend **disabling** it.

Warning: It is not recommended flying without a filter.

Integer var sensor 1-2

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

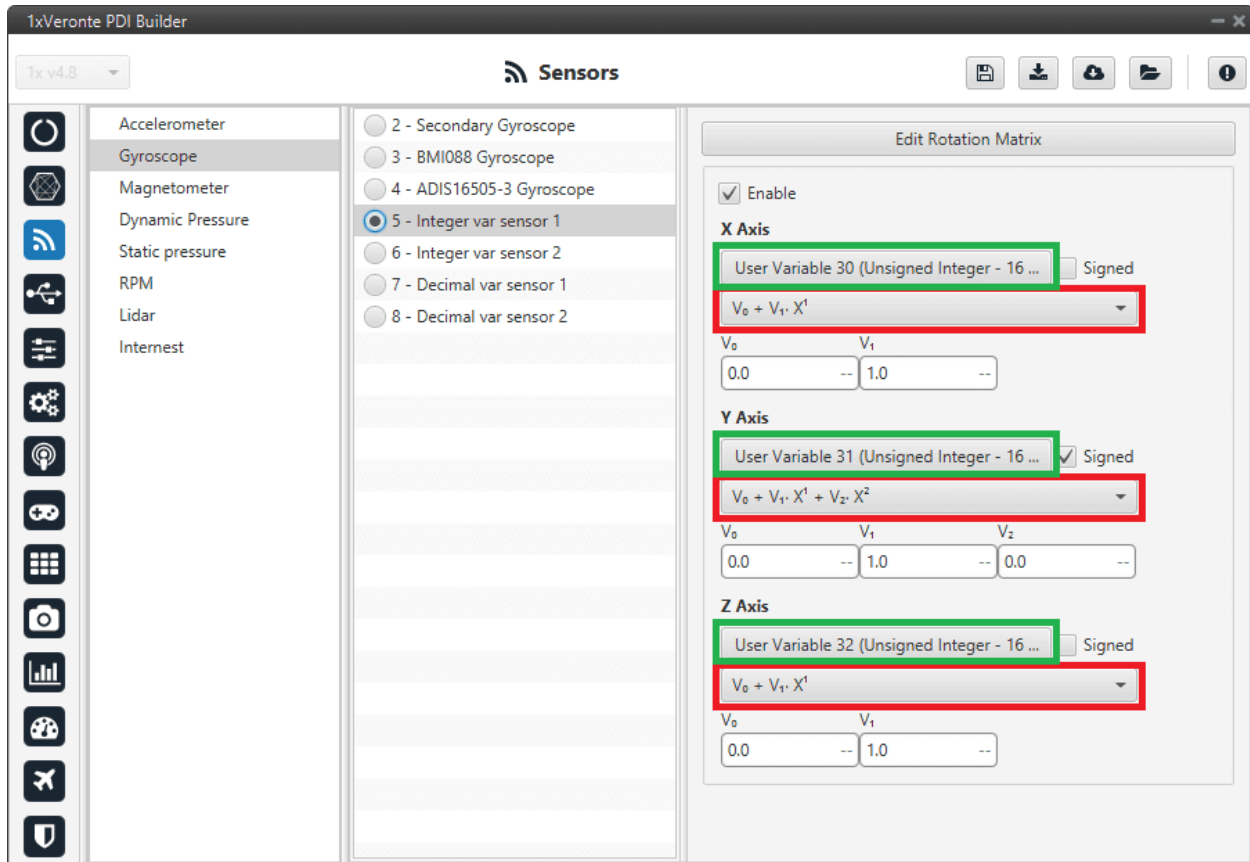


Fig. 38: Integer var gyroscope sensor menu

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:

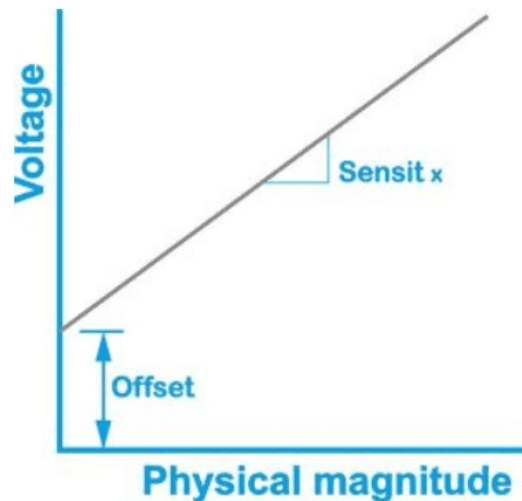


Fig. 39: Linear relation of 2 variables

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 menu of the I/O section*. The configuration will depend on the device in use and its communication protocol.

Warning: **Edit Rotation Matrix** brings the position of the gyroscope inside the 1x Autopilot, it must NOT be changed under any circumstance.

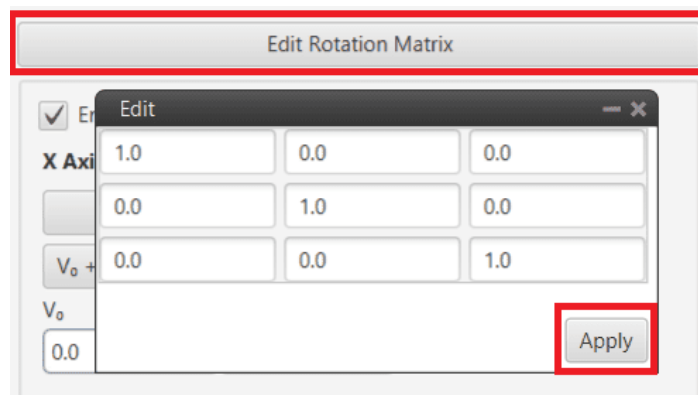


Fig. 40: Rotation matrix

In this menu, the user selects real variables for each axis (X,Y,Z), these do not require a signal treatment. The process of configuration is similar to the one carried out when configuring a Integer Variable.

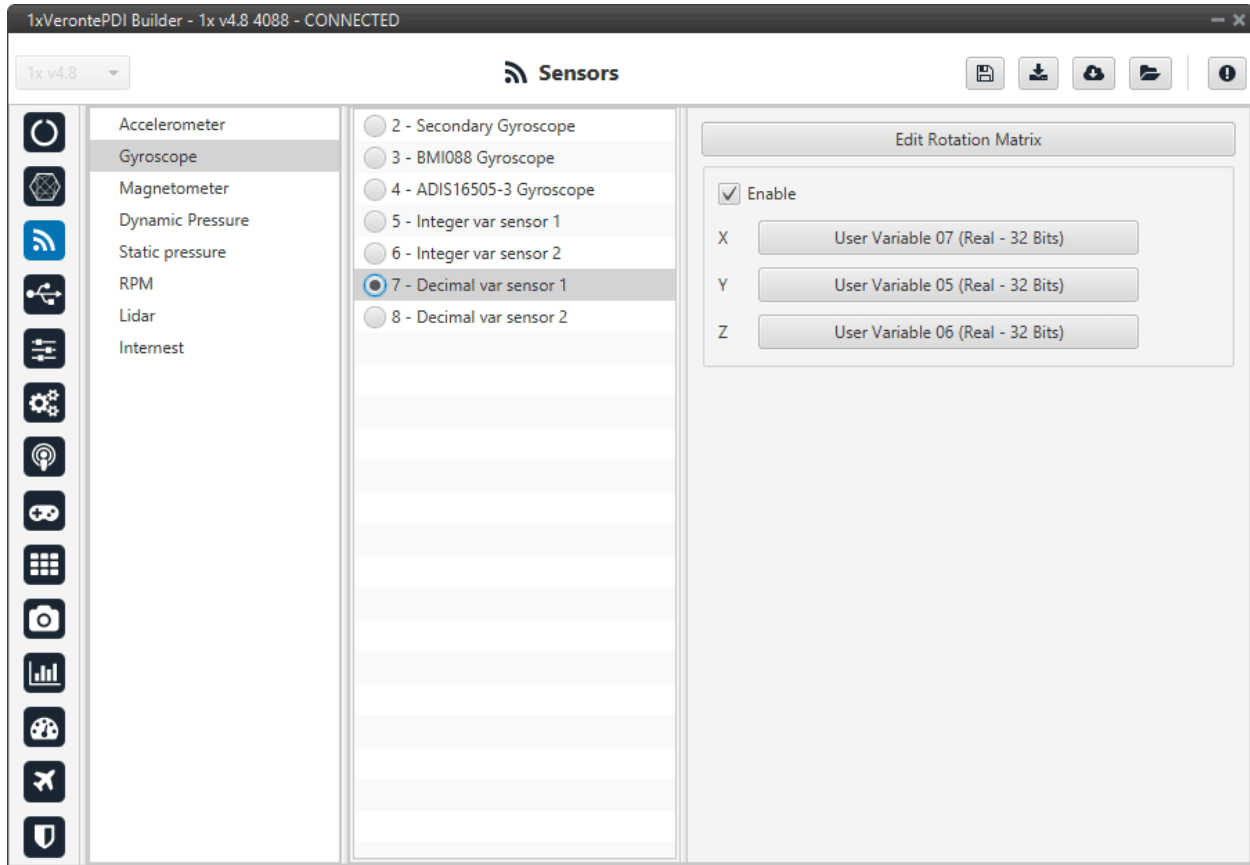


Fig. 41: Decimal var gyroscope sensor menu

2.3.3 Magnetometer

1x autopilot incorporates an internal magnetometer that allows the 1x System to measure the magnetic field.

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

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

Warning: If the user has a 1x autopilot with hardware **version 4.5**, **Internal RM3100** Magnetometer is **not available**.

- **Integer var sensor 1-2:** 1x autopilot uses a integer value provided by a no-integrated external sensor.
- **Decimal var sensor 1-2:** 1x autopilot uses a decimal value provided by a no-integrated external sensor.
- **External (HMR2300/LIS3MDL/HSCDTD008A/MMC5883MA/RM3100):** 1x autopilot uses the information from one of the compatible external magnetometers.

For this sensor, it is **possible** to have **more than one selected** for the navigation algorithm, so the user will have **redundancy** in magnetometer sensor.

Suggestion

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 1x autopilot 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 section, the user can only modify some configurable parameters, the selection of the magnetometer and the configuration of the **Variance** are done in *Magnetometer sensor - Block Programs* section.

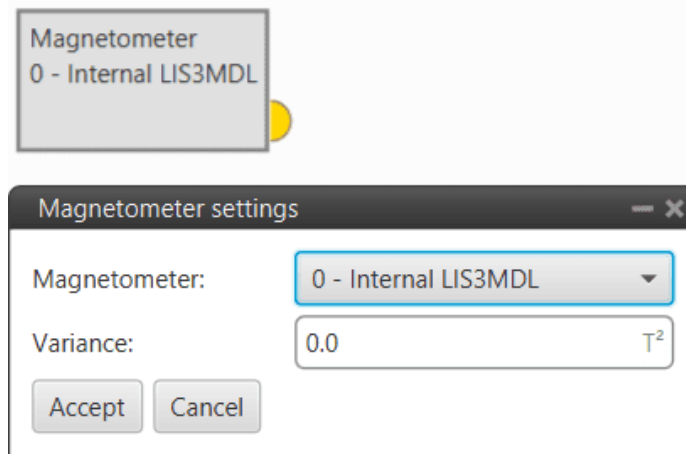


Fig. 42: Magnetometer block

Internal LIS3MDL/MMC5883MA/RM3100

This menu available displays the possible parameters that can be configured for the internal Magnetometer.

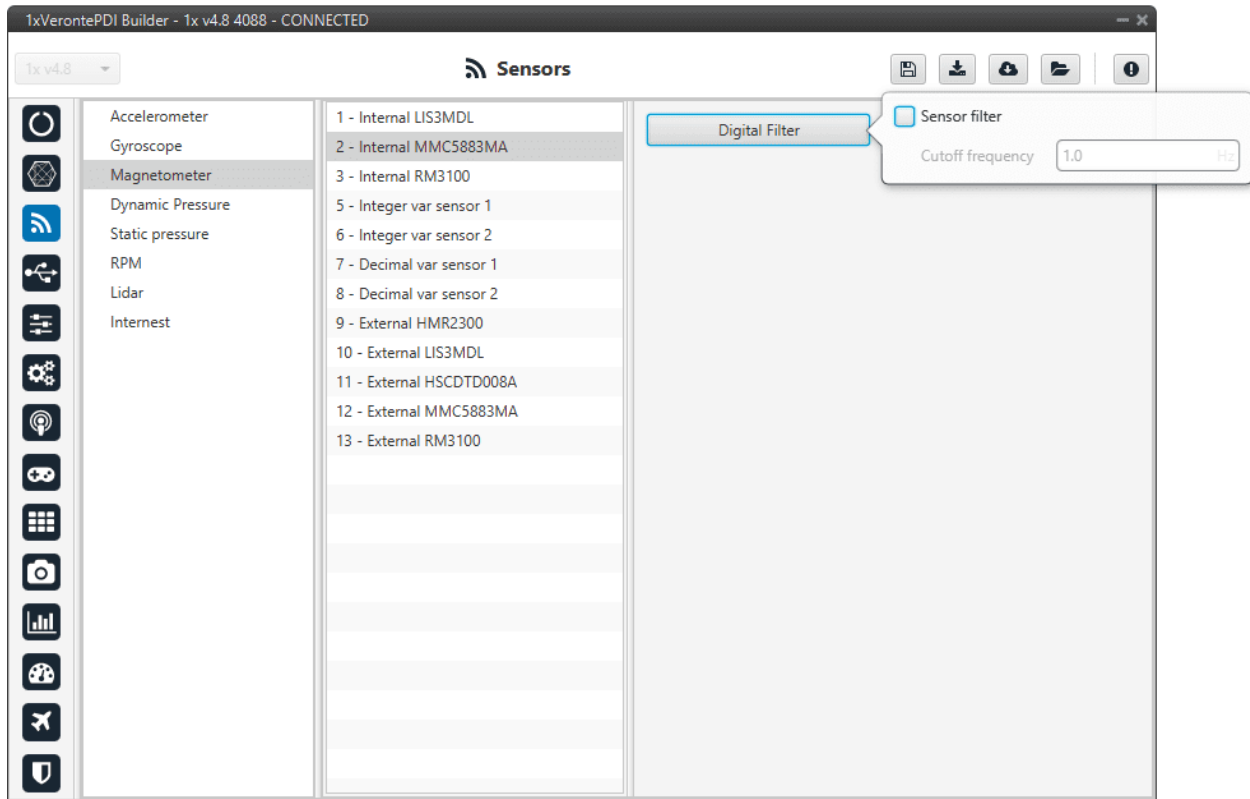


Fig. 43: Internal Magnetometer menu

In this menu it is possible to modify different options regarding the digital filter:

- **Digital 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 1-2

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

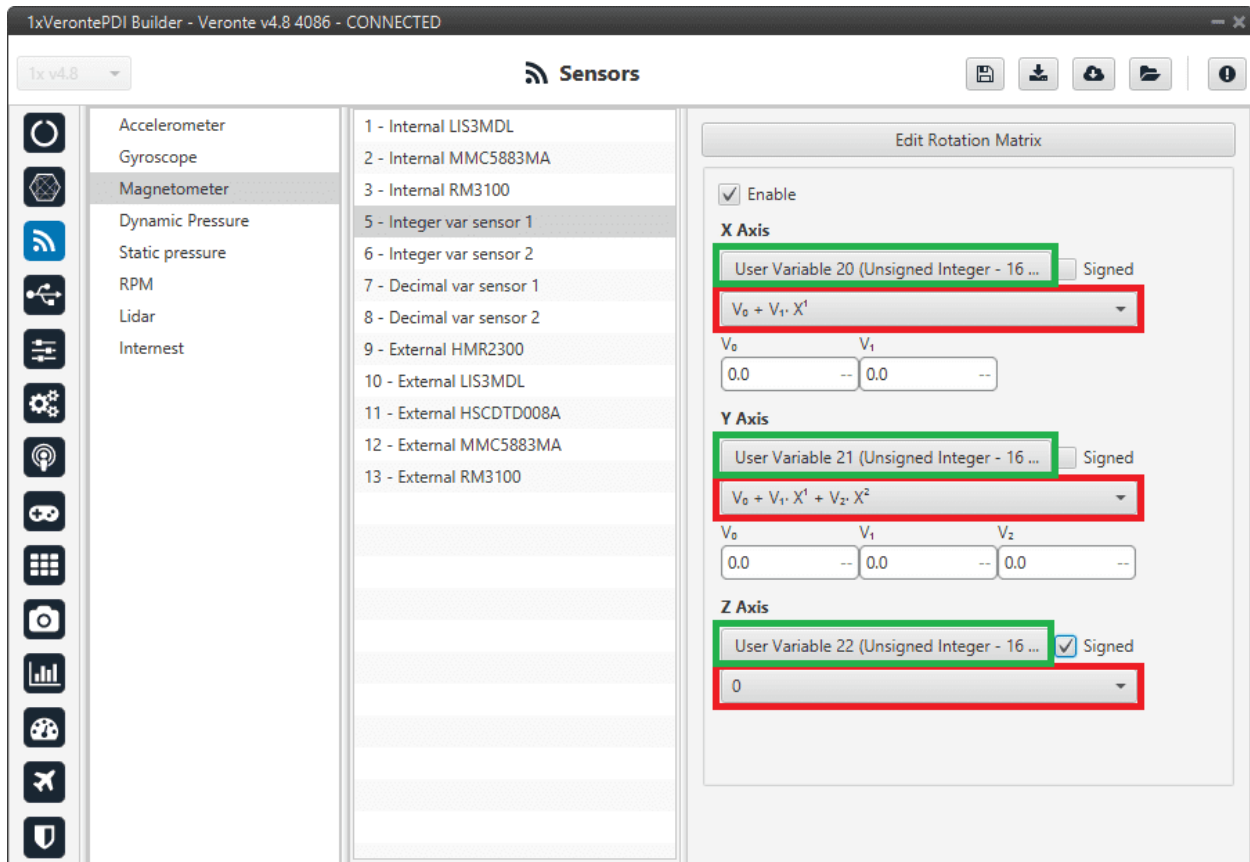


Fig. 44: Integer var magnetometer sensor menu

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:

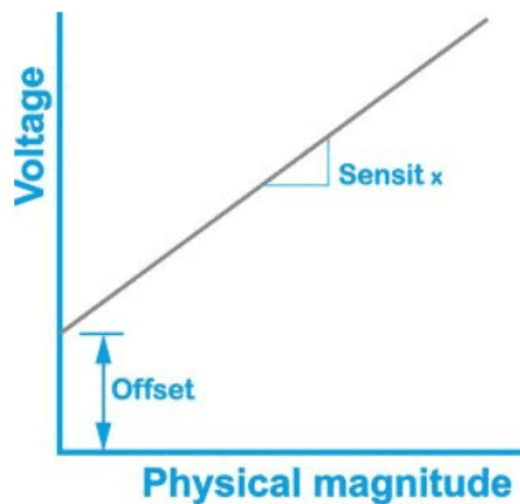


Fig. 45: Linear relation of 2 variables

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 menu of the I/O section*. The configuration will depends on the device in use and its communication protocol.

Warning: **Edit Rotation Matrix** brings the position of the magnetometer inside the 1x Autopilot, it must NOT be changed under any circumstance.

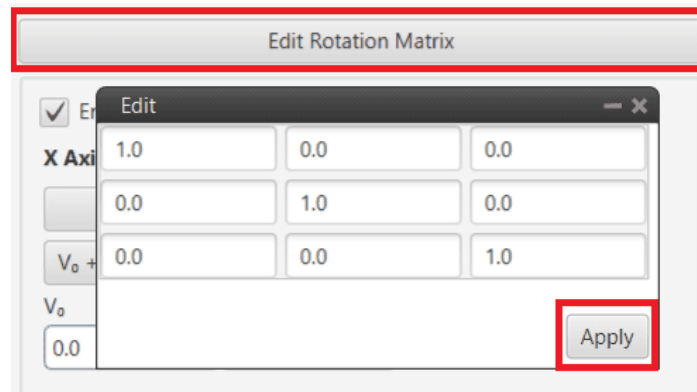


Fig. 46: Rotation matrix

Decimal var sensor 1-2

In this menu, the user selects real variables for each axis (X,Y,Z), these do not require a signal treatment. The process of configuration is similar to the one carried out when configuring a Integer Variable.

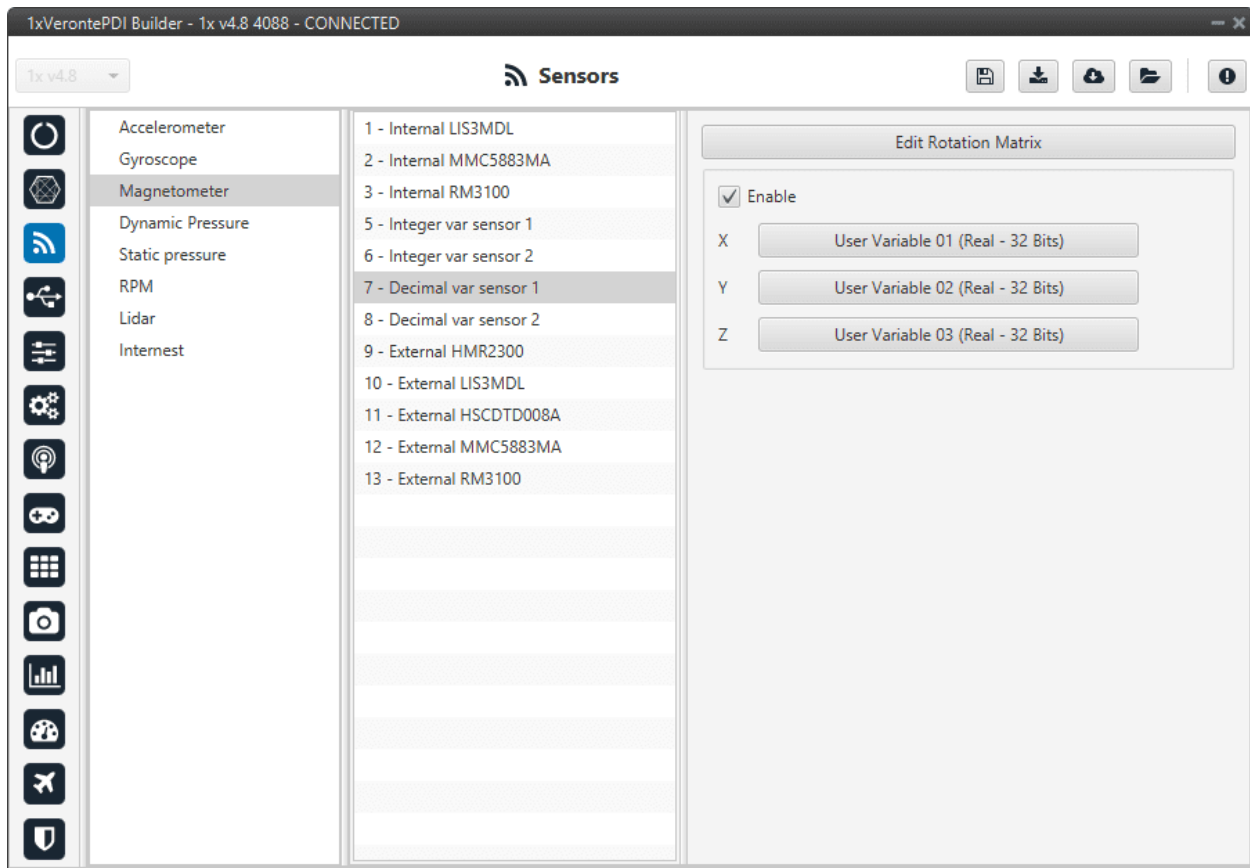


Fig. 47: Decimal var magnetometer sensor menu

External HMR2300

1x autopilot has been designed to have compatibility with this external magnetometer.

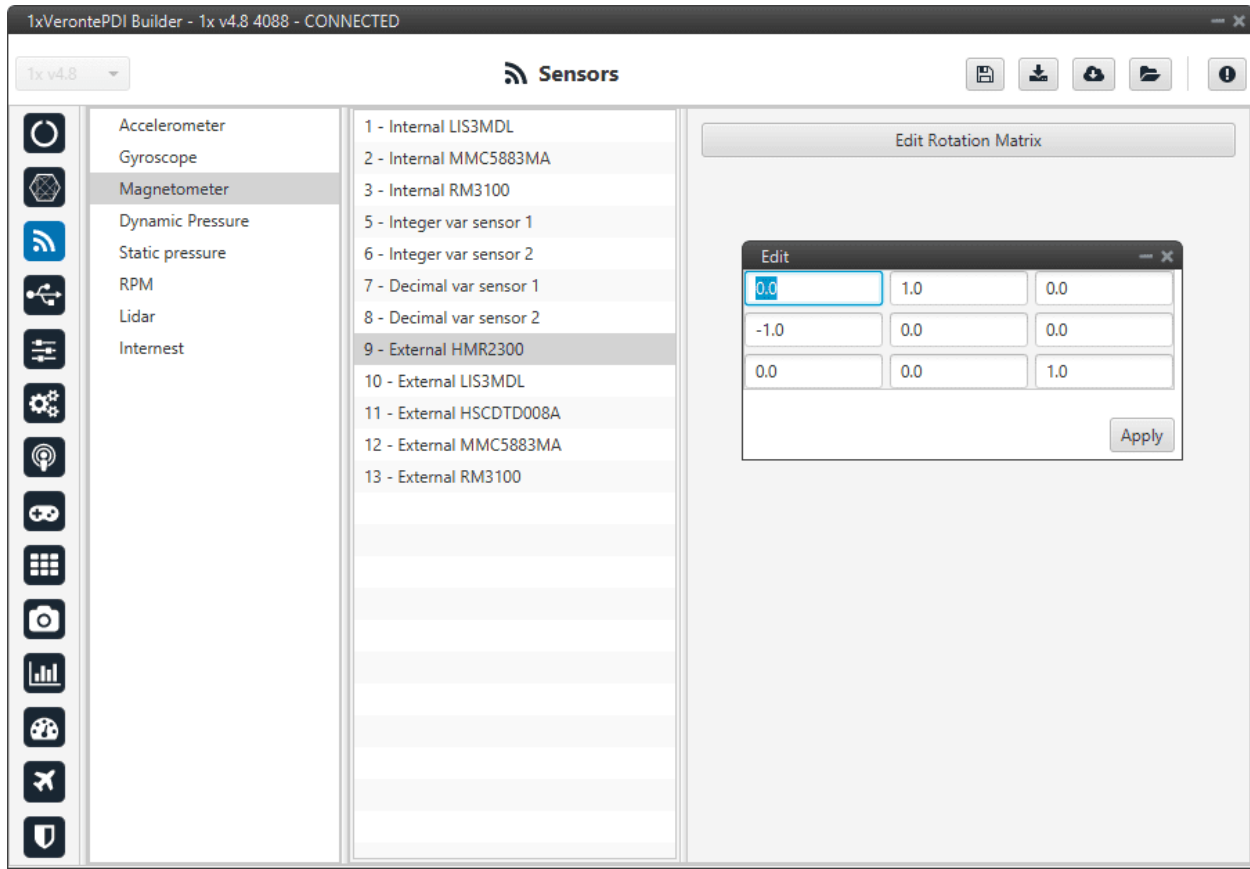


Fig. 48: External HMR2300 Magnetometer menu

This magnetometer has no filters configurable, only the option Edit Rotation Matrix to set the orientation of it.
On the other hand, the connection to the serial port has to be configured in the *I/O Setup section of the I/O menu*.

External LIS3MDL/HSCDTD008A/MMC5883MA/RM3100

1x autopilot has been designed to have compatibility with this external magnetometer.

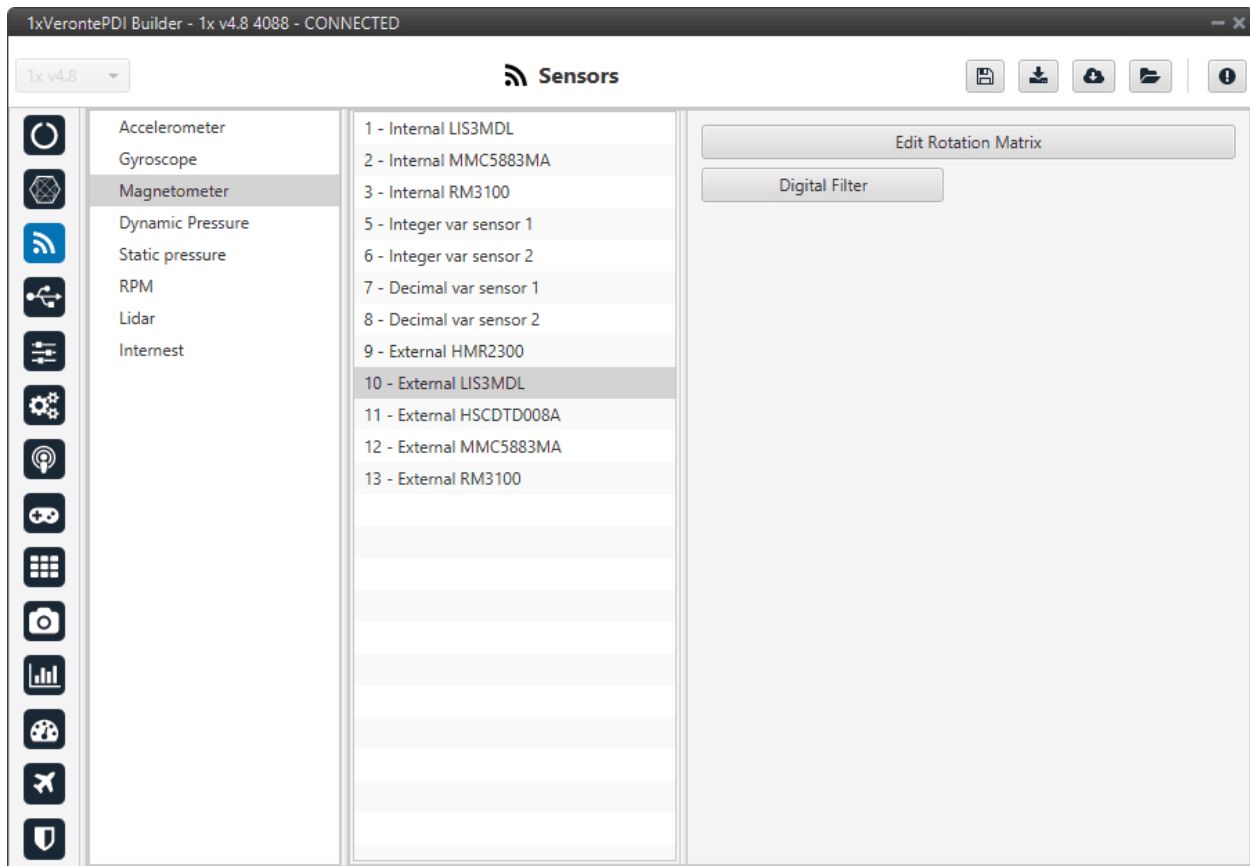


Fig. 49: External Magnetometer menu

The user can modify the following parameters:

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

2.3.4 Dynamic Pressure

1x autopilot 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 menu allows the user to configure a Dynamic Pressure sensor input in 1x.

The user can choose between 3 types of source:

- **Internal:** 1x autopilot uses the internal sensor.
- **Integer var sensor 1-2:** 1x autopilot uses a integer value provided by an external sensor.
- **Decimal var sensor 1-2:** 1x autopilot uses a decimal value provided by an external sensor.

Navigation

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

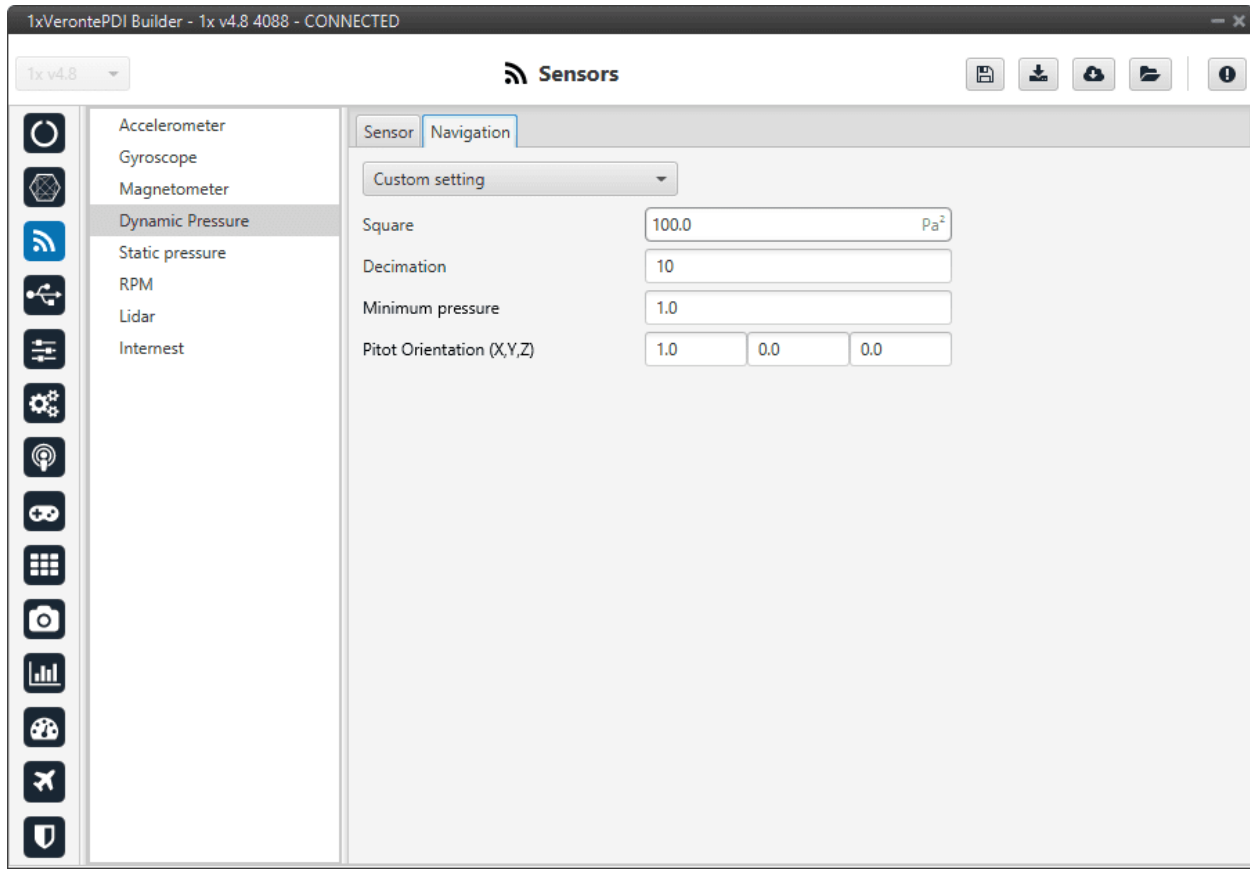


Fig. 50: Dynamic Pressure menu - Navigation parameters

- **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 defined 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.
- **Minimum pressure:** Minimum pressure readable from the sensor.
- **Pitot Orientation:** Vector defining the Pitot orientation on the platform.

Internal

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

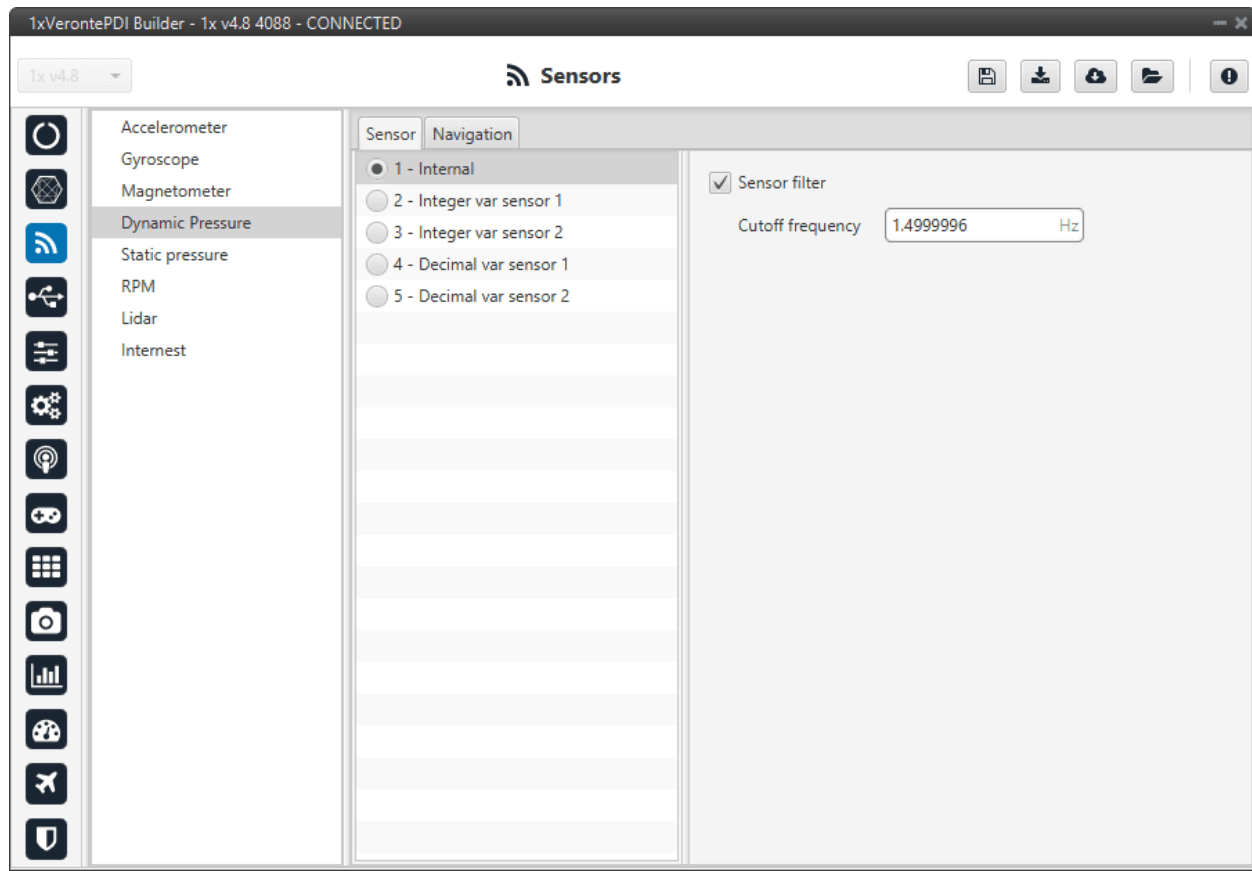


Fig. 51: **Internal dynamic pressure sensor menu**

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

- **Digital 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 1-2

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

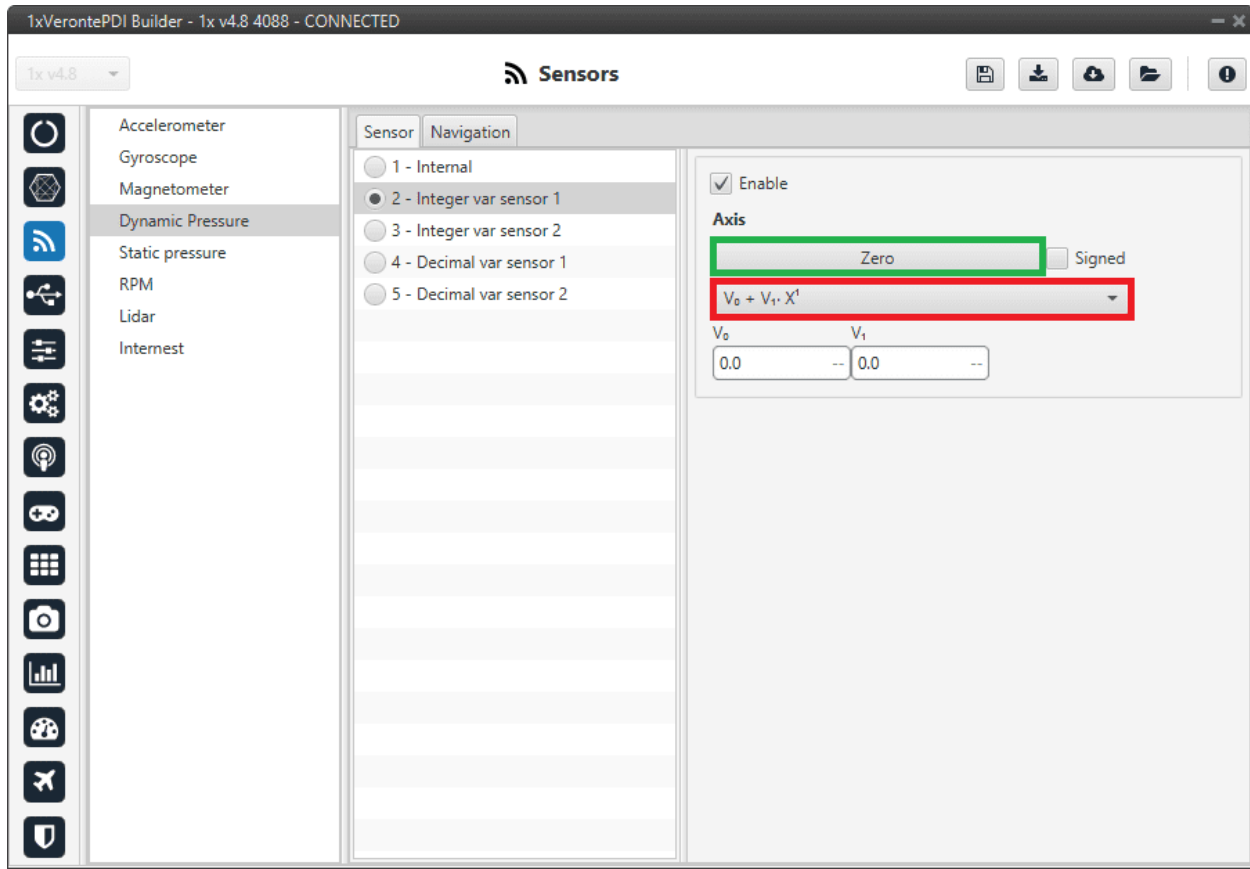


Fig. 52: Integer var dynamic pressure sensor menu

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:

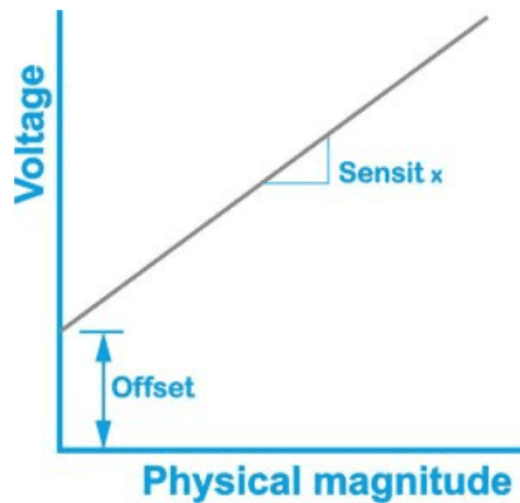


Fig. 53: Linear relation of 2 variables

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 menu of the I/O section*. The configuration will depends on the device in use and its communication protocol.

Decimal var sensor 1-2

In this menu, 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.

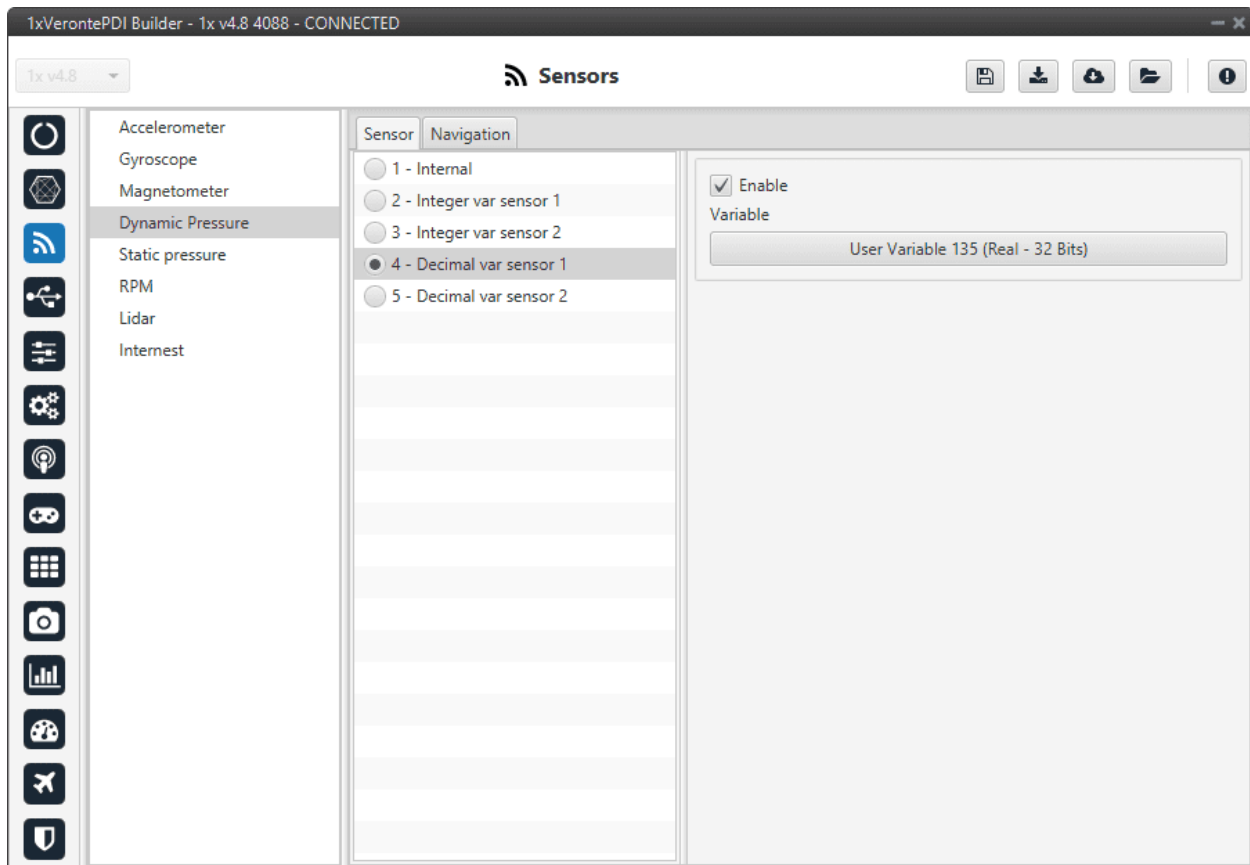


Fig. 54: Decimal var dynamic pressure sensor menu

2.3.5 Static Pressure

1x autopilot 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 menu allows the user to configure a static pressure sensor input in 1x.

The user can choose between 3 types of source:

- **Internal (HSC/MS56/DPS310):** 1x autopilot uses the internal sensor.
- **Integer var sensor 1-2:** 1x autopilot uses a integer value provided by an external sensor.
- **Decimal var sensor 1-2:** 1x autopilot uses a decimal value provided by an external sensor.

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

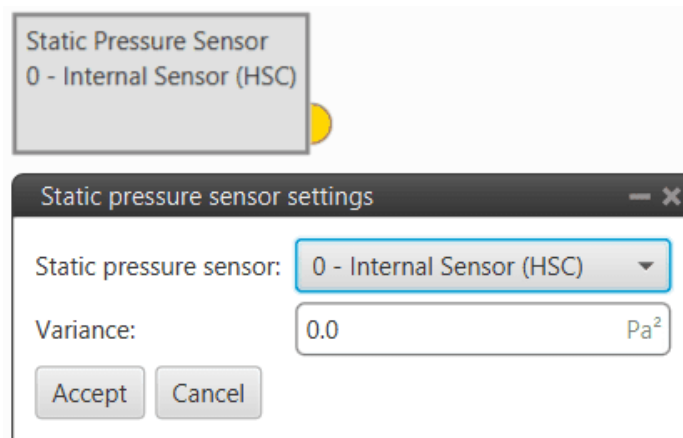


Fig. 55: Static Pressure sensor block

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 for the selected Static Pressure sensor.

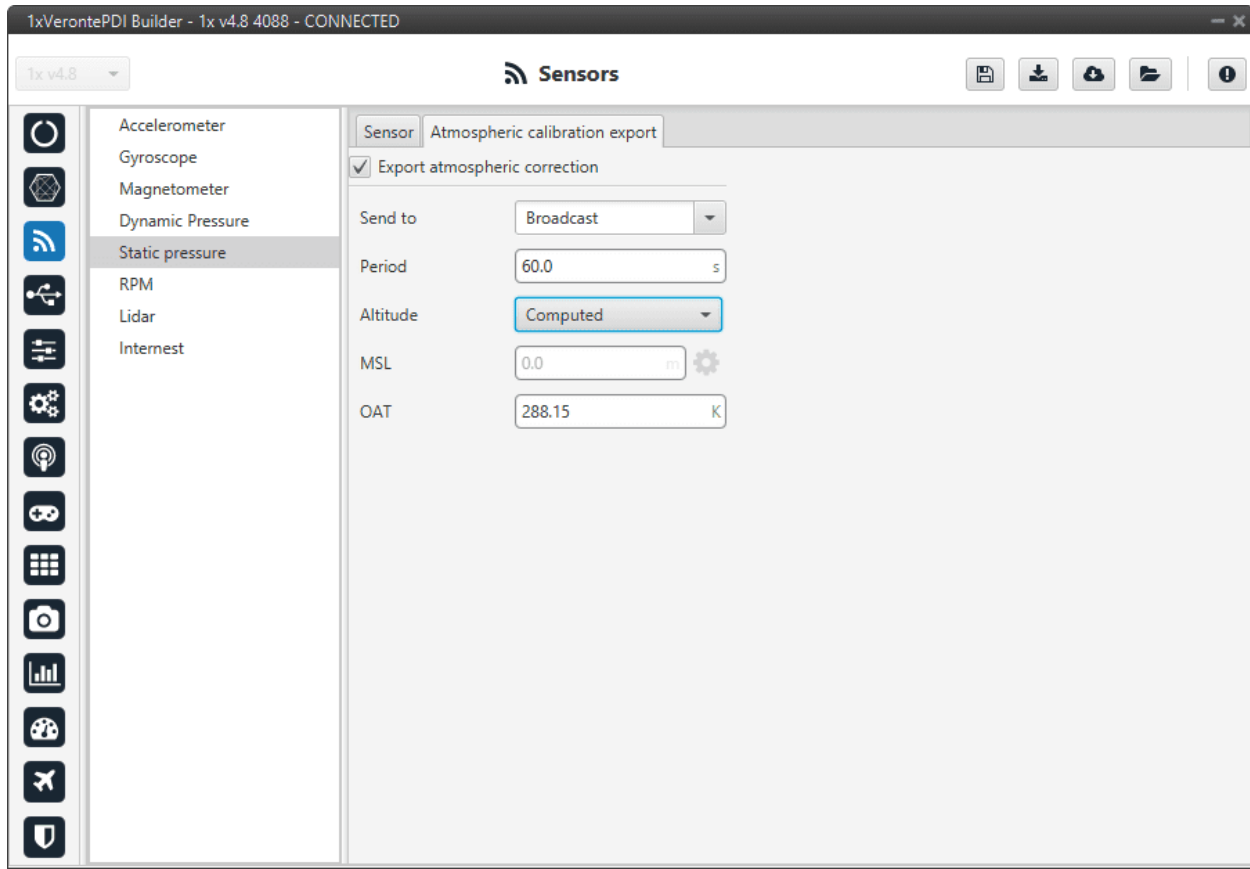


Fig. 56: Static Pressure menu - Atmospheric calibration export parameters

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

This menu has 5 parameters that need to be configured:

- **Send to:** Points where the correction will be sent. It can be *another 1x unit* (Address of another 1x autopilot), *Broadcast* or *App 2*.
- **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), 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.

Internal Sensor HSC/MS56/DPS310

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

1x autopilot has embedded 3 digital static pressure sensors: the **DPS310**, the **MS56** and the **HSC**. See more information on the pressure ports in [Hardware installation - Electrical](#).

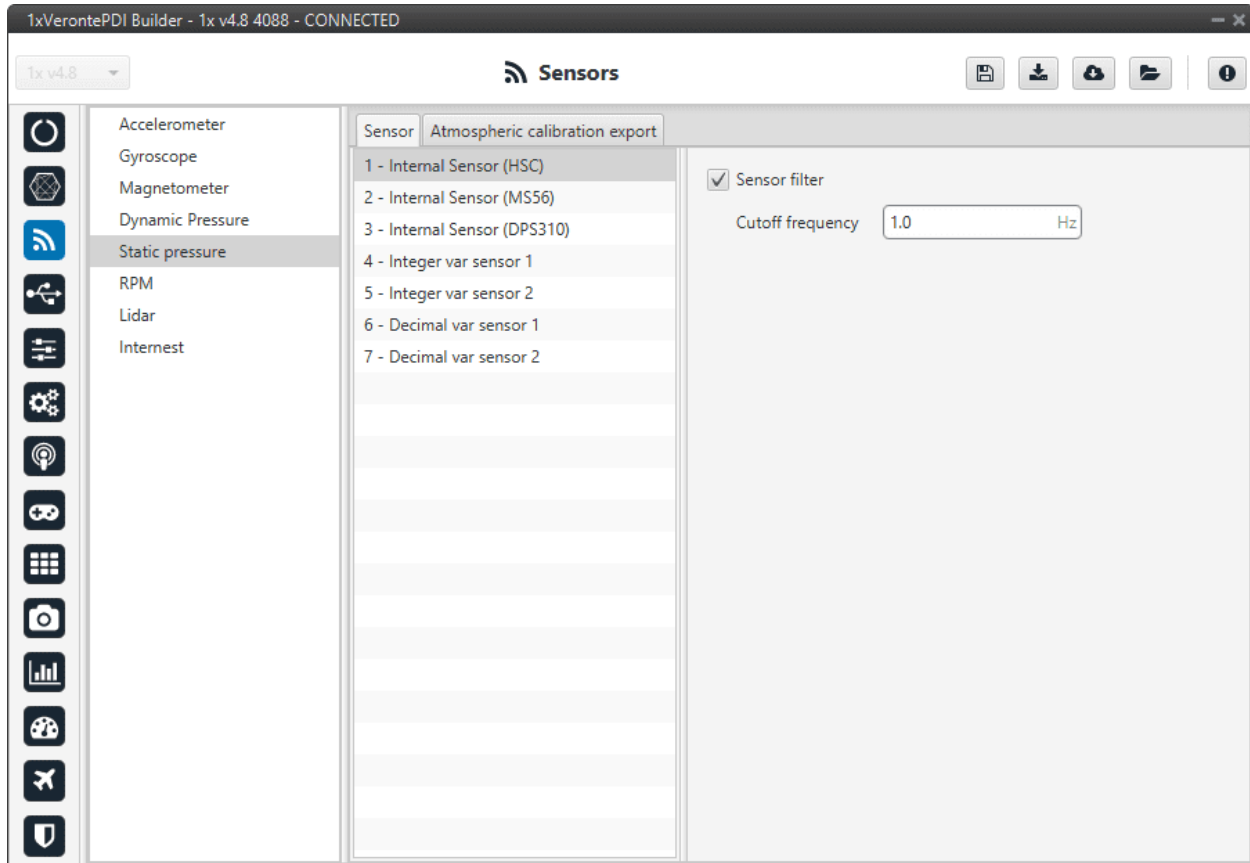


Fig. 57: Internal Static Pressure sensor menu

In this menu 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 1-2

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

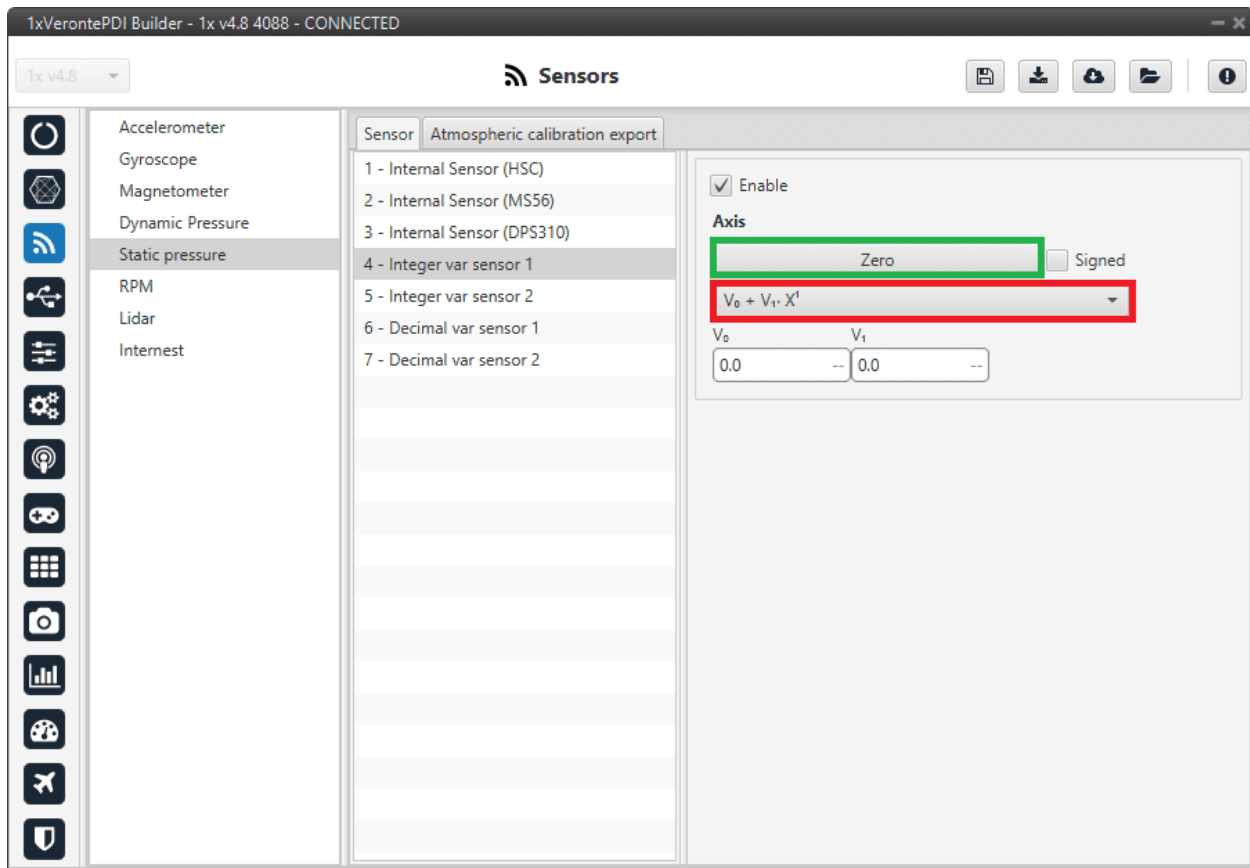


Fig. 58: Integer var static pressure sensor menu

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:

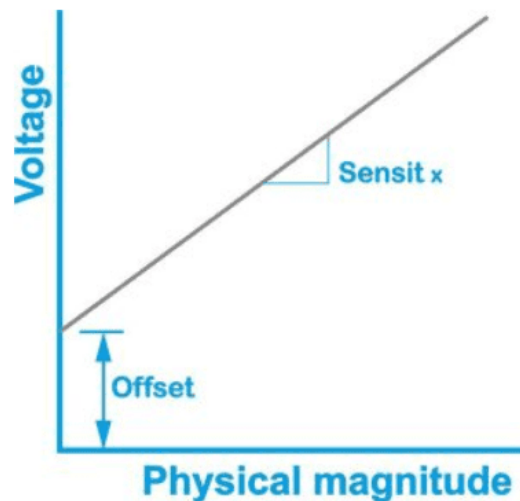


Fig. 59: Linear relation of 2 variables

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 menu of the I/O section*. The configuration will depends on the device in use and its communication protocol.

Decimal var sensor 1-2

In this menu, 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.

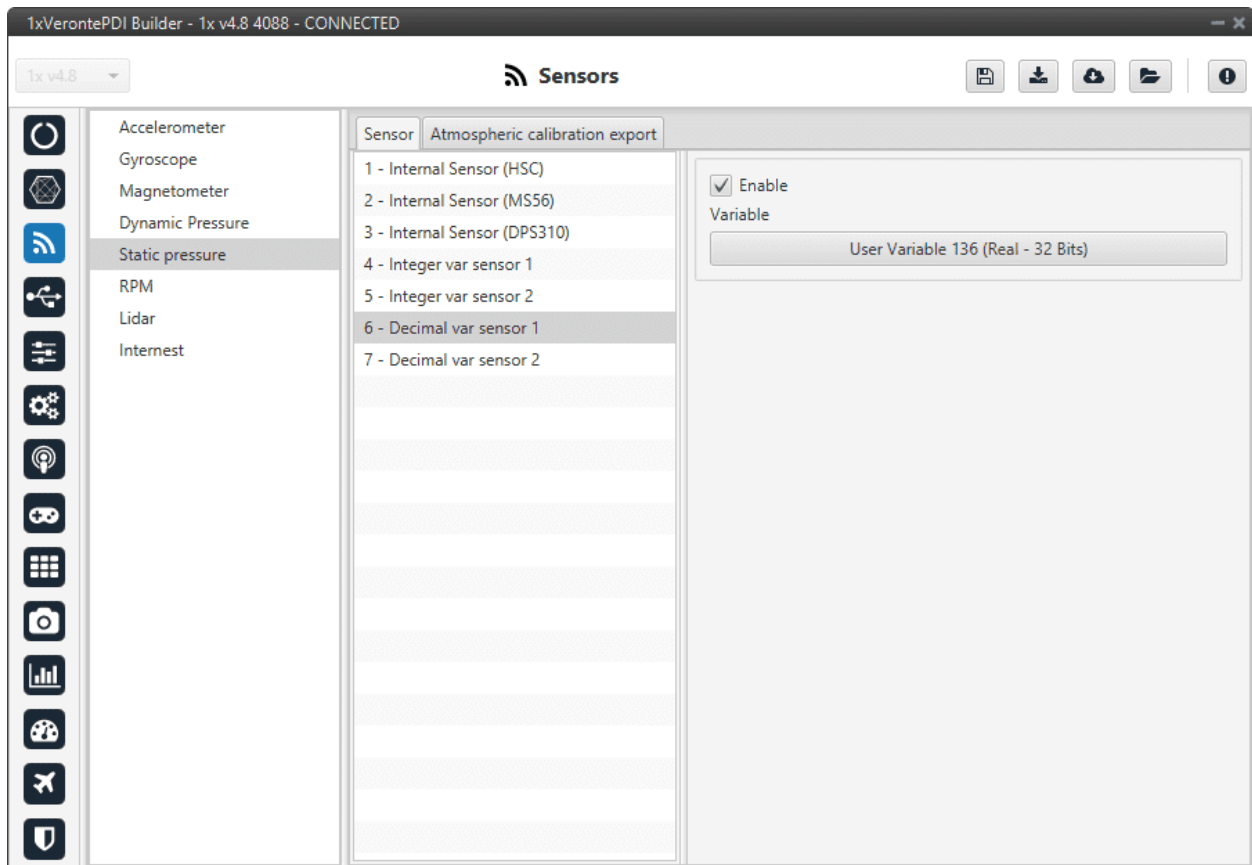


Fig. 60: Decimal var static pressure sensor menu

2.3.6 RPM

1x autopilot can measure RPMs by measuring from up to 6 input sources:

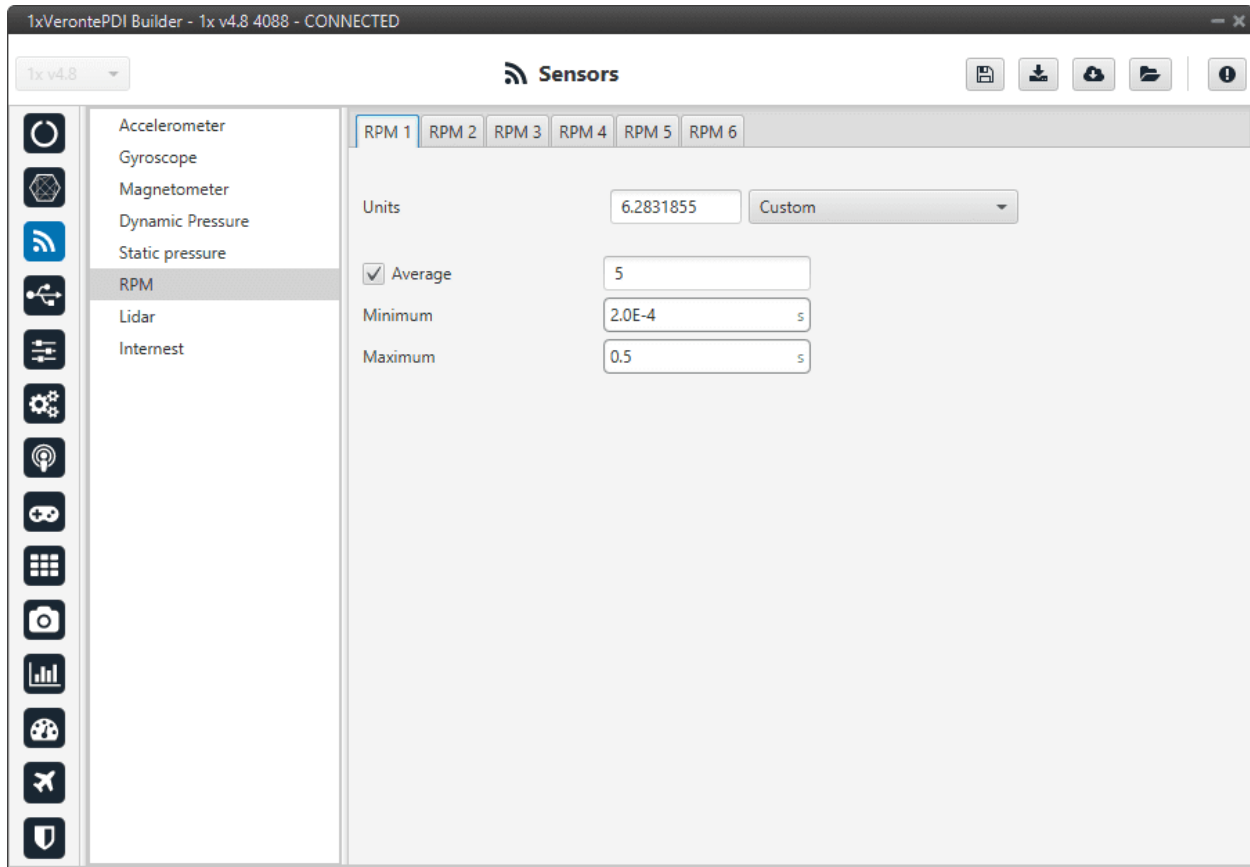


Fig. 61: **RPM menu**

- **Units:** Sensor conversion factor. It can be Custom, Radians per pulse or Pulse per cycle.
- **Average filter (Measures):** It is a filter to avoid voltage spikes.
- **Minimum pulse:** Minimum time to detect a lap.
- **Maximum time without capture:** The maximum period of time allowed without capturing.

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, 1x autopilot supports the following Lidar devices:

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

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

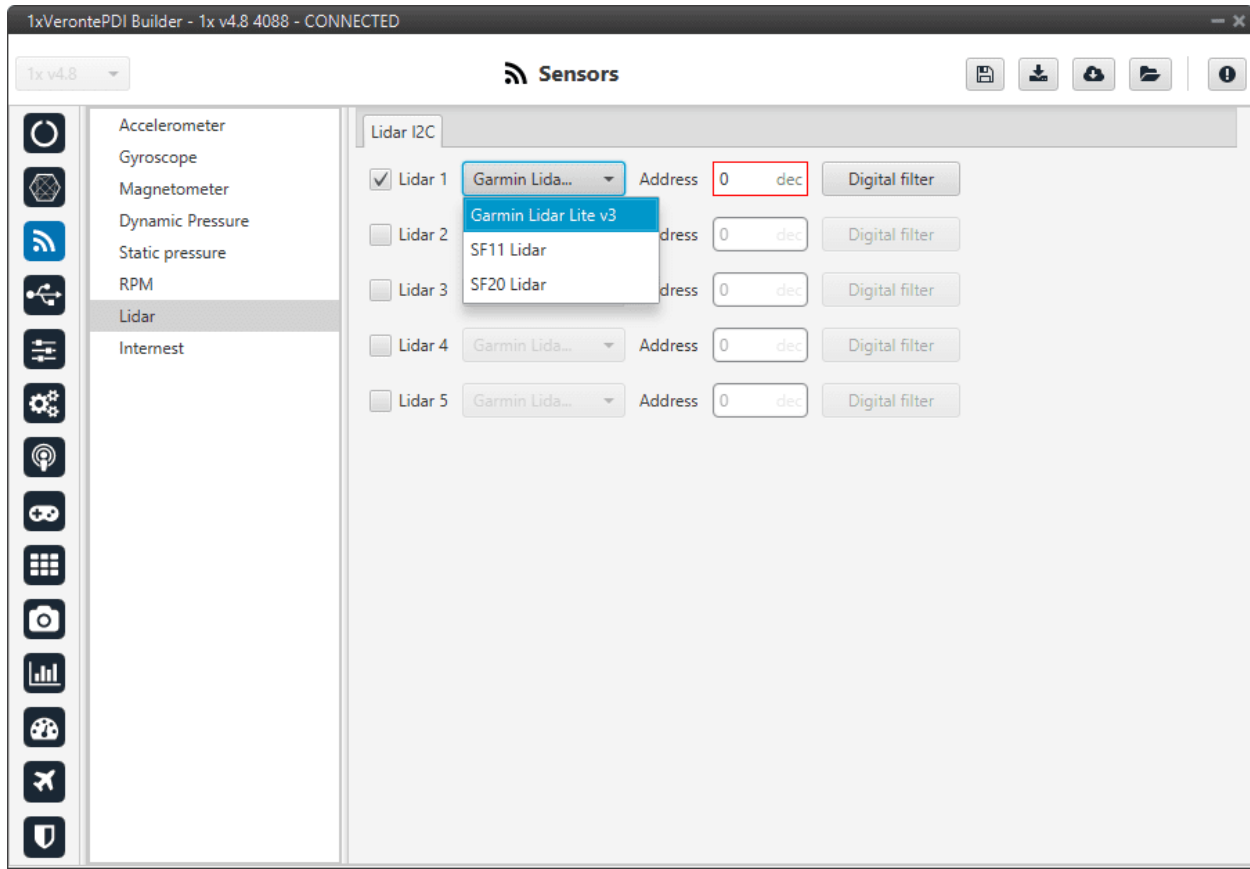


Fig. 62: Lidar devices

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:** 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**.

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 1/5) needs to be kept in order to properly configure the Altimeter later (*Altimers sensors - Block Programs* section).

2.3.8 Internet

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 **Internet system** with Veronte Autopilot 1x.

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

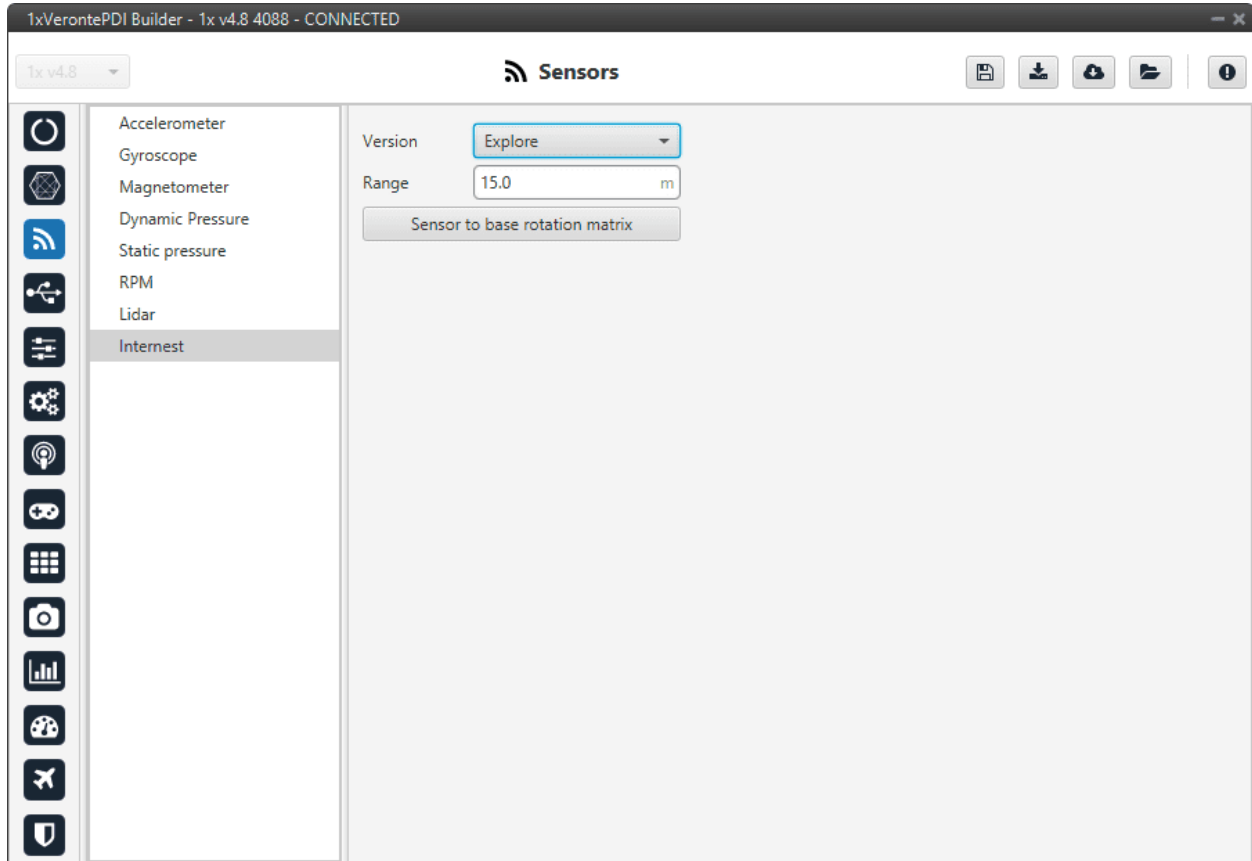


Fig. 63: Internet menu

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

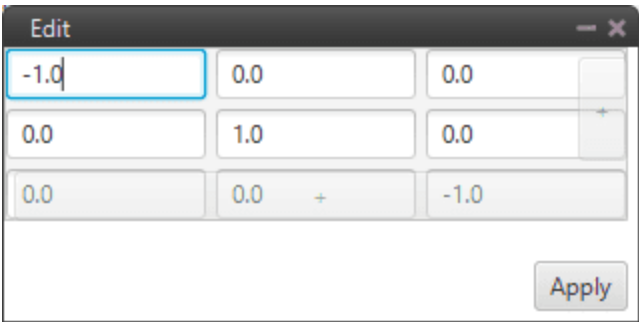


Fig. 64: Internest - Rotation matrix

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>I/O Setup</i>	Configuration of serial ports, LOS, Serial Custom Messages, etc.
<i>CAN Setup</i>	Configuration of the two CAN buses (A and B), CAN Custom Messages and Mailboxes
<i>Digital Input</i>	Configuration of PPM signals, pulses or RPM sensors

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 sect

2.4.1 I/O Setup

In this panel the user can stablish 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.

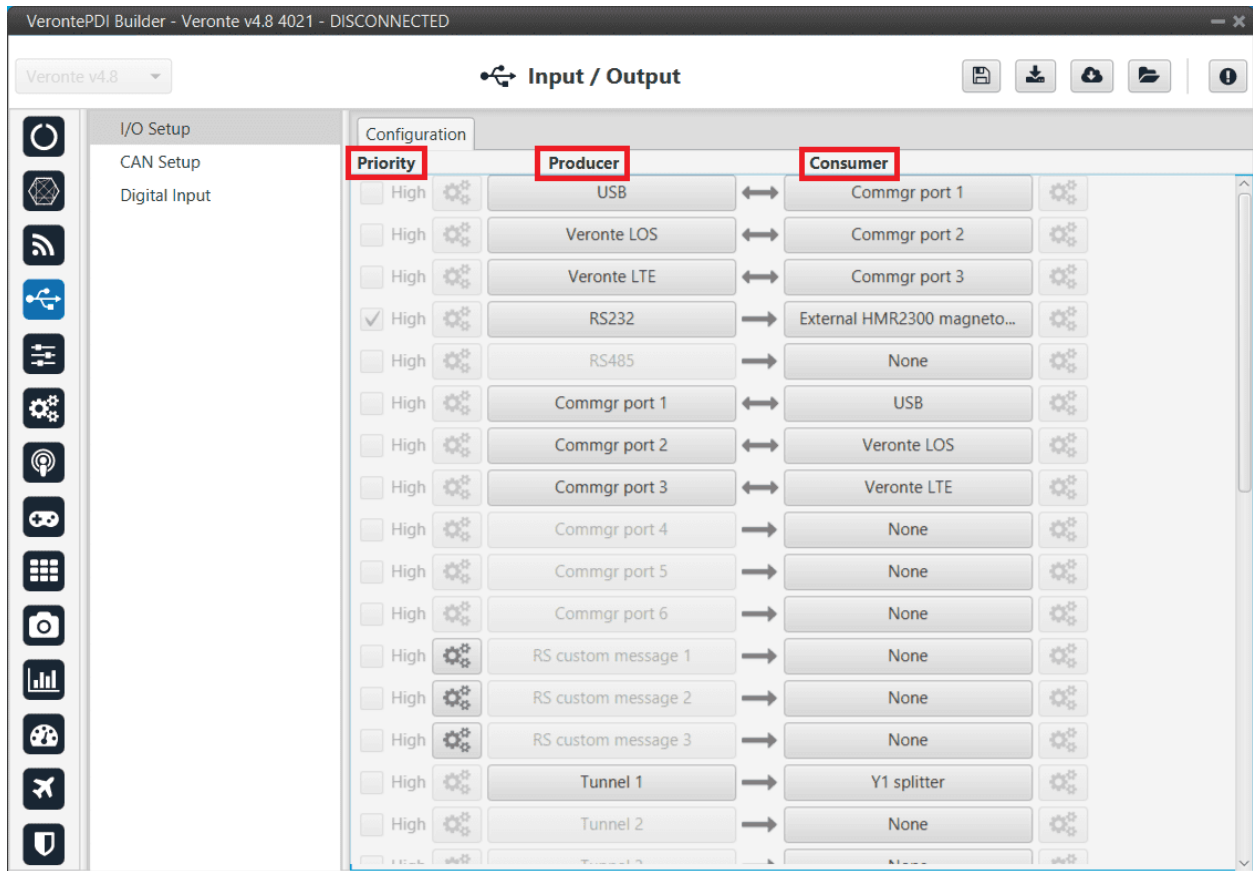


Fig. 65: I/O Setup menu

- **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**.
- **Producer:** Functions for creating and sending messages.
- **Consumer:** Functions for receiving and parsing messages

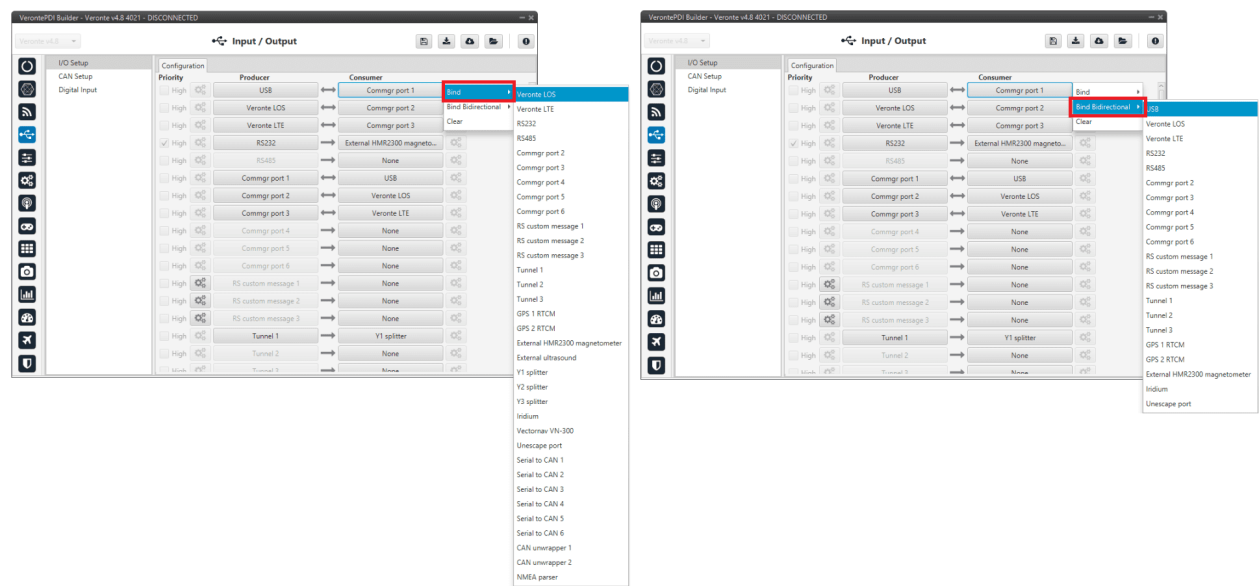


Fig. 66: I/O Setup consumer options

Firstly, users have to configure the **Producer** selecting the I/O port or information to use. Later, users have to configure the **Consumer** by clicking on an element, a new window will be displayed to select an item. The relationship between them can be unidirectional (Bind) or bidirectional (Bind Bidirectional), the last enables a port to receive or send information.

The following I/O ports are available:

Field	Description
USB	USB Port
Veronte LOS	Radio
Veronte LTE	4G Connection
RS232	Serial Port 232
RS485	Serial Port 485
Commgr port	COM Manager ports send and receive VCP messages. This is the protocol used by Veronte products to communicate. For more information on VCP, read the VCP user manual
Tunnel	Creates a bidirectional bridge between two devices, see Tunnel
RS Custom Message	This allows user to send/receive a serial custom message, see Serial Custom Messages
GPS RTCM	This allows the user to send/receive RTK information from GND unit to AIR unit
External HMR2300 magnetometer	External magnetometer sensor, see External HMR2300 magnetometer
Iridium	Iridium communication, see Iridium section
Splitter	Used to split a signal into 2
NMEA Parser	NMEA 0183 messages parser, see NMEA Parser
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
External ultrasound	External ultrasound sensor, see Interrest section
Vectornav VN-300	Vectornav VN-300 is an external IMU. For more information, see the Vectornav VN-300 -> Integration examples section of this manual

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

2.4.1.1 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 air 1x autopilot 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 Ground 1x autopilot, and then with the tunnel send the signal to the air one. With this configuration it would be like if the button were physically connected to the aircraft.

Let's consider the following image:

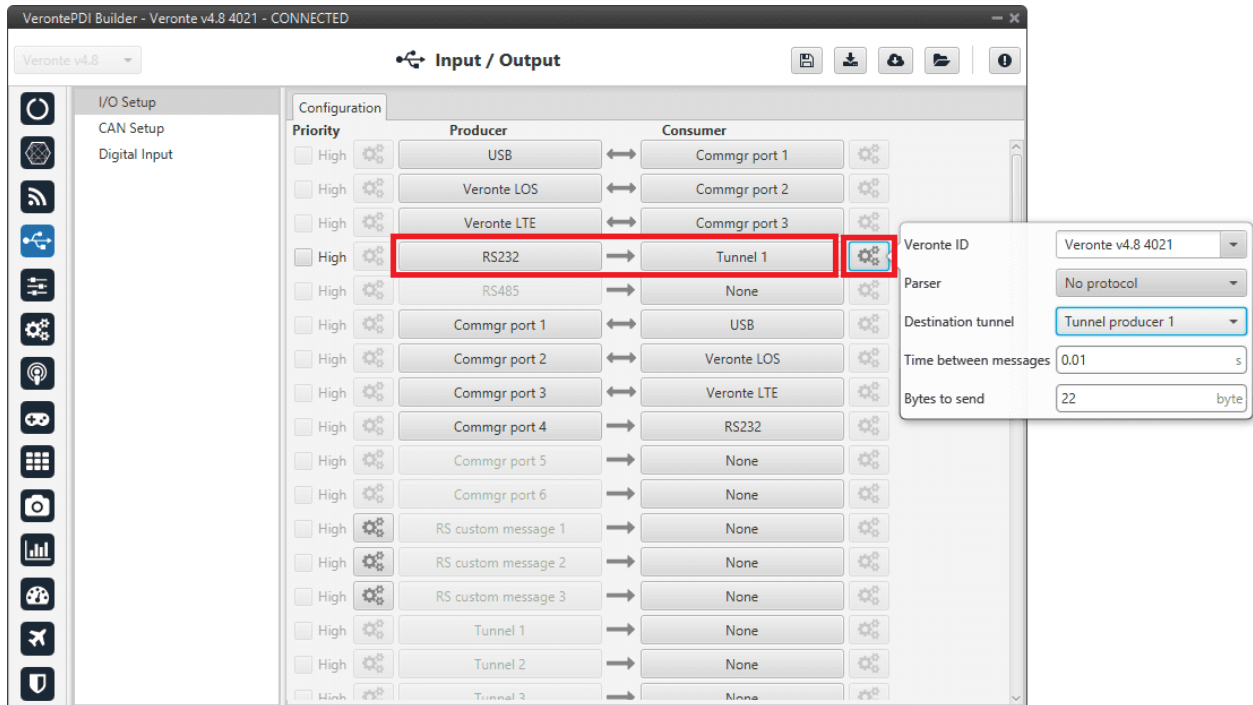


Fig. 67: Tunnel configuration

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 1x autopilot 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:** Select the address that will receive the information.
 - **App 2:** Veronte Ops address.
 - **Broadcast:** All units on the network. Select this option for a generic configuration.
 - **Veronte v4.X XXXX:** Address of a specific Veronte unit, it can be a 1x, a 4x, a CEX, etc.
- **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 1, 2 and 3* 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.

2.4.1.2 Serial Custom Messages

Warning:

- 1x autopilot 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.

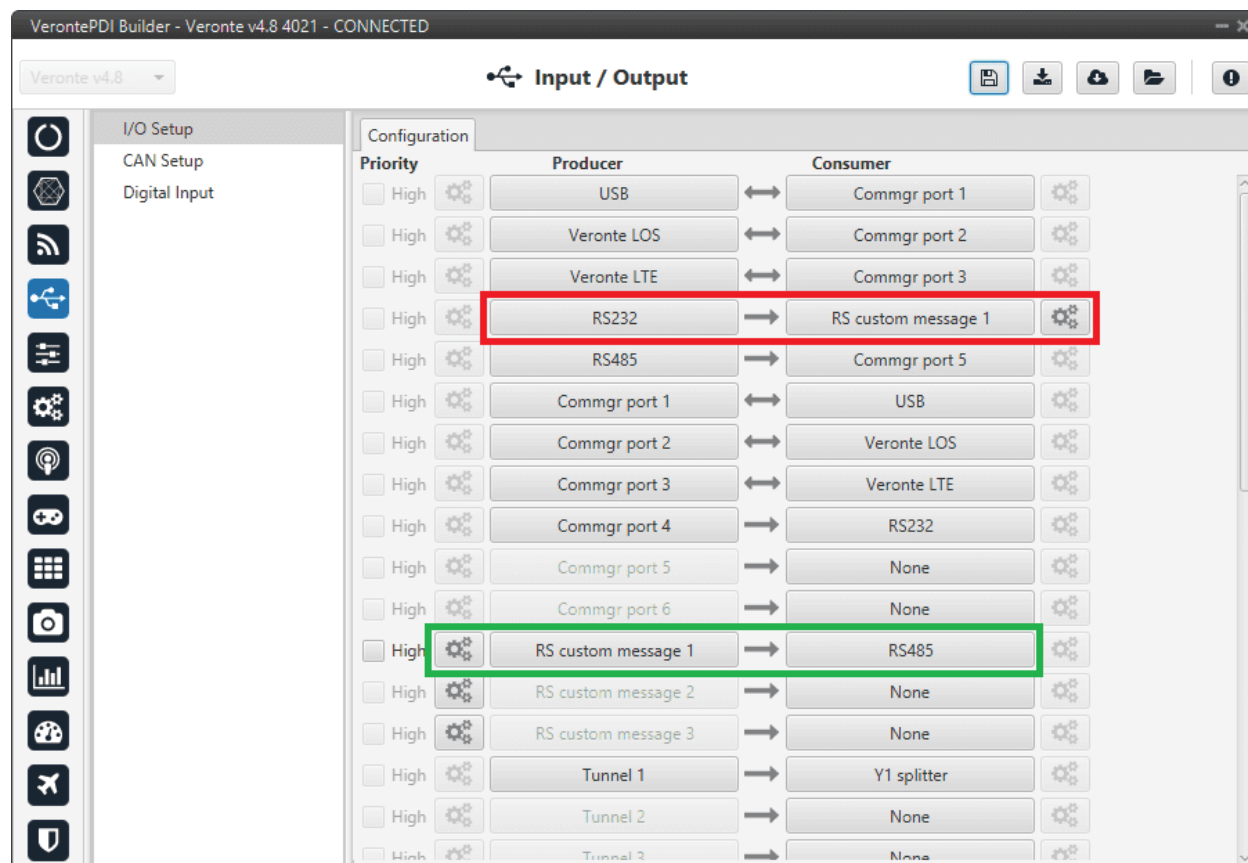




Fig. 68: Serial Custom Messages

In the image above can be seen two possible configurations using a RS Custom Message. The 'red' one is configured to receive a determined message from a RS-232 serial port and the 'green' is used to send a RS Custom Message through a RS-485 serial port. It is also possible to use the same RS Custom Message for both tasks if Bind Bidirectional is used (the arrow indicates this).

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

1. Press the **configuration button** ( icon) and another window will be displayed. In this window press the  icon to add a custom message, the user can choose between **System variables** or **ADSB Vehicle**.

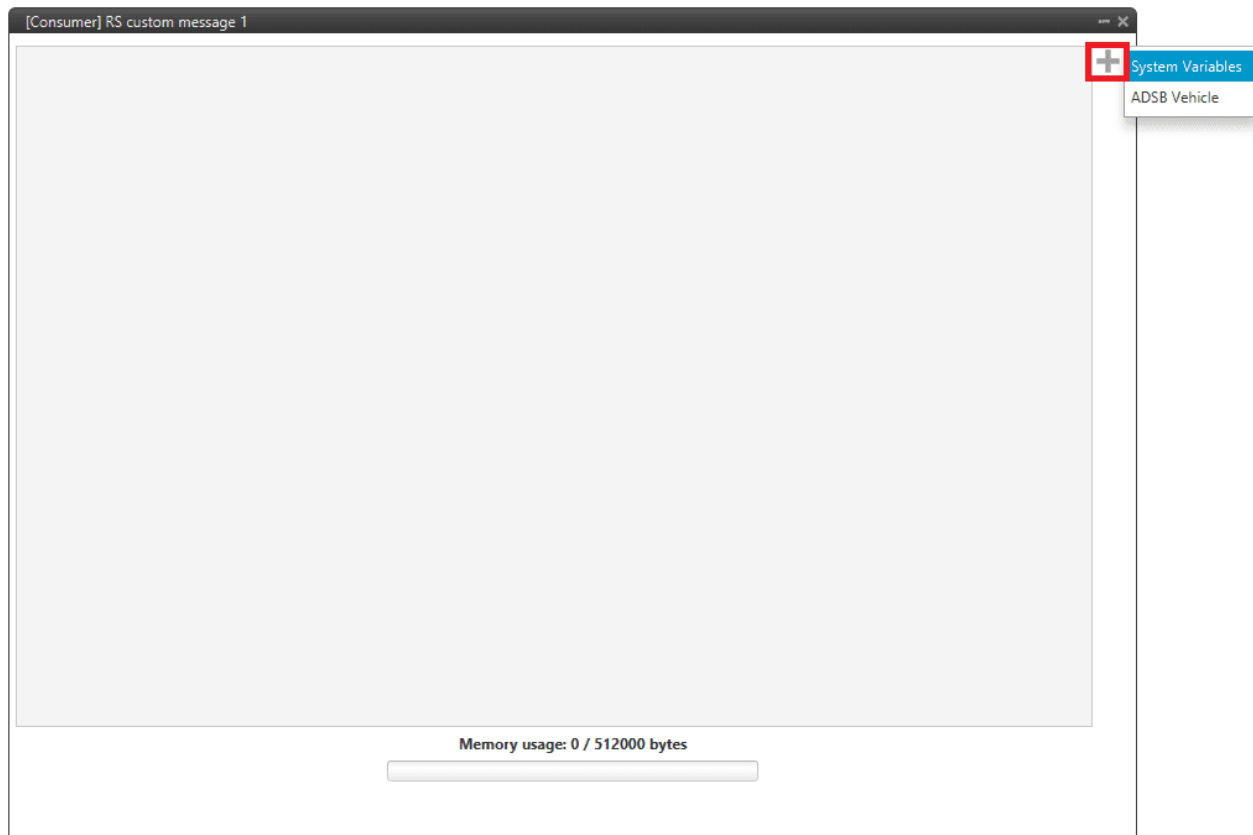
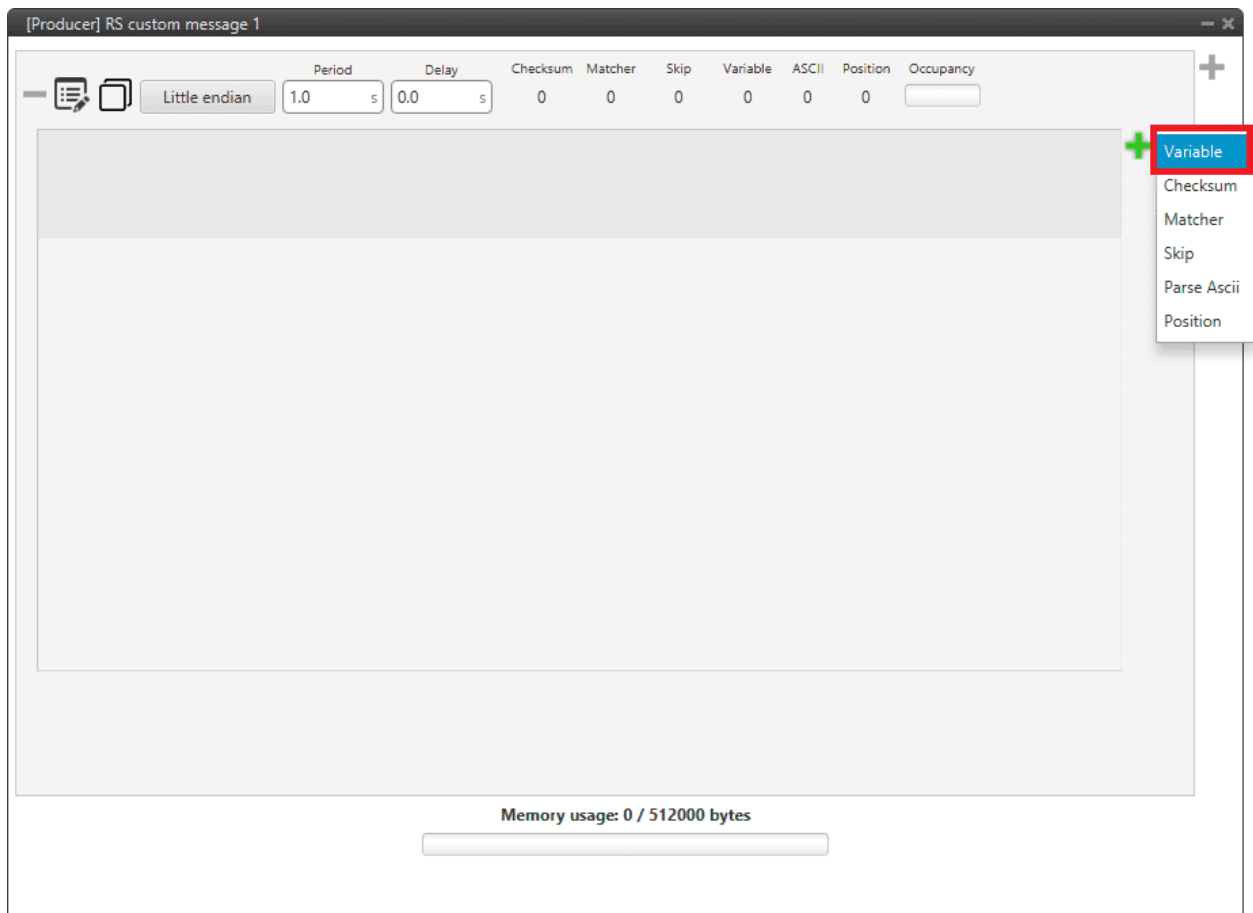


Fig. 69: Serial Custom Message configuration

Note: The difference between **System variables** and **ADSB Vehicle** is that when the user has selected the second option, only the ADSB variables will appear and not all the ones available in 1x PDI Builder.



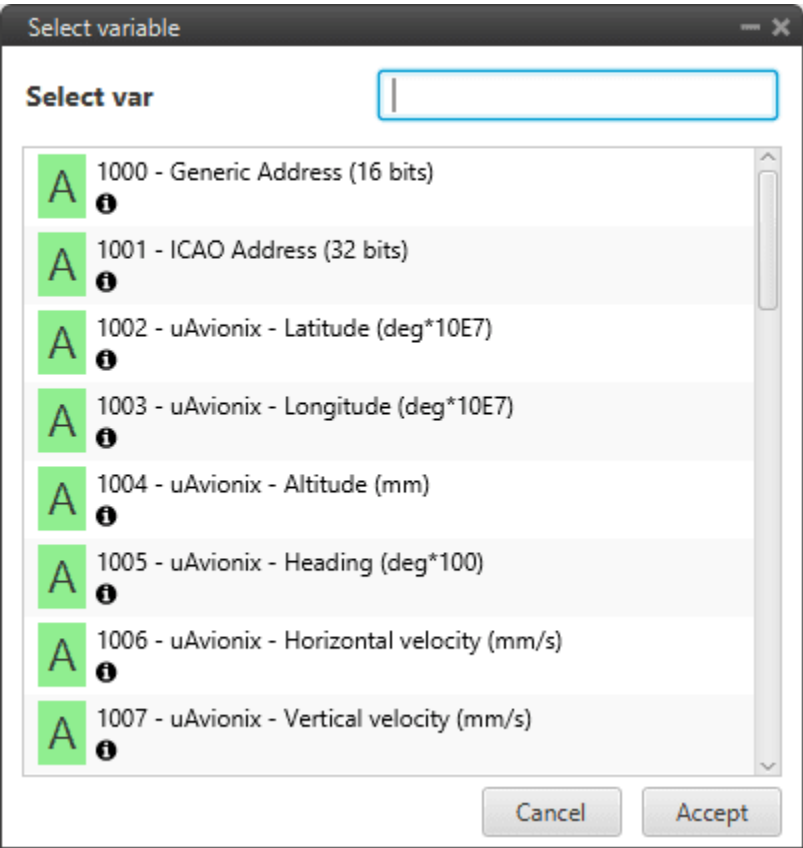


Fig. 70: ADSB Vehicle Variable custom message

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

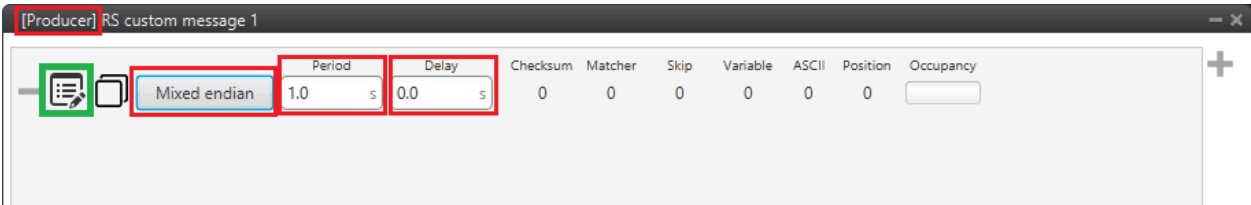


Fig. 71: Producer RS Custom Message configuration





Fig. 72: 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:** This option has a dual role depending on if it is used to transmit or receive data.
 - **Delay - Producer:** It is a delay applied before sending the message. This serves to send messages with the same period without overloading the Serial bus.
 - **Time to Idle - Consumer:** This is the time 1x autopilot 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 [Custom Messages types section](#).

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.

2.4.1.3 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:

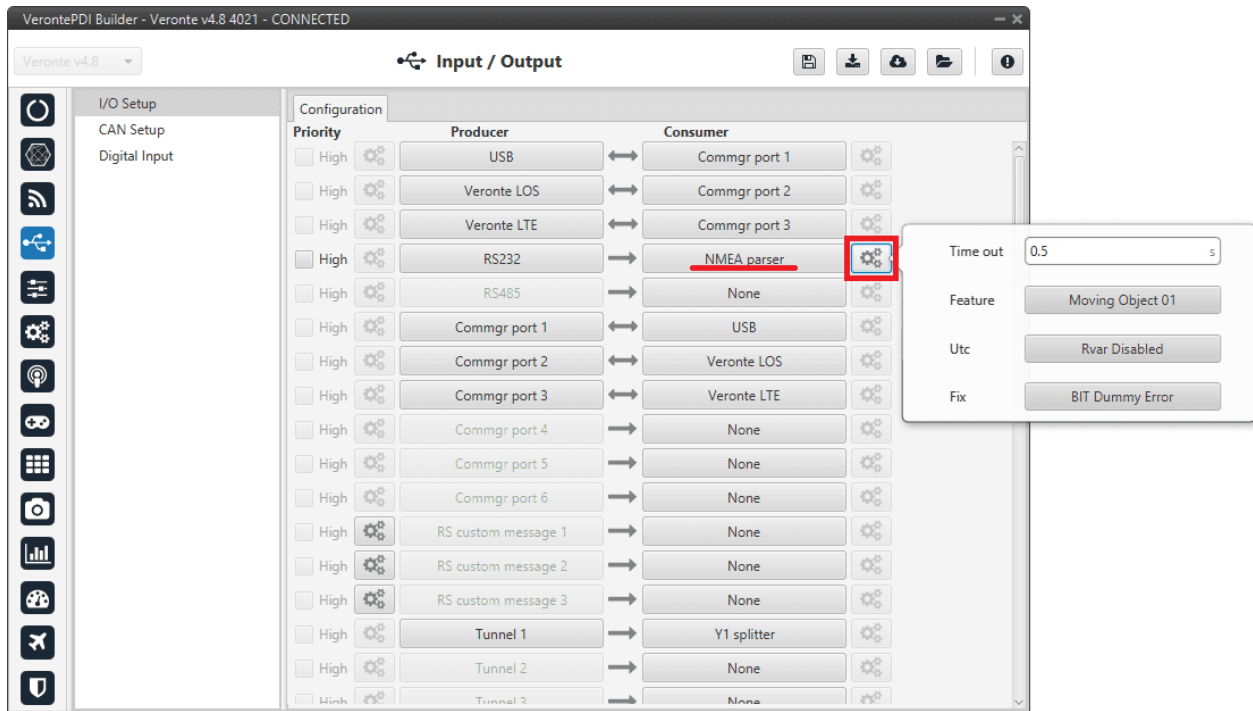


Fig. 73: NMEA Parser configuration

- **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 in 1x PDI Builder.
- **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](#) section.

2.4.1.4 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 misinterpreted 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.

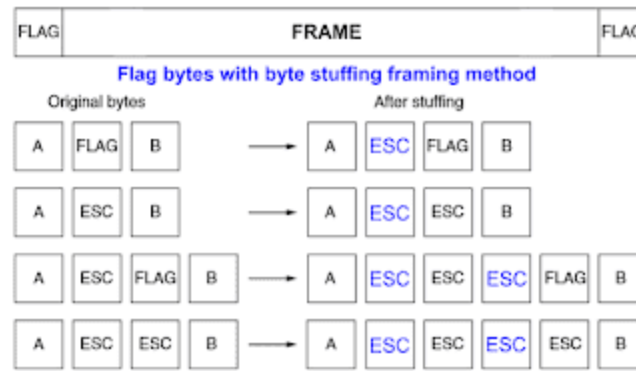


Fig. 74: Escape byte

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

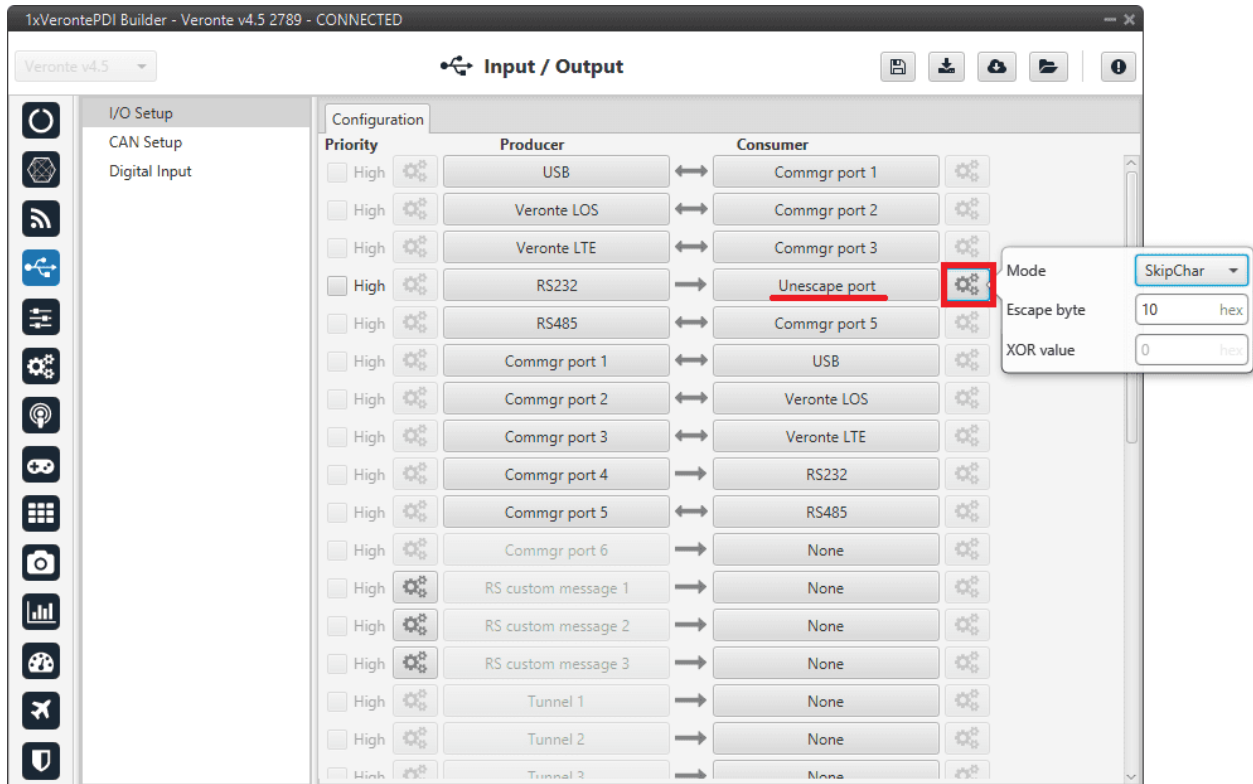


Fig. 75: Unescape port configuration

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.

2.4.2 CAN Setup

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

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

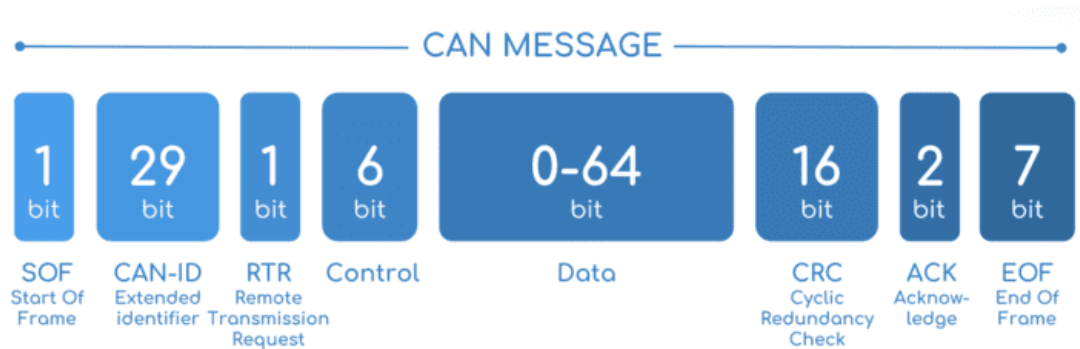


Fig. 76: CAN message structure

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 section*.

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

2.4.2.1 Configuration

This menu allows the configuration of communications between different devices.

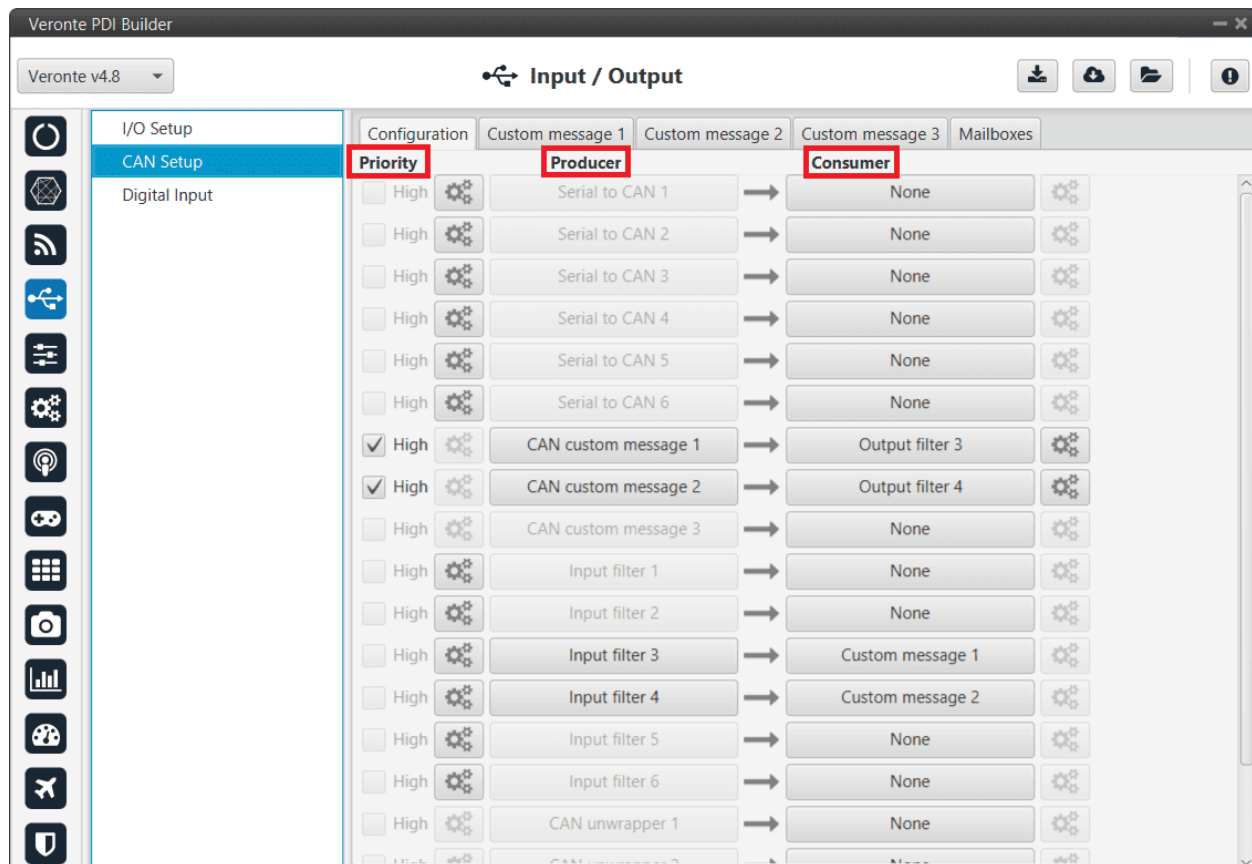



Fig. 77: CAN configuration section

In this menu, the user can find the same 'columns' (**Priority**, **Producer** and **Consumer**) as in the *I/O Setup menu*.

Warning: In CAN, in Low state the specified period is not guaranteed but in 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, 1x autopilot 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:

Warning: For correct communication, mark it as **High priority** (with the Priority checkbox).

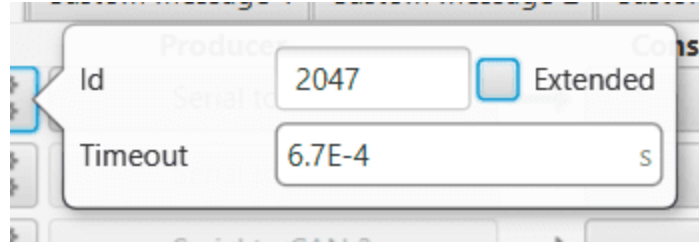



Fig. 78: Serial to CAN configuration

- **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 next section, see *Custom Messages*.
- **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:

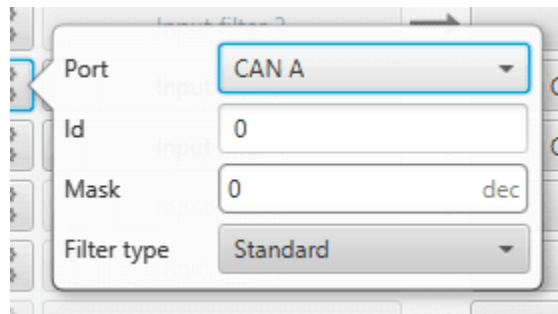



Fig. 79: Input filter configuration

- **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 defines the bits that should match.
- **Filter type**: The options available are *Standard*, *Extended* and *Both*.

Attention: Make sure that the mask is set properly to be able to receive the desired CAN messages.

- **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 and Arbiter. It can be configured in the **configuration button** ( icon), a pop-up window will appear:

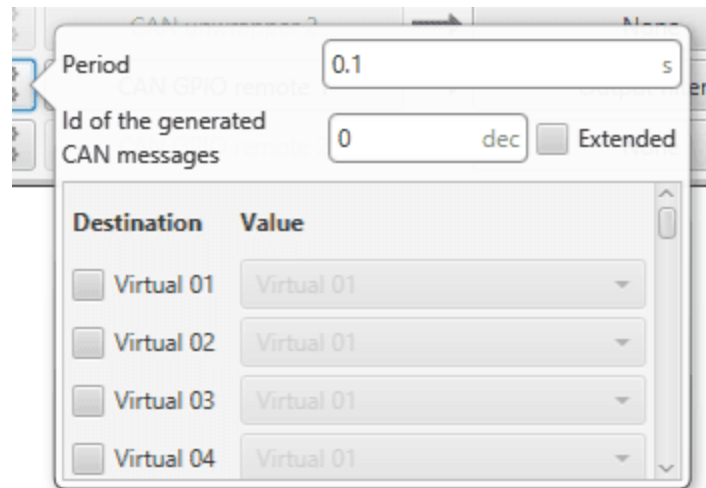



Fig. 80: CAN GPIO remote configuration

- **Period:** It is the period of sending messages.
- **Id of the generated CAN message:** 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.
- **Destination:** Here the user select the destination CEX pins.
- **Value:** The user must select the 1x pin to be connected to the CEX pin.

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*.

Warning: For correct communication, mark it as **High priority** (with the Priority checkbox).

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

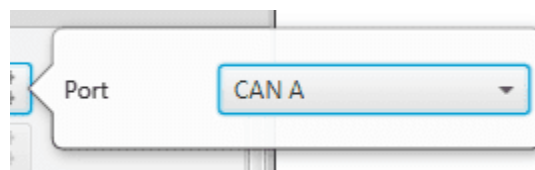


Fig. 81: Output filter configuration

- **CAN wrapper:** CAN messages over serial output, it has to be connected to *I/O Setup producer*.

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.
- 1x autopilot 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**

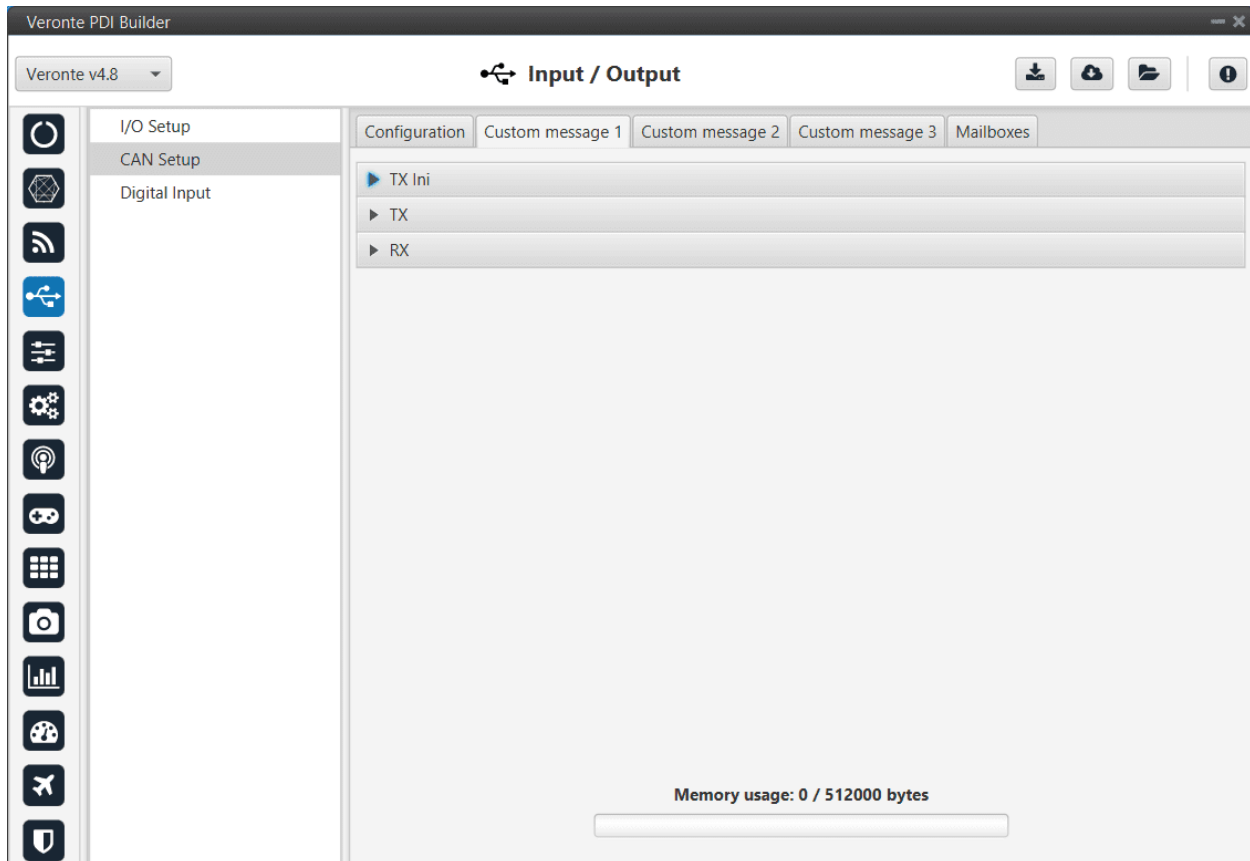


Fig. 82: CAN Custom Message section

TX Messages

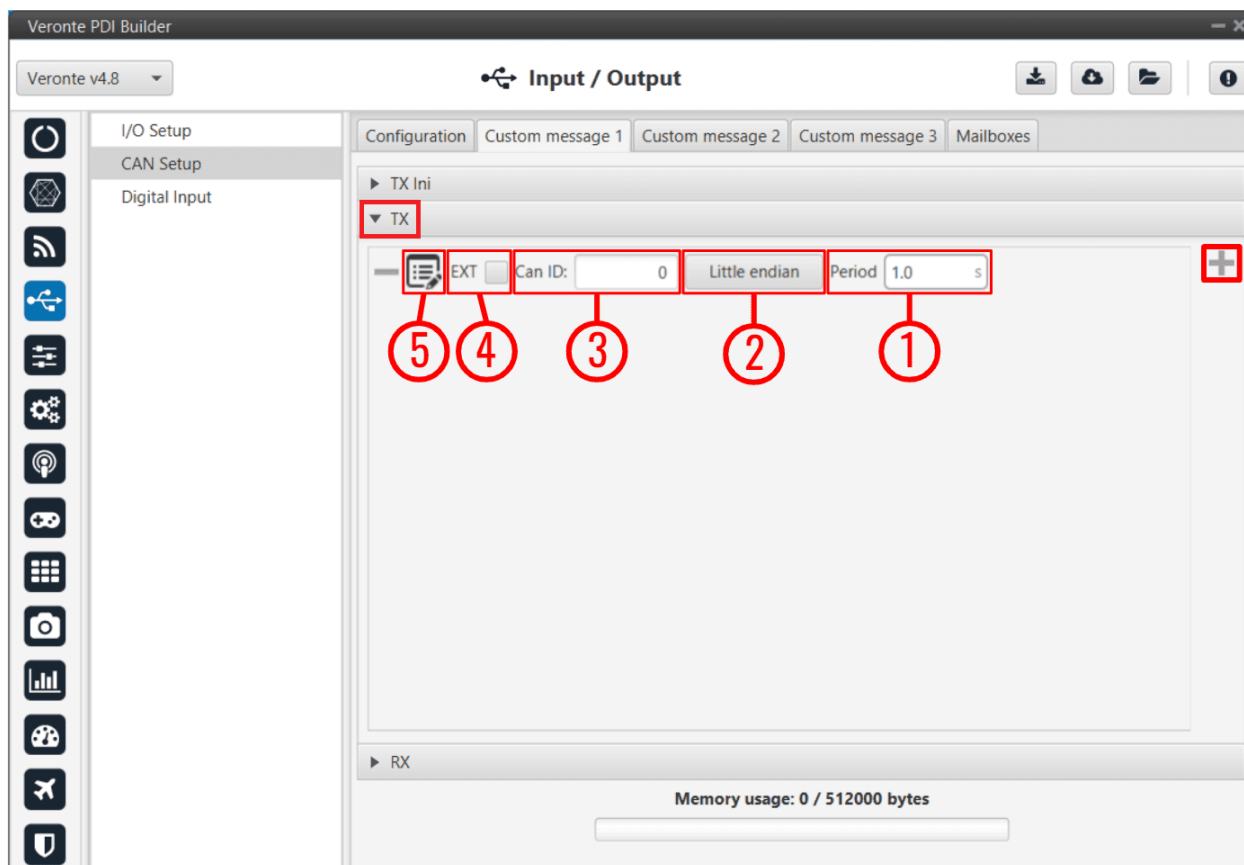



Fig. 83: CAN Custom Message - TX

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:** Enables the frame format with a 29-bit identifier (Extended).
5. **Edit message button** ( icon): 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.

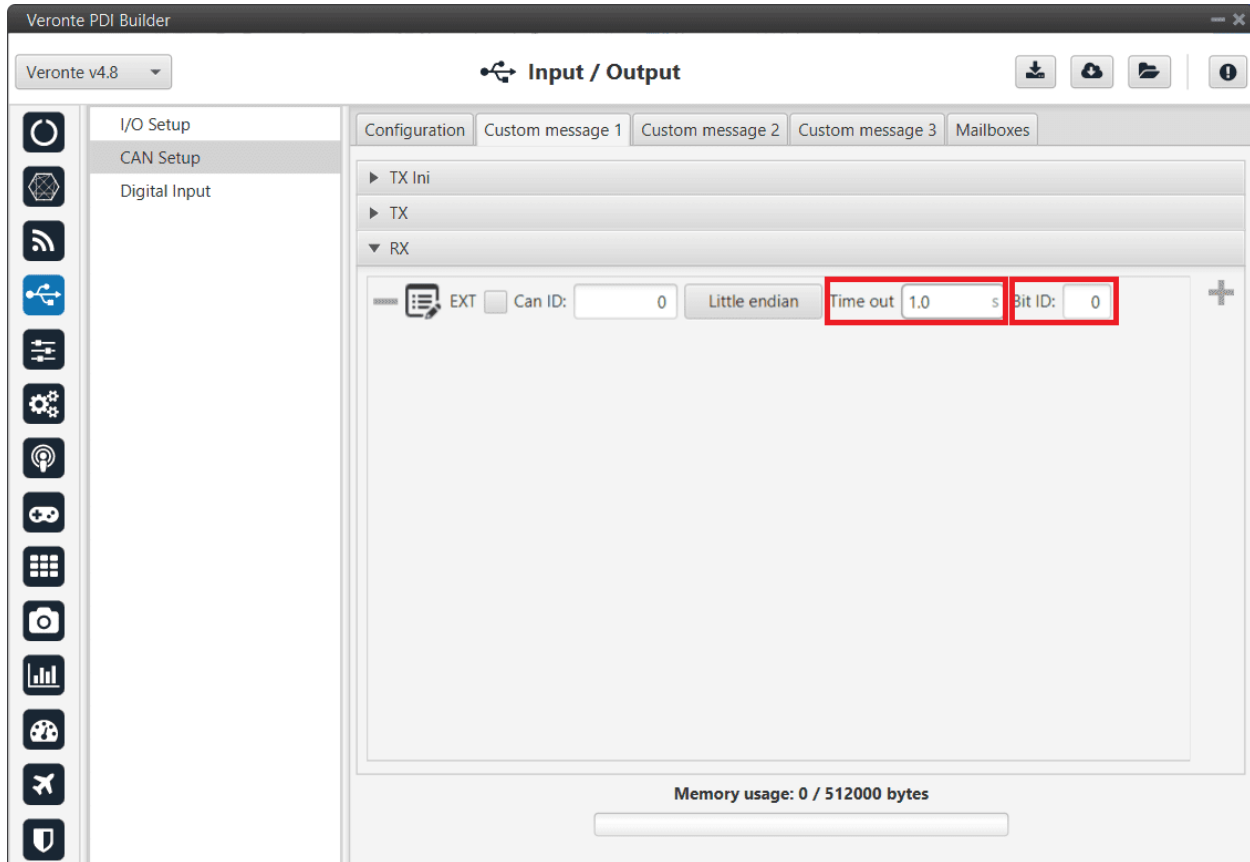


Fig. 84: CAN Custom Message - RX

The options and parameters to configure here are almost the same as those described in *TX 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* for more information.

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

- **Time out:** 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) need to be used to store that data.

2.4.2.3 Mailboxes

Here the user can modify the mailboxes from the selected CAN bus (CAN A or CAN B).

When 1x autopilot is going to receive data on the CAN Bus, it is mandatory to configure a certain number of mailboxes. However, it is also necessary to **have at least 1 mailbox for transmission (TX)**.

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

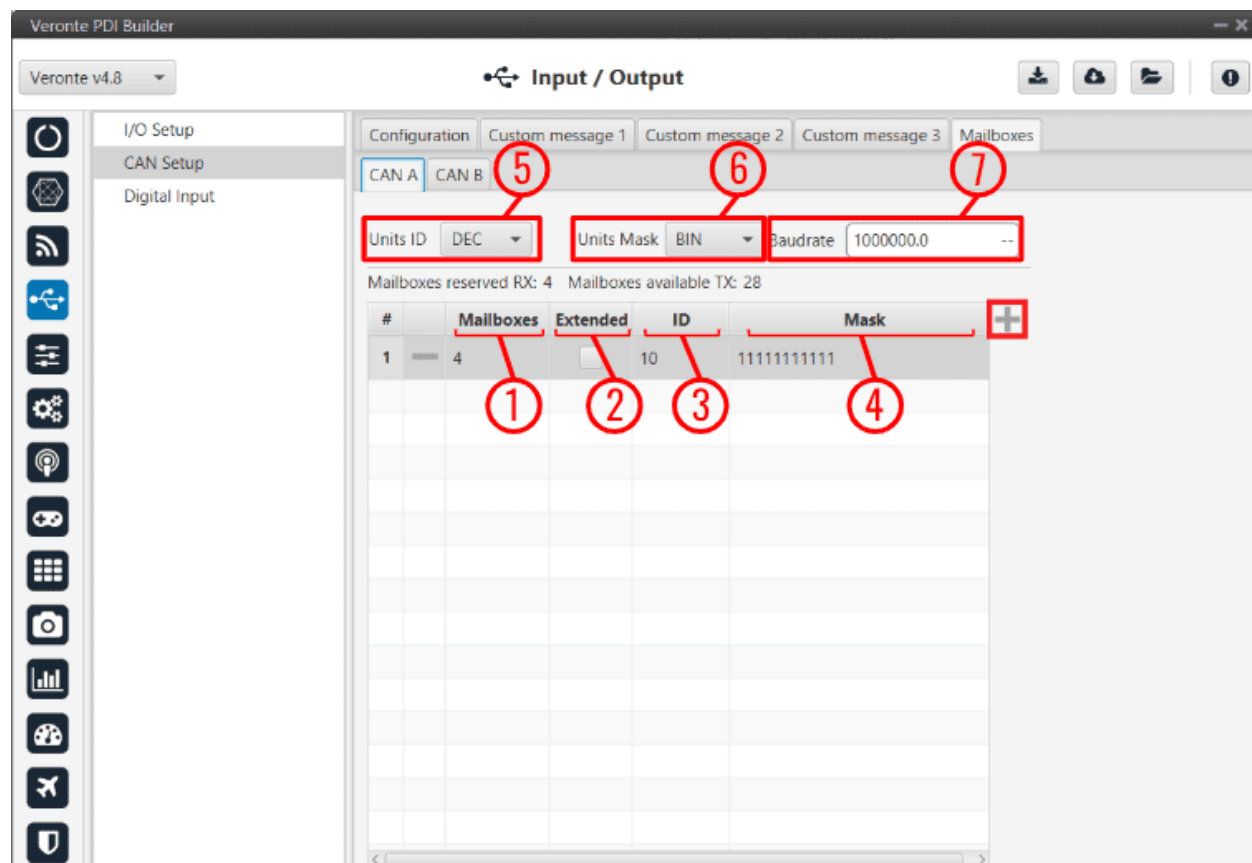


Fig. 85: Mailboxes section

Mailboxes are required to store the data received until 1x autopilot reads it and it is **necessary at least one mailbox per RX message**. 1x PDI Builder allows up to **32 mailboxes**.

Warning: Do not use all 32 mailboxes for RX, leave at least one free for TX.

The configurable parameters when adding a new mailbox are:

1. **Mailboxes:** Number of mailboxes assigned to that ID.
2. **Extended:** Select this option for 29-bit IDs.
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.
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.

Warning: If any mailbox is **full** and another message arrives, the **new** message is **discarded**.

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 bits in a system variable (RX)** or to **send a certain variable (TX)**.

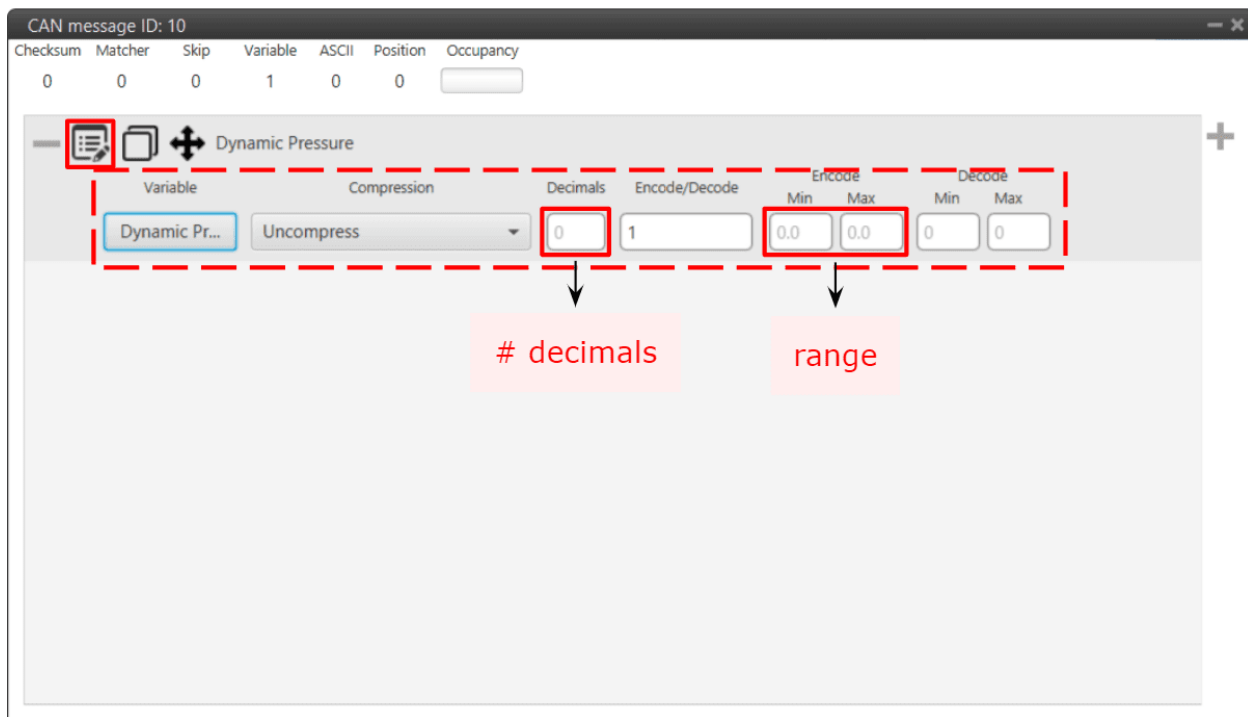


Fig. 86: Variable configuration - CAN Custom Message

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:** Uncompress from 32 to 64 bits (TX). In RX, uncompress from 64 to 32 bits.

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

- **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.
- **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.

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** section 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**).

The image shows a configuration window for a variable named 'REF_HDG_VALUE'. The 'Variable' field contains 'REF_HDG_V...'. The 'Compression' dropdown is set to 'Compress - Bits Unsigned'. The 'Bits' field is set to '16'. The 'Encode/Decode' dropdown is set to '1'. The 'Encode' section has 'Min' set to '0.0' and 'Max' set to '359.0'. The 'Decode' section has 'Min' set to '0' and 'Max' set to '35900'. Red boxes highlight the 'Bits', 'Encode/Decode', 'Encode Min', 'Encode Max', 'Decode Min', and 'Decode Max' fields.

Fig. 87: 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.

The image shows a configuration window for a CAN message ID: 0. The 'Checksum' field is set to '1'. The 'Matcher' field is set to '0'. The 'Skip' field is set to '0'. The 'Variable' field is set to '0'. The 'ASCII' field is set to '0'. The 'Position' field is set to '0'. The 'Occupancy' field is set to '0'. The 'CRC (crc16veronte)' section is expanded. The 'Type' dropdown is set to 'Polynomial'. The 'Bits' field is set to '16'. The 'Endianness' dropdown is set to 'Little e...'. The 'Crc - Preset' dropdown is set to 'crc16veronte'. The 'BackFrom' field is set to '2'. The 'BackTo' field is set to '0'. The 'Polynomial StartValue' field is set to '32773'. The 'Final XOR' field is set to '23073'. The 'R. In' checkbox is checked. The 'R. Out' checkbox is checked. The 'Binar...' dropdown is set to 'Binar...'. Red boxes highlight the 'Type', 'Bits', 'Endianness', 'BackFrom', 'BackTo', 'Polynomial StartValue', 'Final XOR', 'R. In', 'R. Out', and 'Binar...' fields.

Fig. 88: 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 sequence, 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 Sagatech, that is used by Sagatech 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.

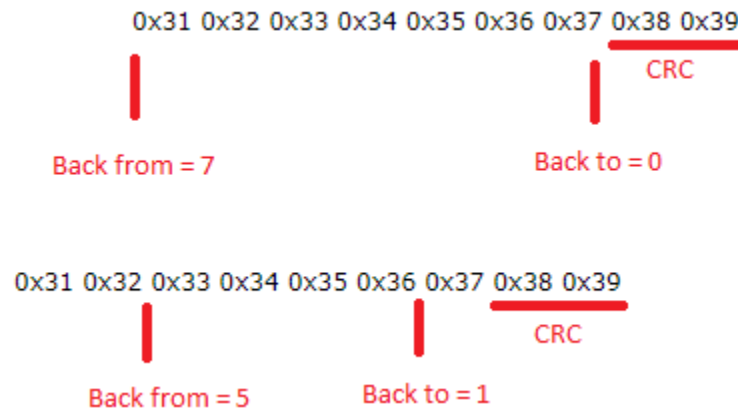


Fig. 89: **Back to/Back from** explanation

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

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

Polynomial type

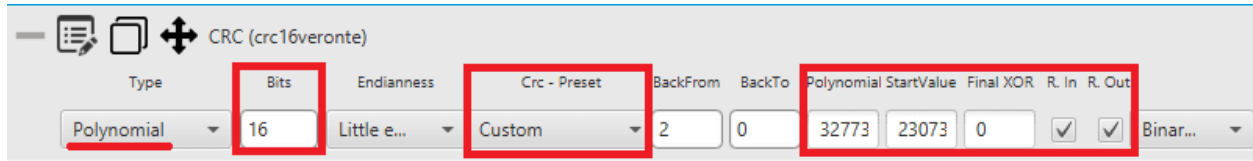


Fig. 90: Checksum configuration - Polynomial example

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 n° 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 (n 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.

sum8 type

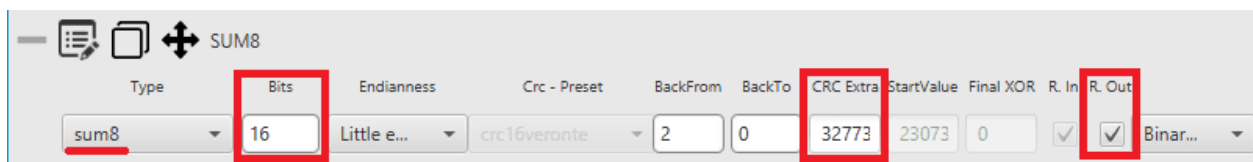


Fig. 91: Checksum configuration - sum8 example

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 (n 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.

sumMod type

Fig. 92: 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 (n bits).
- **Module:** This value is the dividend of the operation carried out.
- **Direct:**
 - If **enable**, sumMod retrieves the result code directly from the remainder of the division.
The mathematical operation done is: $CRC \% module$.
 - If **disable**, sumMod keeps the subtraction of the remainder of the division.
The mathematical operation done is: $module - (CRC \% module)$.

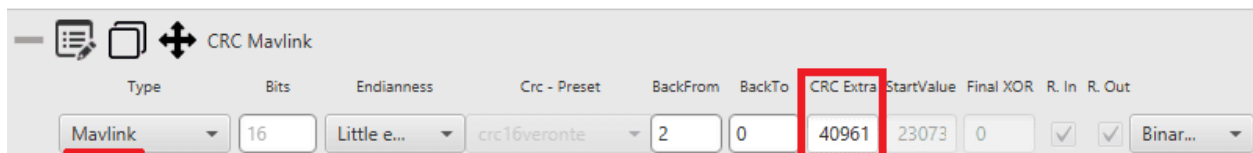
Mavlink type

Fig. 93: Checksum configuration - Mavlink example

In addition to the ‘general’ parameters described above, one further parameter 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.

8-bit sagetech checksum

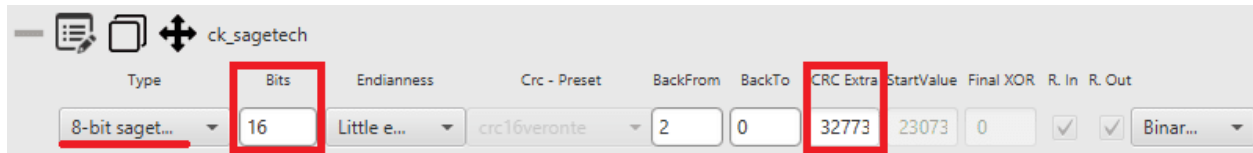


Fig. 94: Checksum configuration - 8-bit sagetech checksum example

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 (n 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.

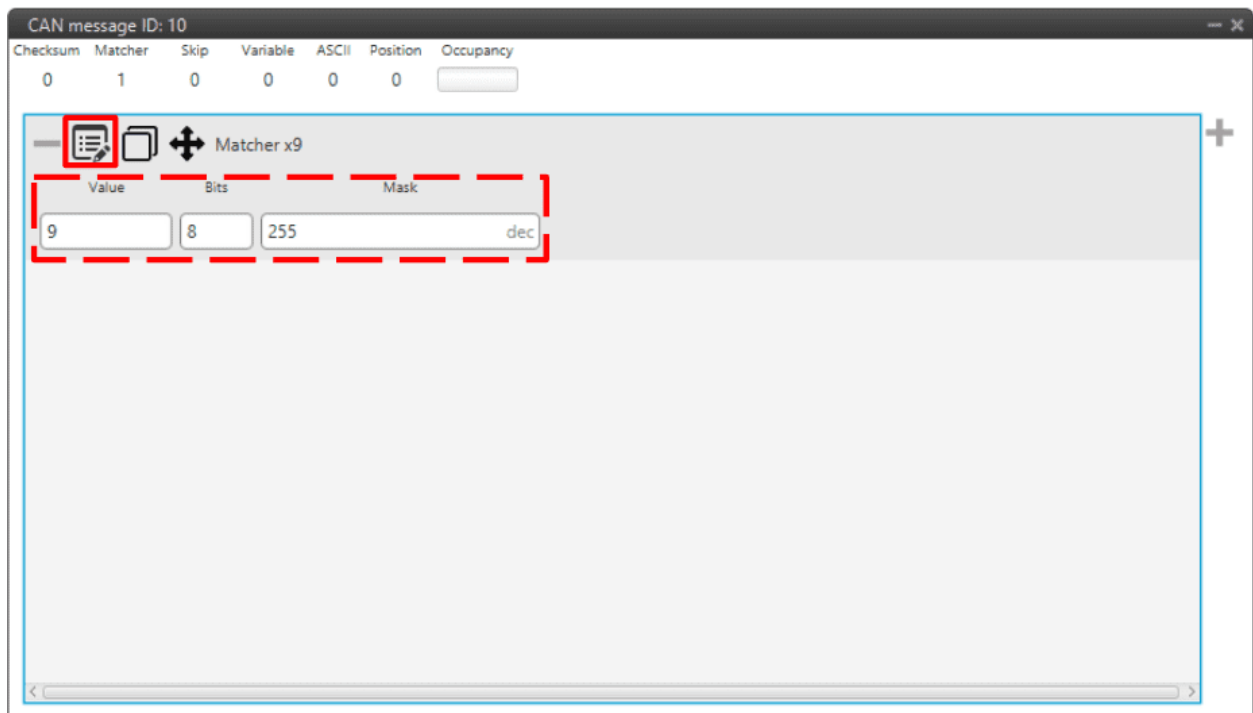


Fig. 95: Matcher configuration - CAN Custom Message

- **Value:** Sent/received value for the n° of bits defined below.
- **Bits:** Number of bits in which the matcher is performed.
- **Mask:** It is automatically set when the n° 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.

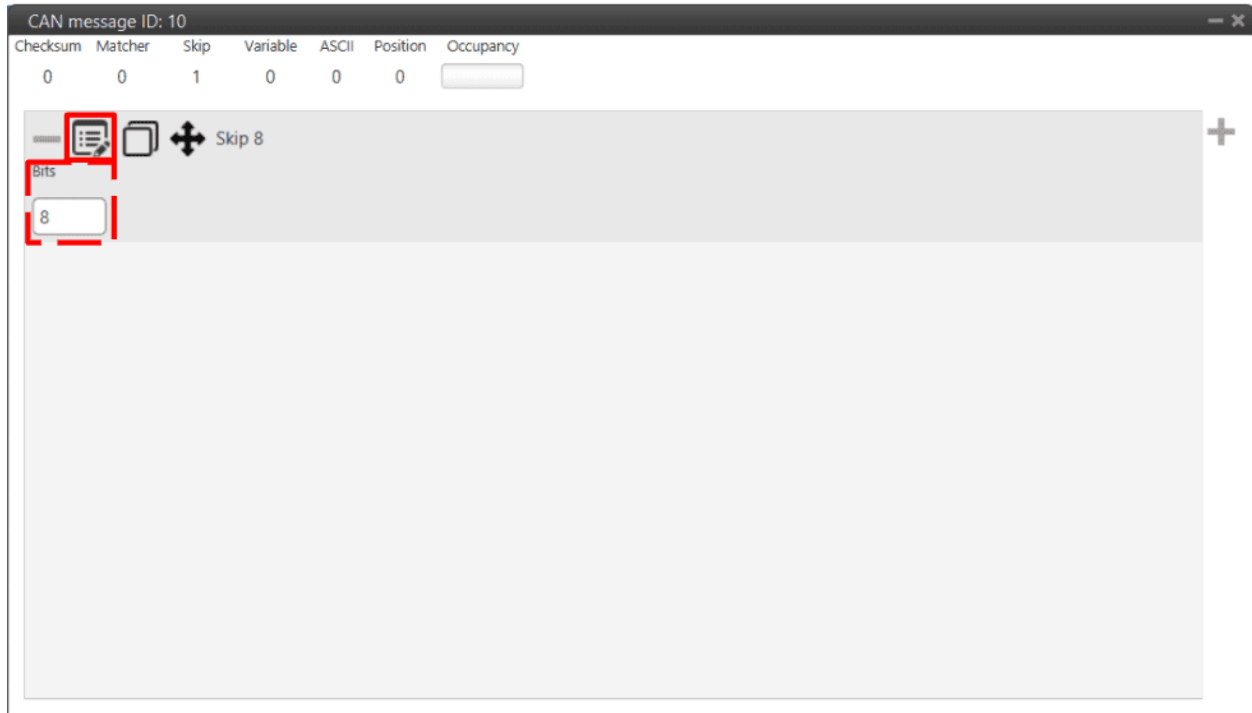


Fig. 96: Skip configuration - CAN Custom Message

2.4.3.5 Parse ASCII

Parsing ASCII is used when the protocol required is of this kind.

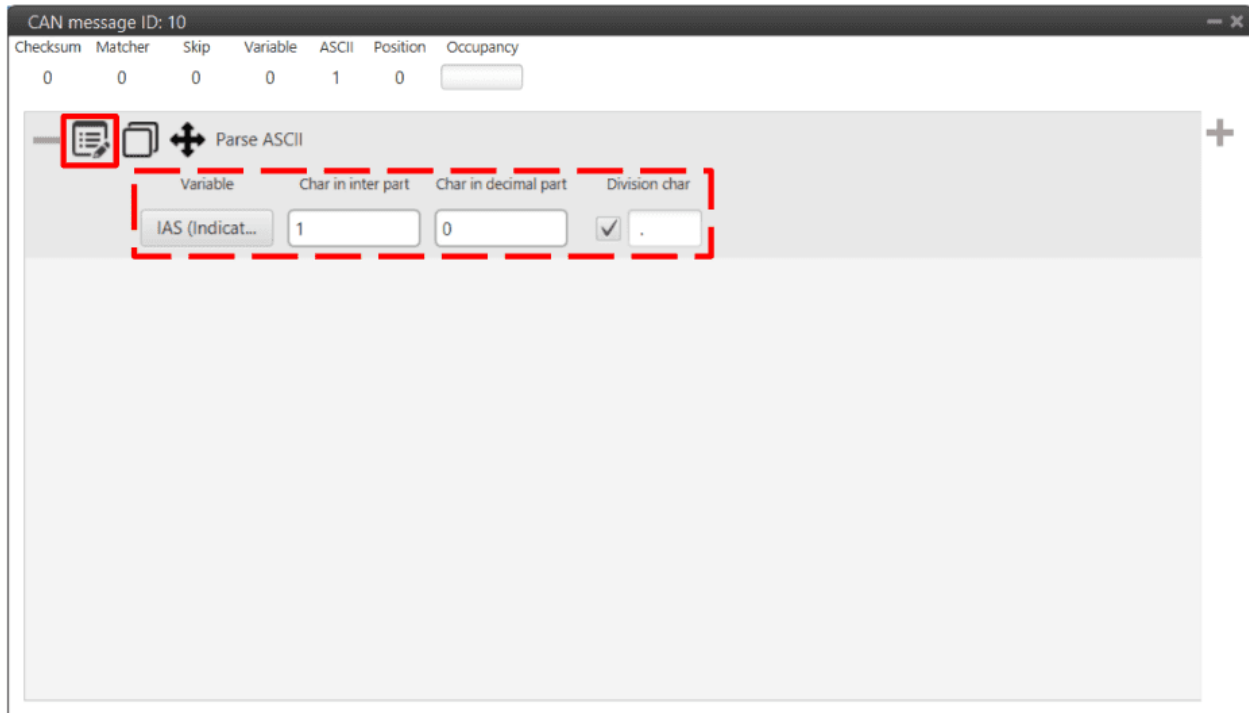


Fig. 97: Parse ASCII configuration - CAN Custom Message

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 tranformed 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 from Features variables.

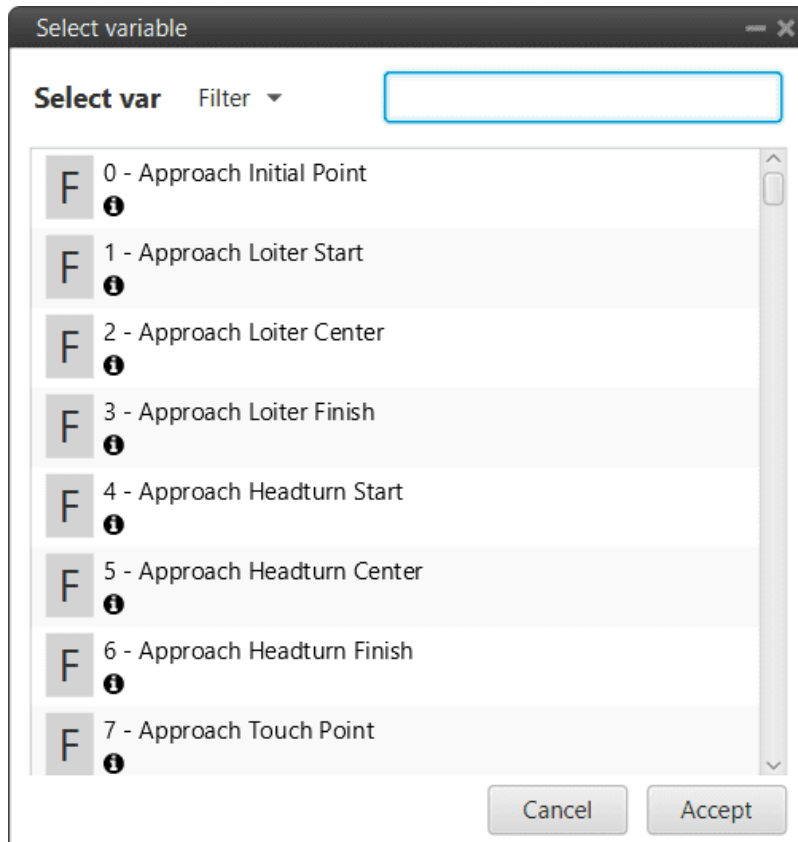


Fig. 98: Position variables

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**.

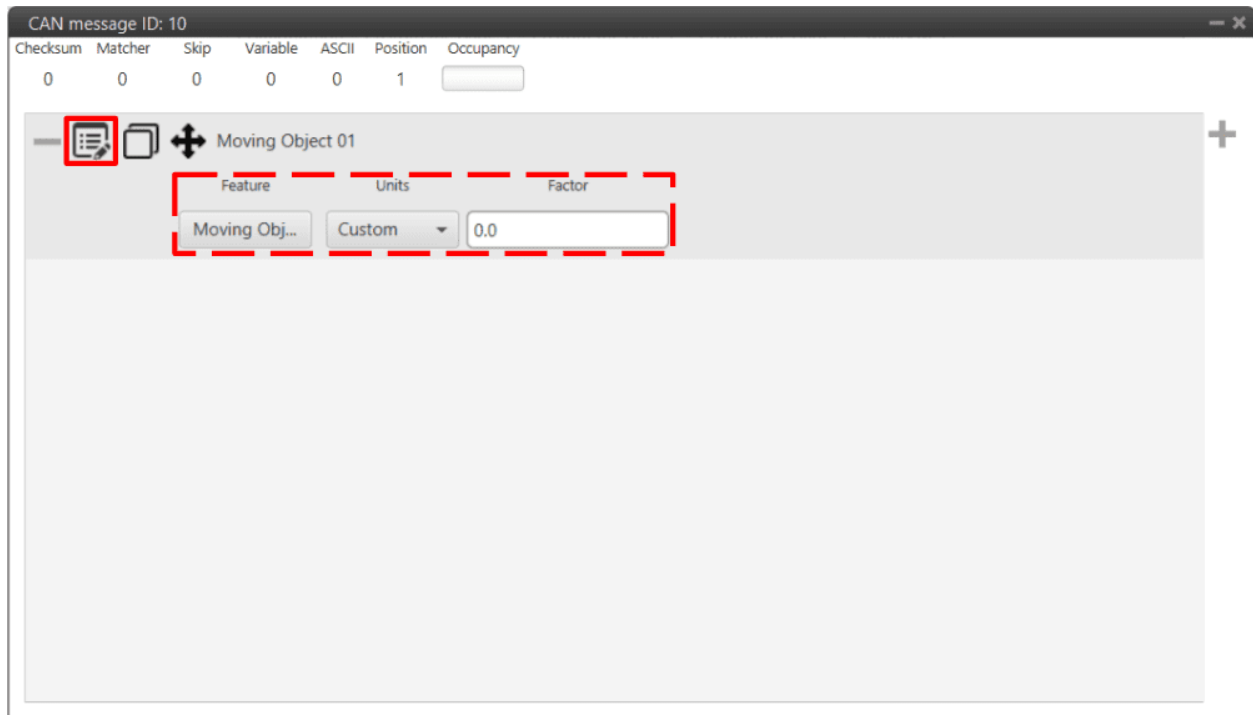


Fig. 99: Position configuration - CAN Custom Message

- **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

GPIO pins can work as Digital Input or Output as well as PWM. In order to configure some custom sensors such as a Stick PPM, Pulse sensor or RPM sensor pins are reserved, which correspond to pins 55-58. These pins can also be used as Digital I/O.

Sensors using a Digital Input are configured in this menu.

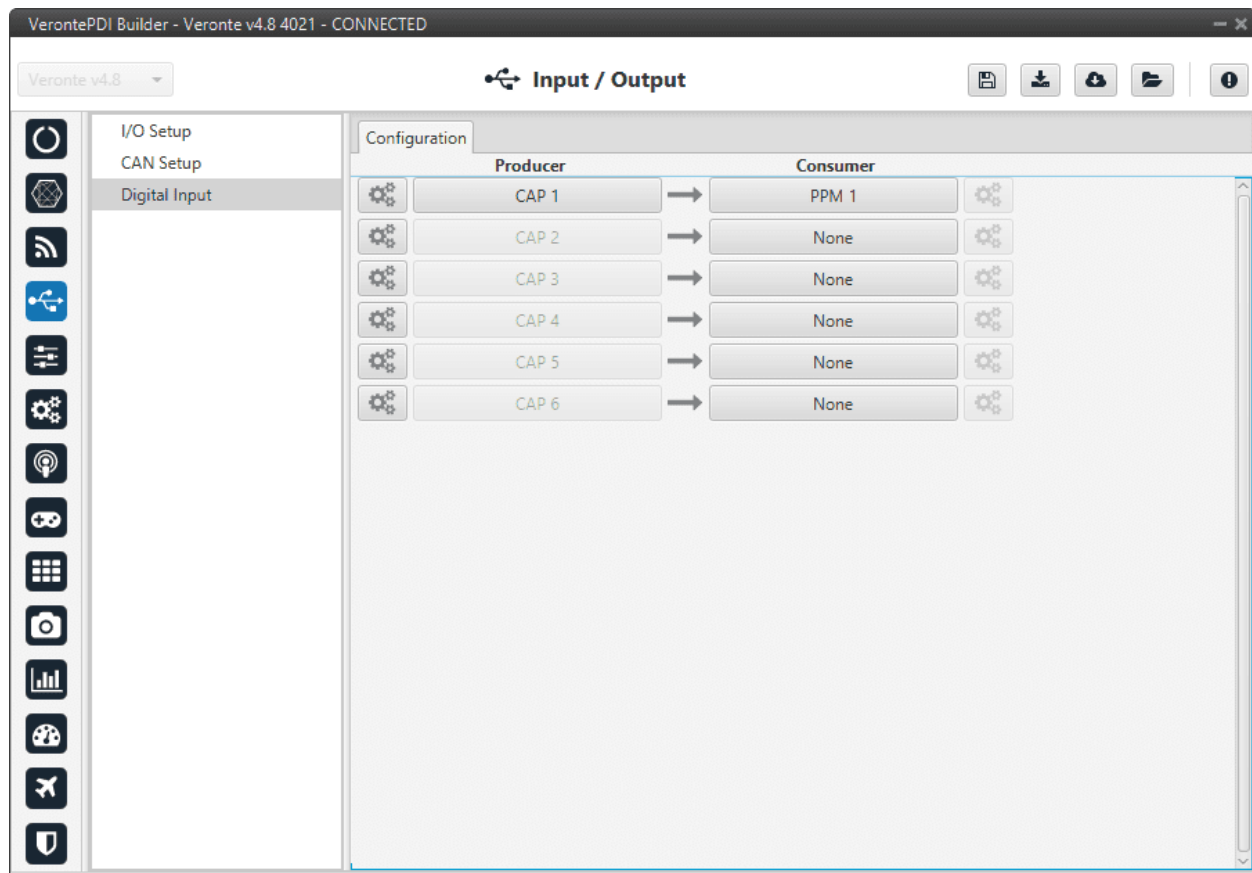


Fig. 100: Digital Input section

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

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

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

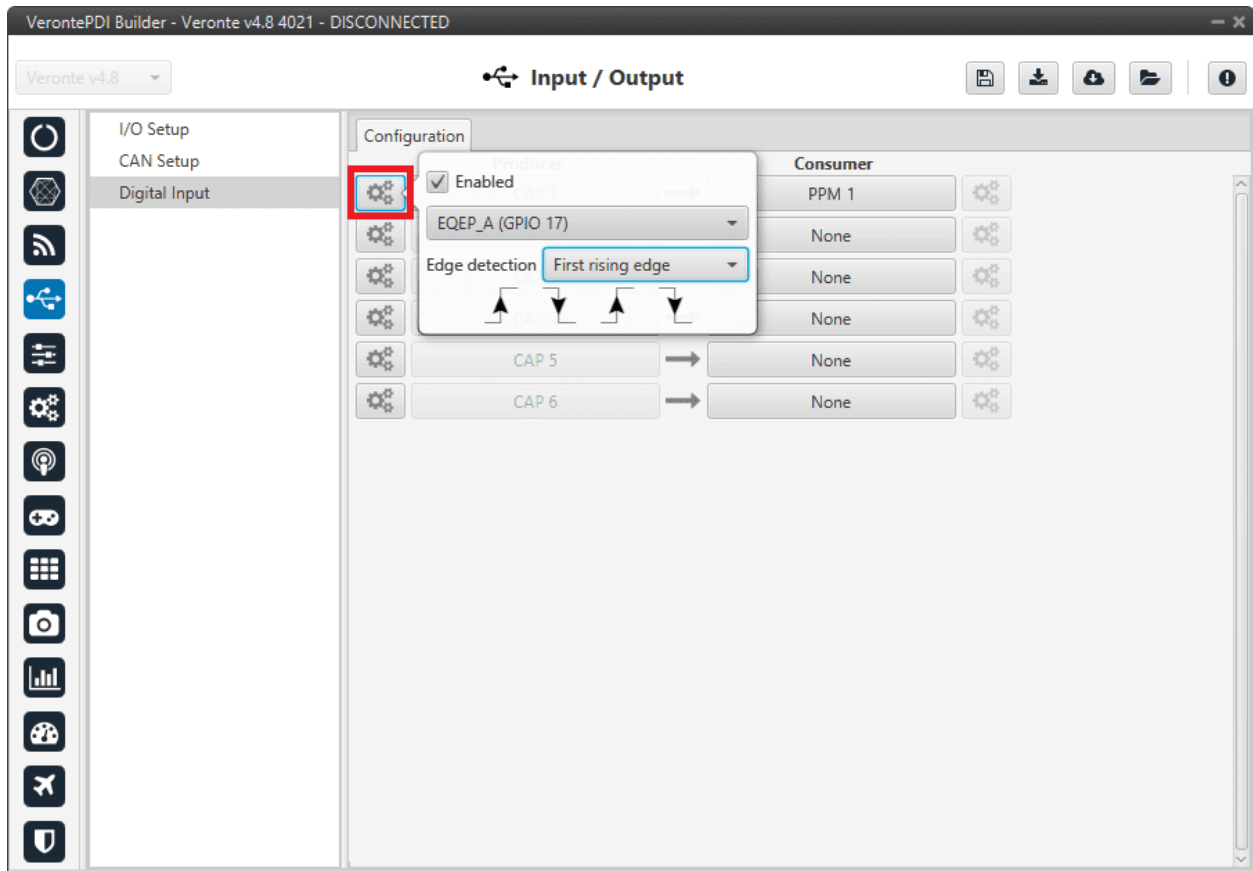


Fig. 101: Digital Input - Producer

In the pop-up window, users can check:

- That this producer is **enabled**.
- Which **pin** this CAP is associated and, therefore, to which device is connected. It is possible to select these pins. Pins available are GPIO 1 to 16, and EQEP A, B, S and I. When using the harness provided by Embention the transmitter Digital Input is connected to the pin 55 (EQEP_A) with pin 49 as Ground.

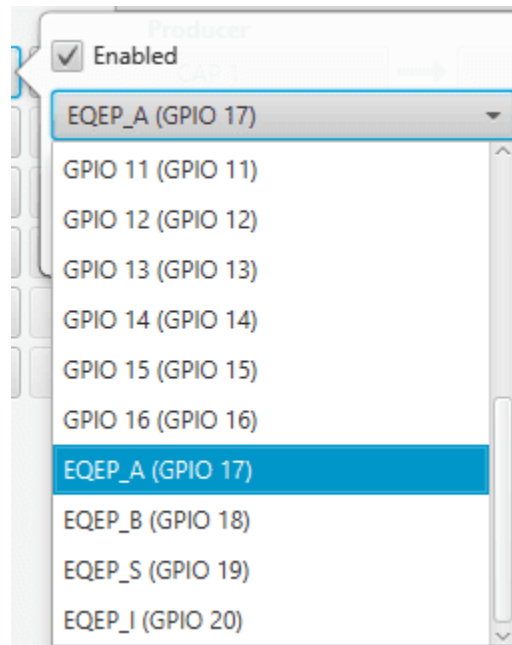


Fig. 102: Digital Input - CAP

- How the pulses are read and transformed into a digital signal (how they are processed). That can be configured with the **Edge detection** option.

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

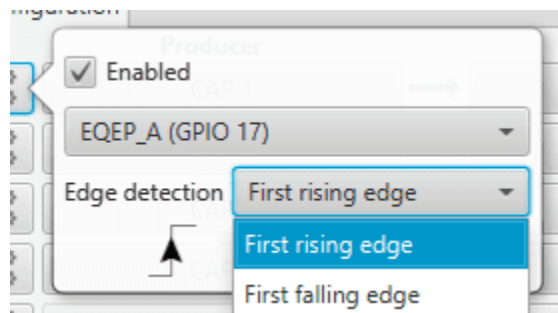


Fig. 103: Digital Input - Edge detection option

- **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. 104: Digital Input - Edge detection option

- Click on the **Bind** button to select the type of **Consumer**, it is possible to choose among a PPM 1-4 (Stick PPM), RPM 1-6 (RPM sensor) or Pulse 1-4 (Pulse).

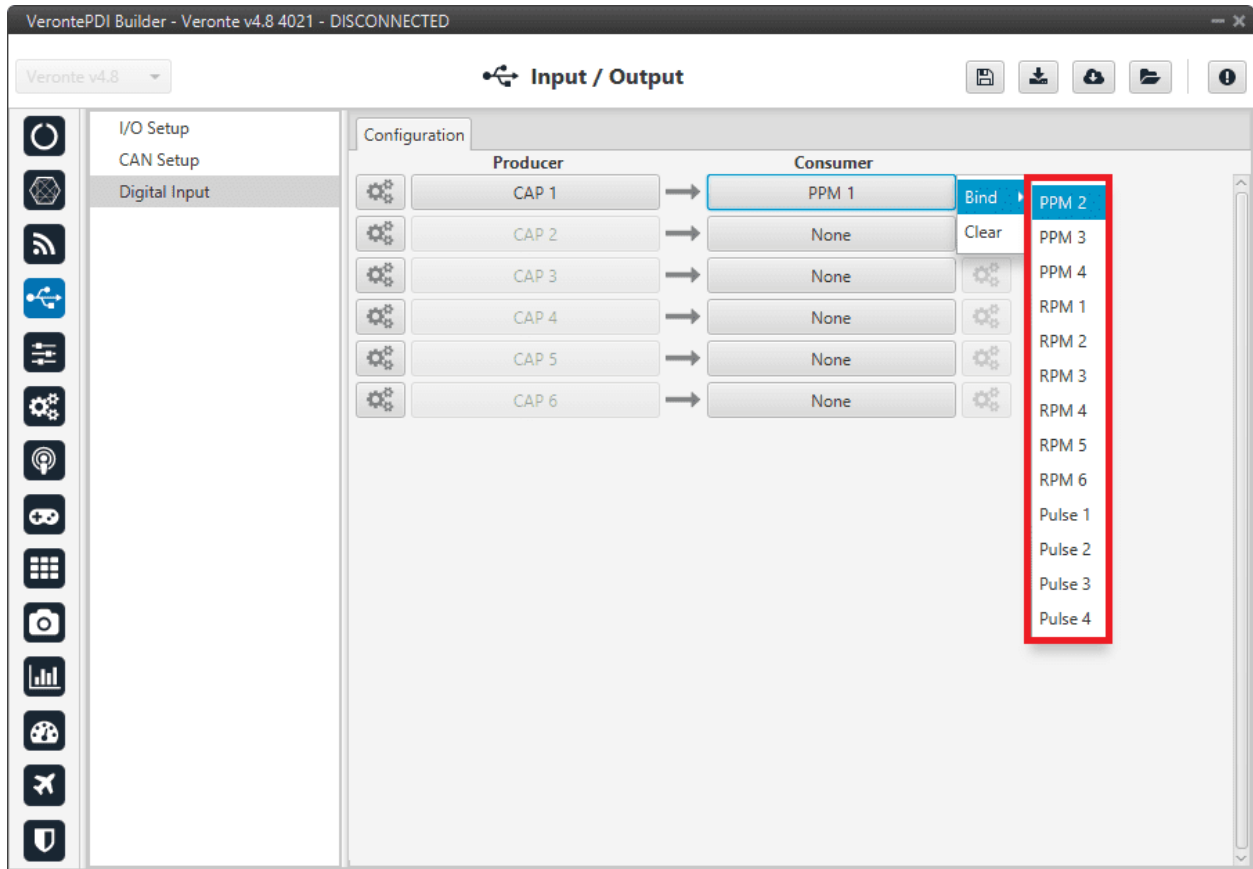



Fig. 105: **Digital Input - Consumer**

- PPM 1-4** selected: PPM is configured in the [Stick menu](#).
- RPM 1-6** selected: The variables in which the information read here is stored are '**RPM 1-6**'. For more information on the configuration of RPM, see the [RPM section](#).
- Pulse 1-4** selected: The variables in which the information read here is stored are '**Captured pulse 1-4**'.

It is possible to configure it clicking on the **configuration button** ( icon):

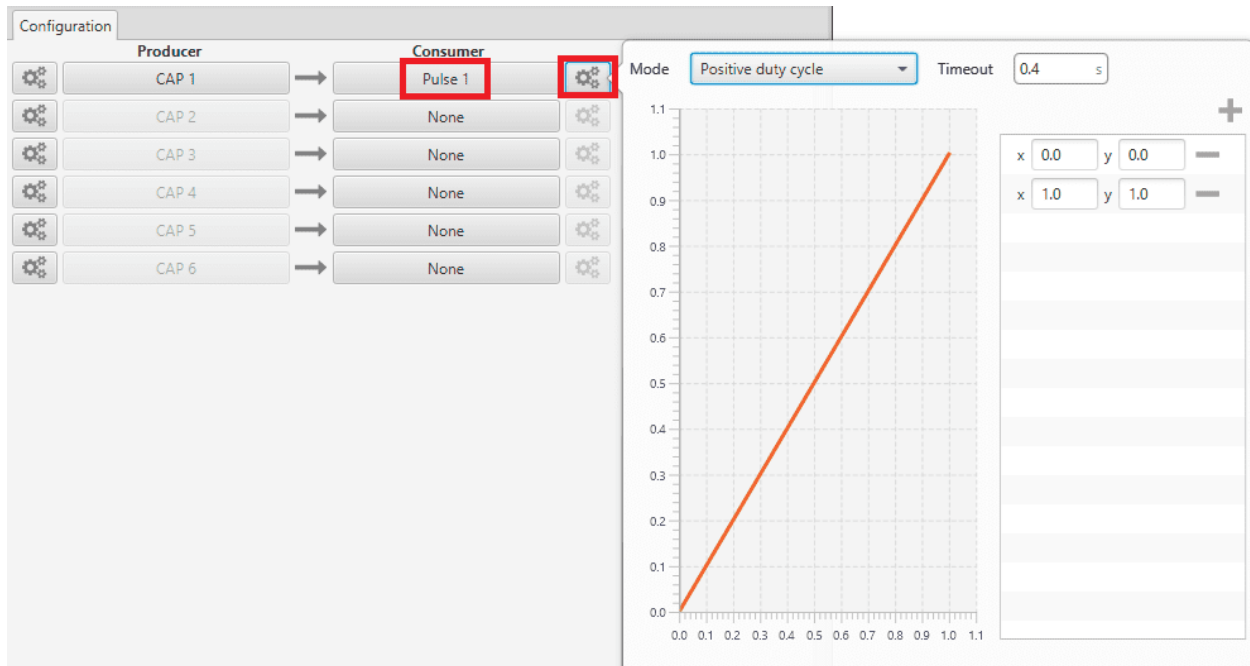


Fig. 106: Digital Input - Pulse

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. 107: 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.

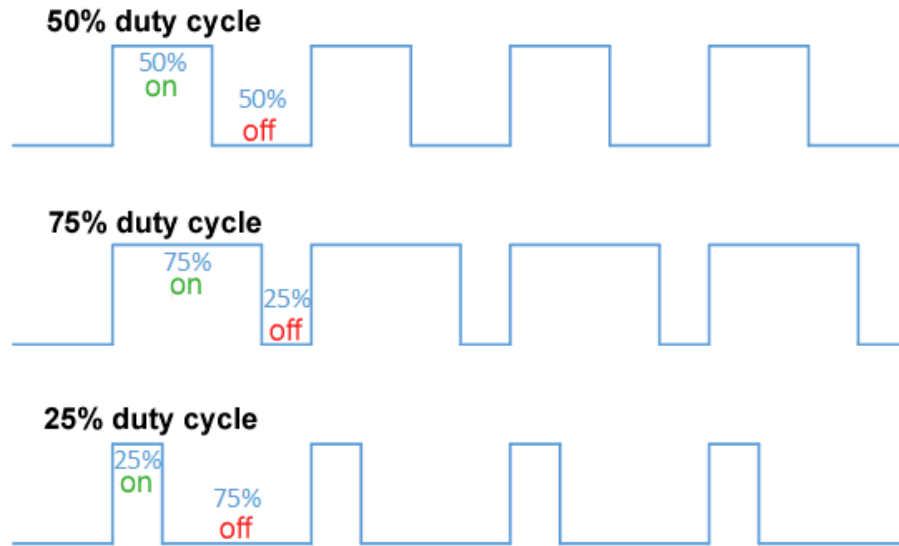


Fig. 108: Positive/Negative duty cycle

- **Time out:** This defines the time to consider that no signal is received.
- **Function:** Here the user can customise 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).

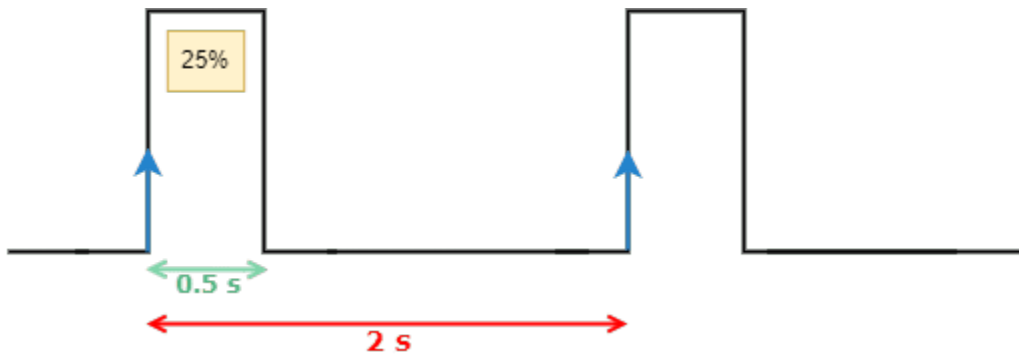


Fig. 109: Signal generated

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.

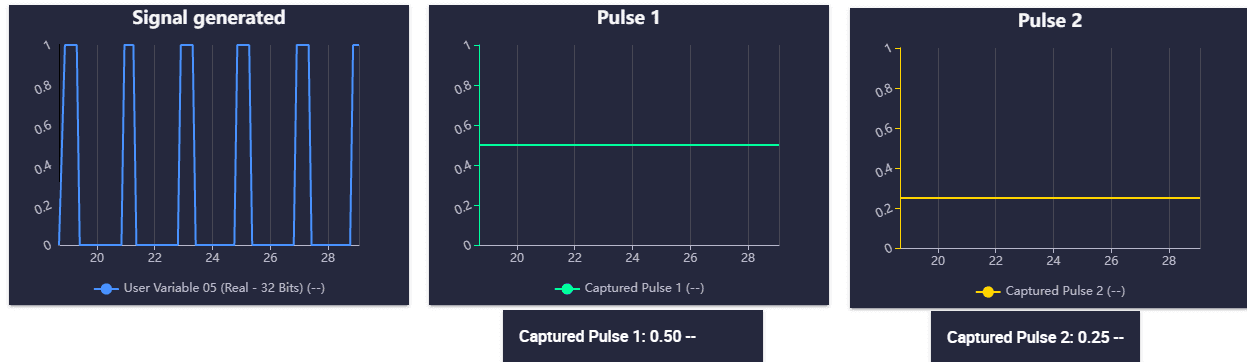


Fig. 110: Digital Input example

2.5 Control

In this panel all the parameters related to the control of the platform can be found. There are 4 sections, each one showing a different menu of configuration.

2.5.1 Phases

In this section are created (defined but not configured) the Flight Phases that will control the aircraft at different stages of the operation.

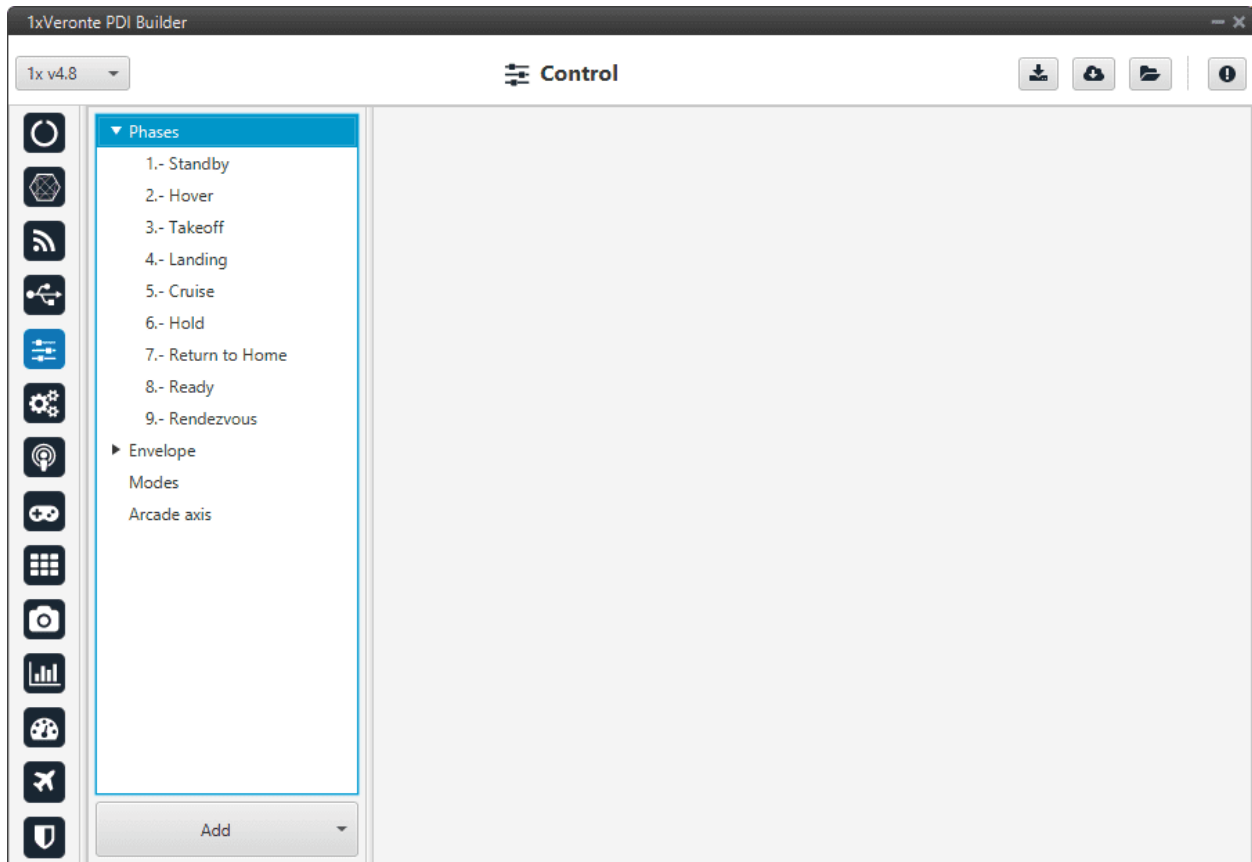


Fig. 111: **Phases menu**

To create a new phase click on *Add*, and select **Phase**.

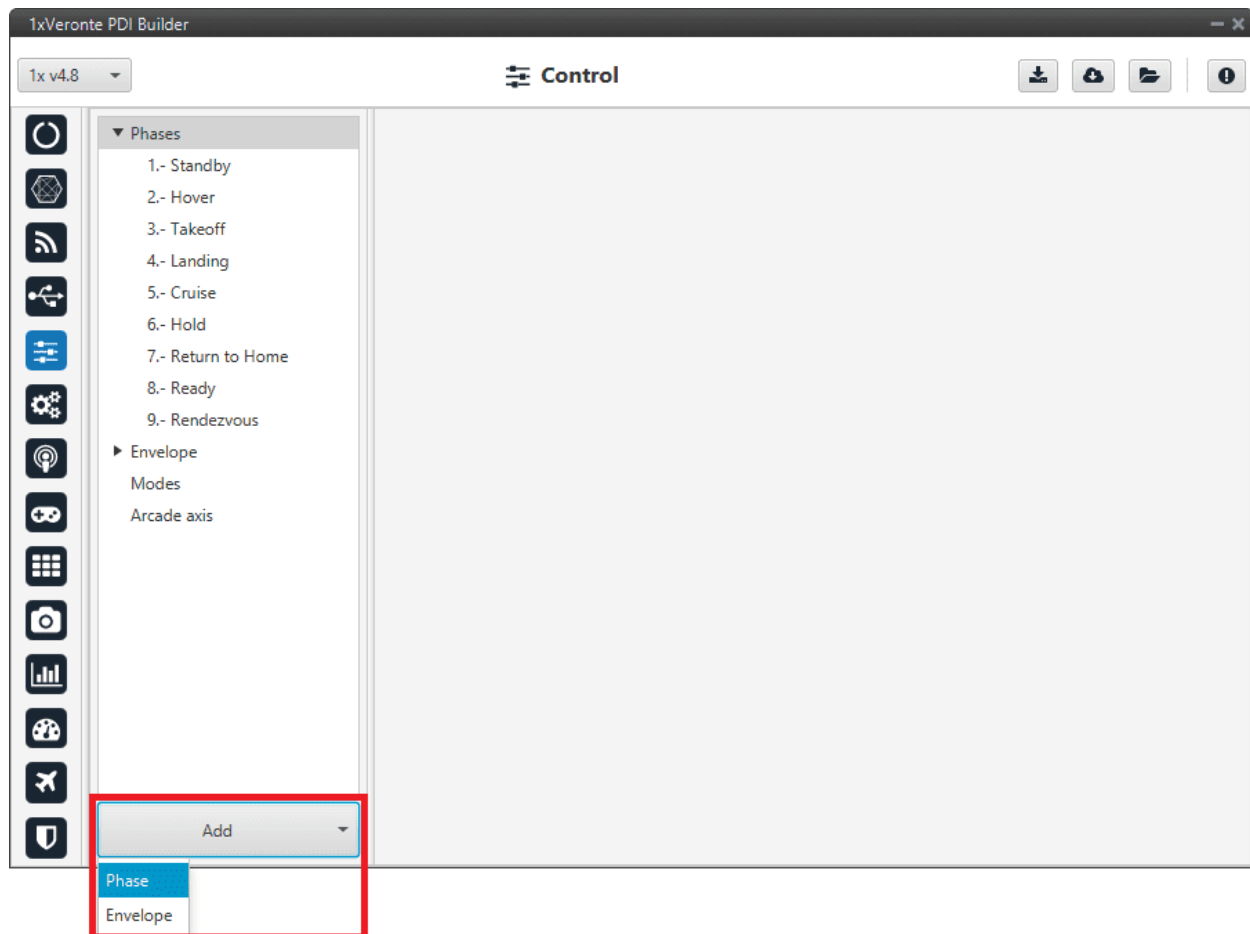


Fig. 112: Add phase

The user can select a phase already created or create a new one.

The screenshot shows a 'Select phase' dialog box. At the top, there is a warning icon and the text 'Customized ID selection'. Below this, there is a label '- New phase ID:' followed by a text input field containing the number '10'. To the right of the input field is a lock icon and the text '9 / 20'. Below the input field is a list of phase options, each preceded by a radio button. The options are: 1 - Standby, 2 - Hover, 3 - Takeoff, 4 - Landing, 5 - Cruise, 6 - Hold, 7 - Return to Home, 8 - Ready, and 9 - Rendezvous. The first five options are grouped within a red rectangular box. At the bottom left of the dialog is a 'Create' button, also highlighted with a red rectangular box. At the bottom right is an 'Accept' button.

Fig. 113: Create phase

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

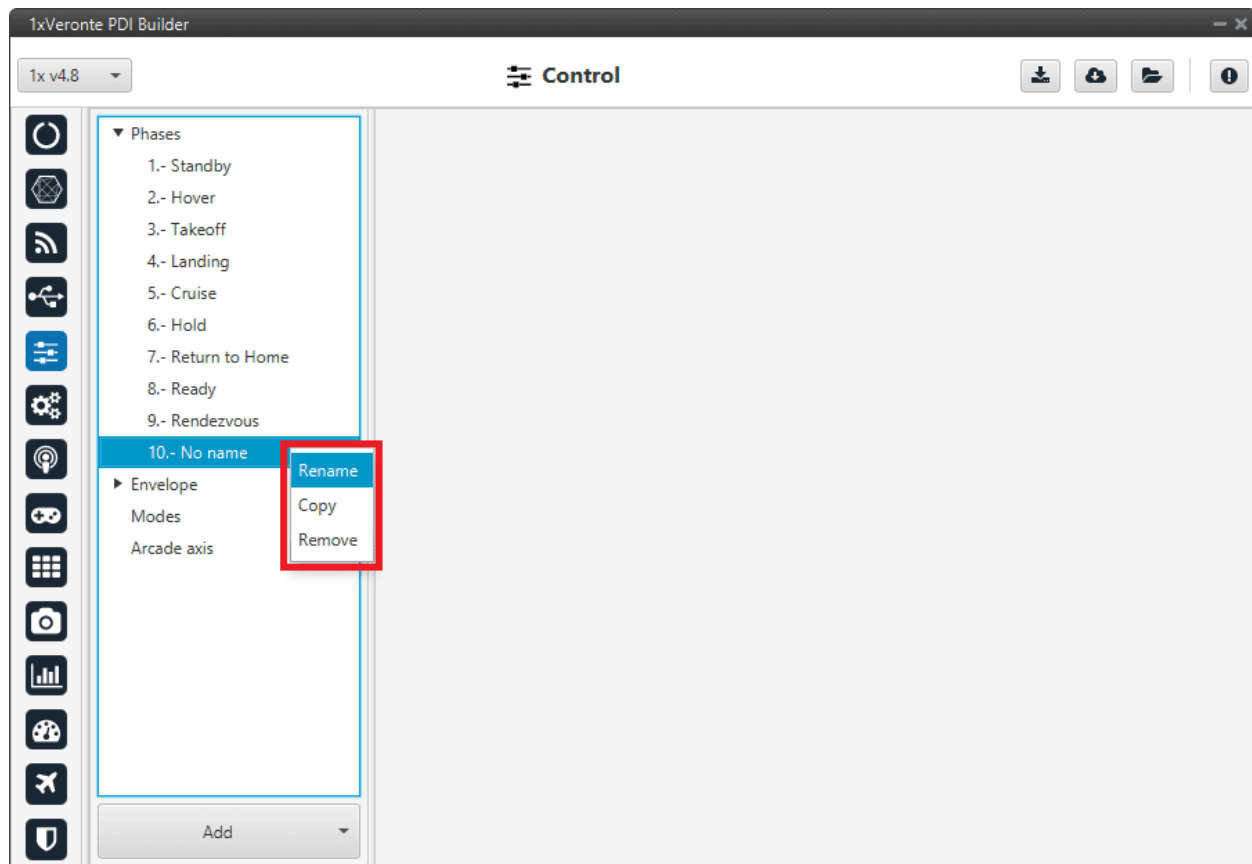


Fig. 114: Phase options

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

2.5.2 Envelope

Menu 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.

Envelope

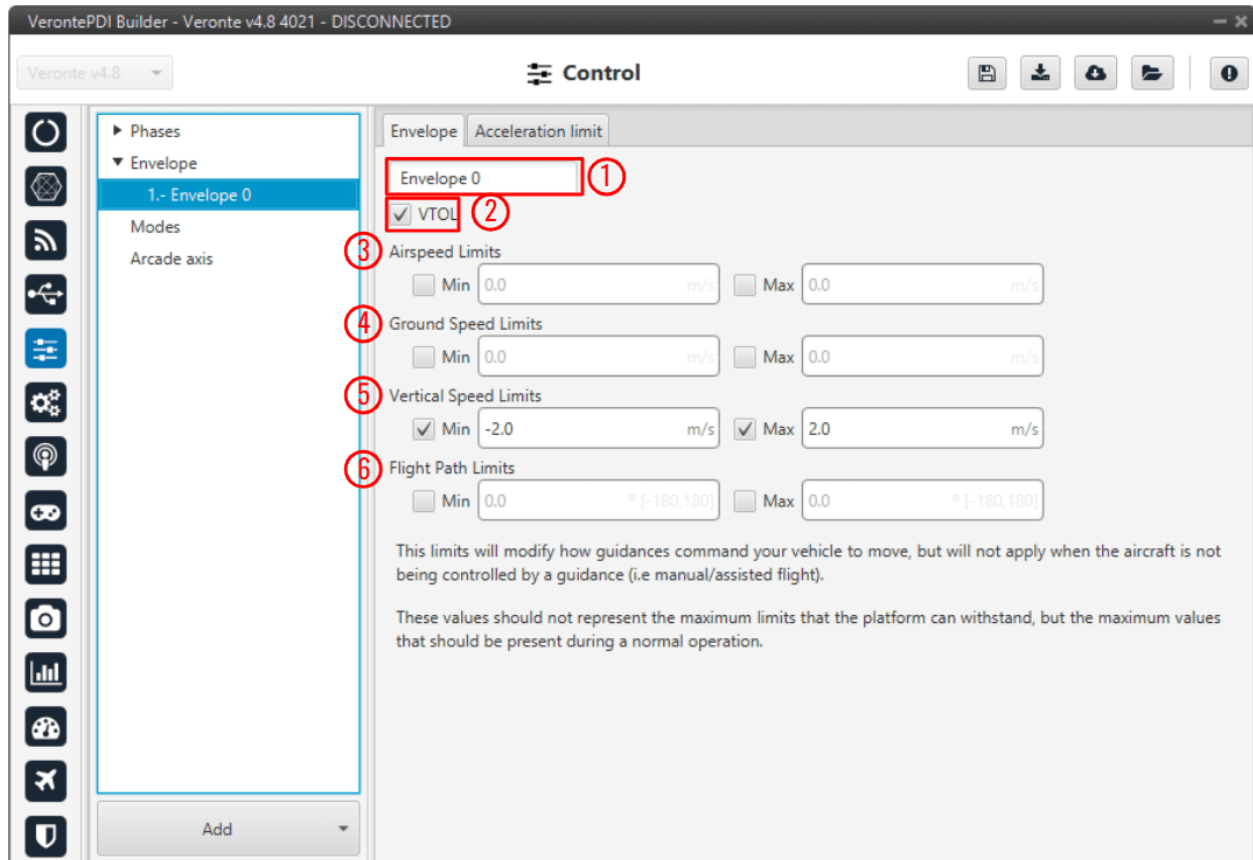


Fig. 115: Envelope menu

1. Envelope name
2. **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.
3. **Airspeed Limits**: Limit for the indicated airspeed (IAS). The value indicated here has effect over the “Cruise” guidance, but is **overridden** if there is a **Hold command** on the IAS, so the user must be careful with the velocity commands.
4. **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.
5. **Vertical Speed Limits**: Similar to the previous limit. It sets the minimum and maximum value for the vertical speed of the platform.
6. **Flight Path Limits**: Maximum and minimum values for the flight path angle (angle of climb or descent).

It's possible to insert multiple envelopes (useful for hybrid configurations) by clicking in *Add*. The change between envelopes can be performed using *Automations*.

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 that stored in previous step. There are six values to define.

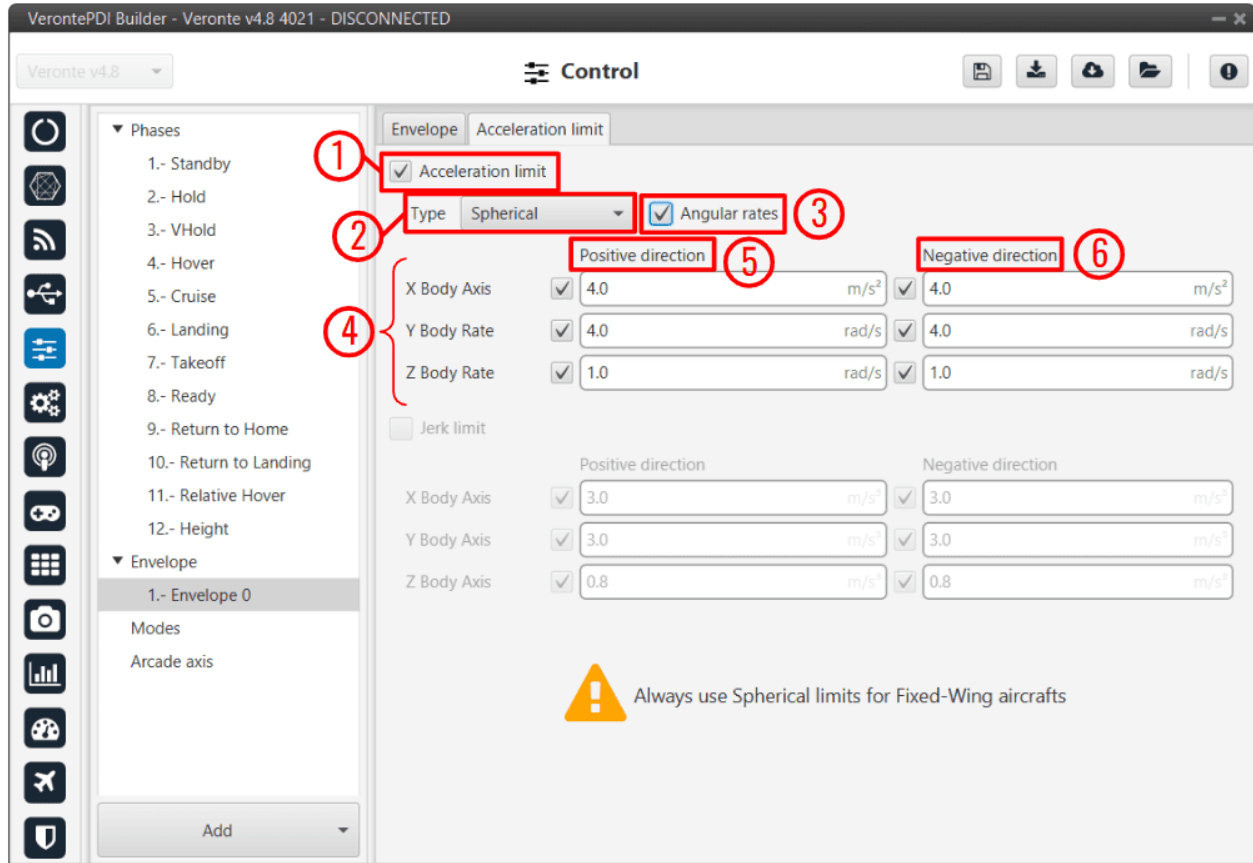


Fig. 116: Acceleration limit menu - Spherical frame

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 behaviour 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).

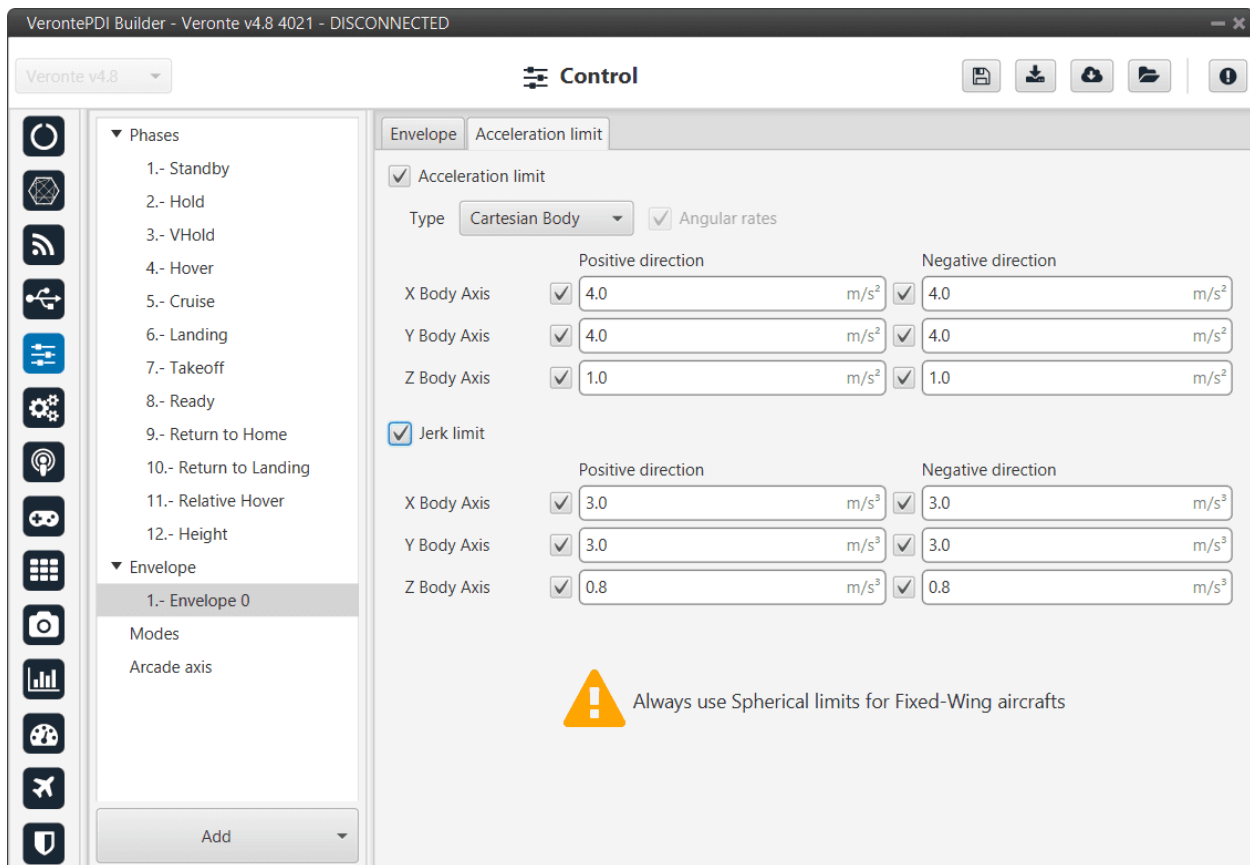


Fig. 117: Acceleration limit menu - Cartesian frame

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 (high values reached).

Fig. 118: 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. 119: Acceleration limit (Thrust)

To compare acceleration and jerk, roll axis is chosen. In the first gif below only the first limit is applied.

Fig. 120: 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. 121: **Acceleration limit (Cartesian frame)**

2.5.3 Modes

Modes

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

There are 5 different control modes and it is possible to combine them to create custom flight modes.

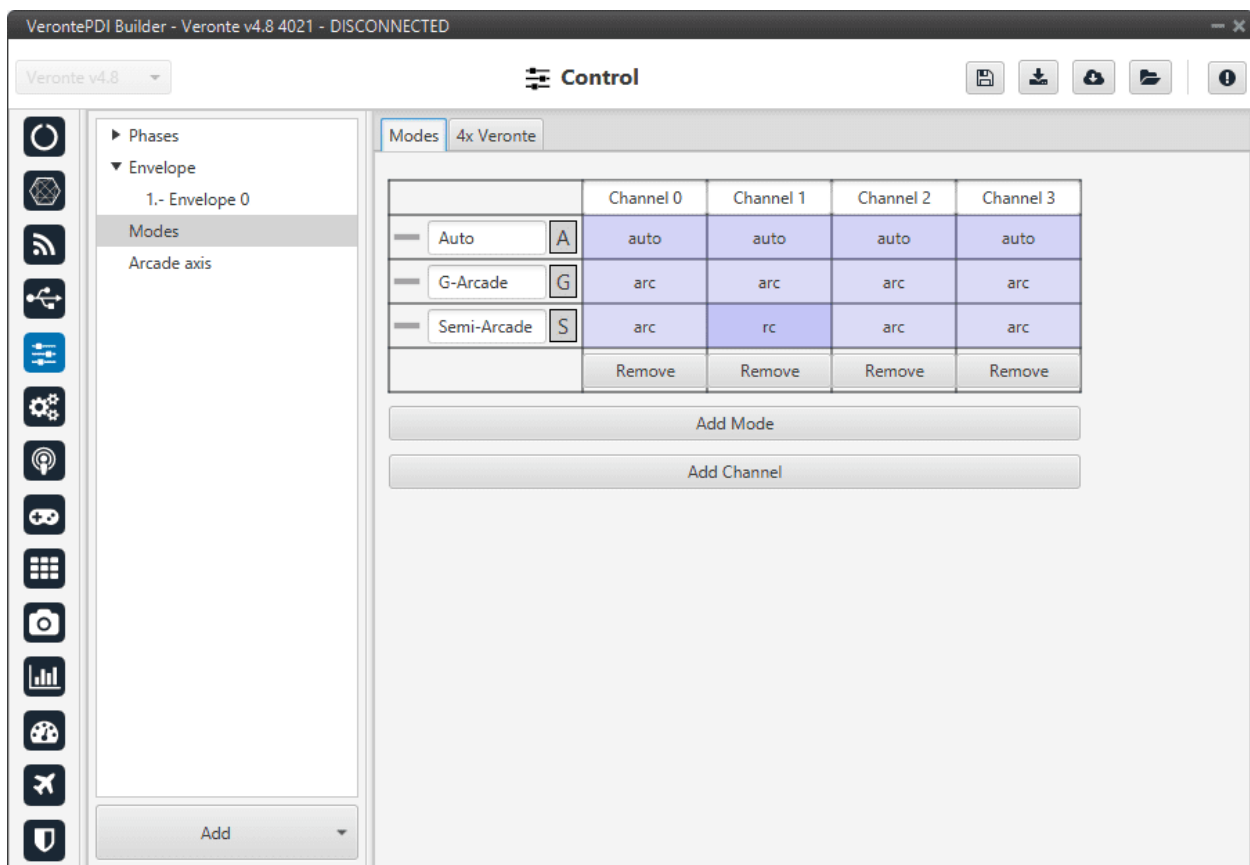


Fig. 122: **Modes menu**

The options available are:

- **Automatic:** The control channel is controlled totally by the autopilot.
- **RC:** 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:** 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):

		Channel 0	Channel 1	Channel 2
—	Auto	A	rc	rc
—	Arcade	G	auto	auto
		Remove	Remove	Remove

Fig. 123: Modes configuration

- Moreover, although the mode is set “sensefully” here, in the block configuration (*Block Programs*) 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:

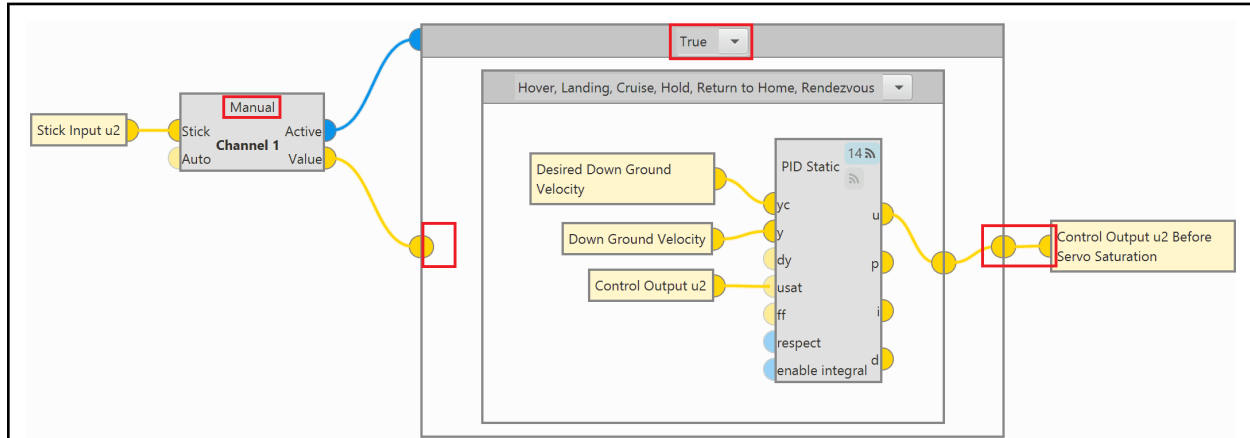


Fig. 124: Modes configuration in blocks

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).

4x Veronte

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

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.

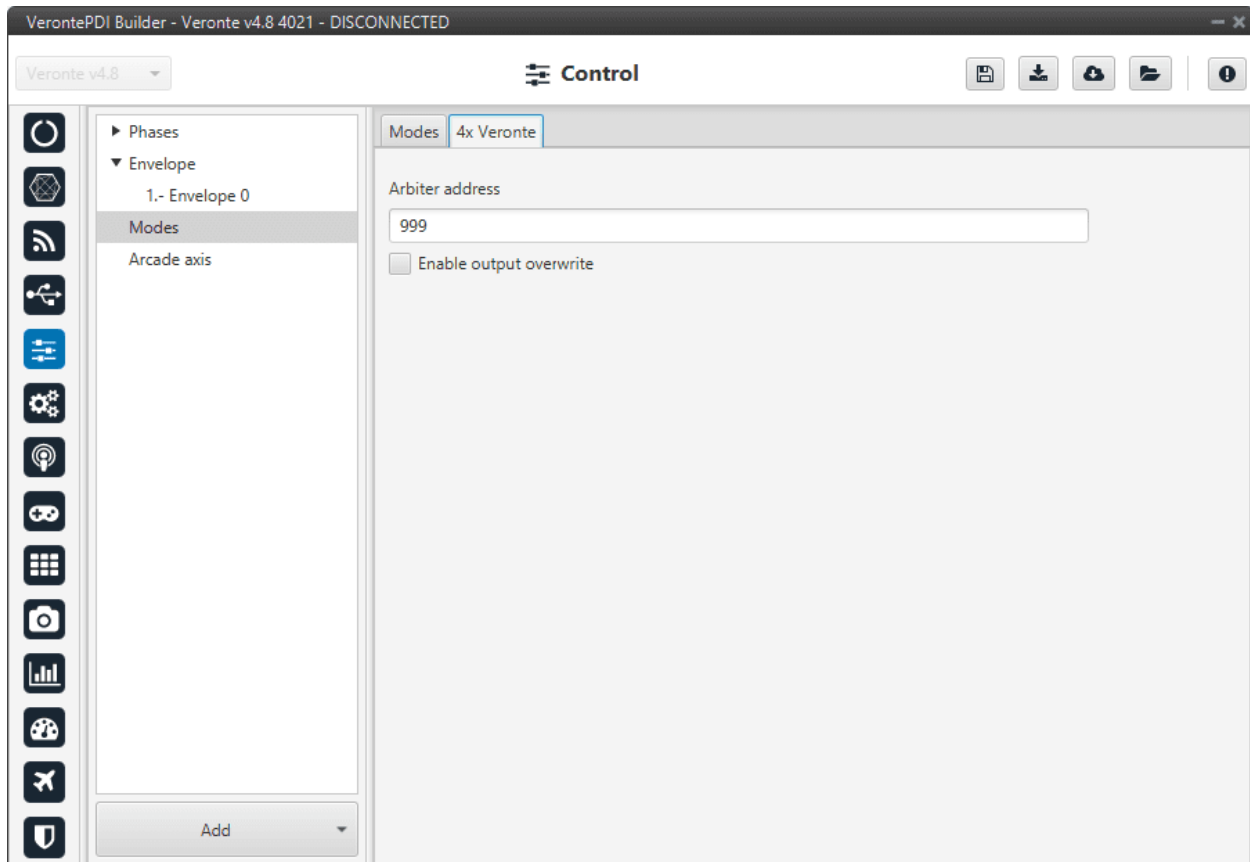


Fig. 125: 4x autopilot menu

To allow the **output to be overwritten**, the checkbox must be checked.

By enabling it, a table can be created in which columns correspond to each 1x Autopilot and rows to the different channels. For each channel and autopilot, a variable should be selected.

This option must work in conjunction with the [AP Selection block](#). In this block, the output “Value” will be the variable associated to the selected channel and autopilot.

In the “**Selected**” field, the selected variable is **4xV Veronte selected**, which indicates which Autopilot 1x is selected. This information is received from the status message from the arbiter (for more information on the this status message, see [CAN Bus protocol section](#)) of the **4x Software Manual**.

An example configuration is presented below:

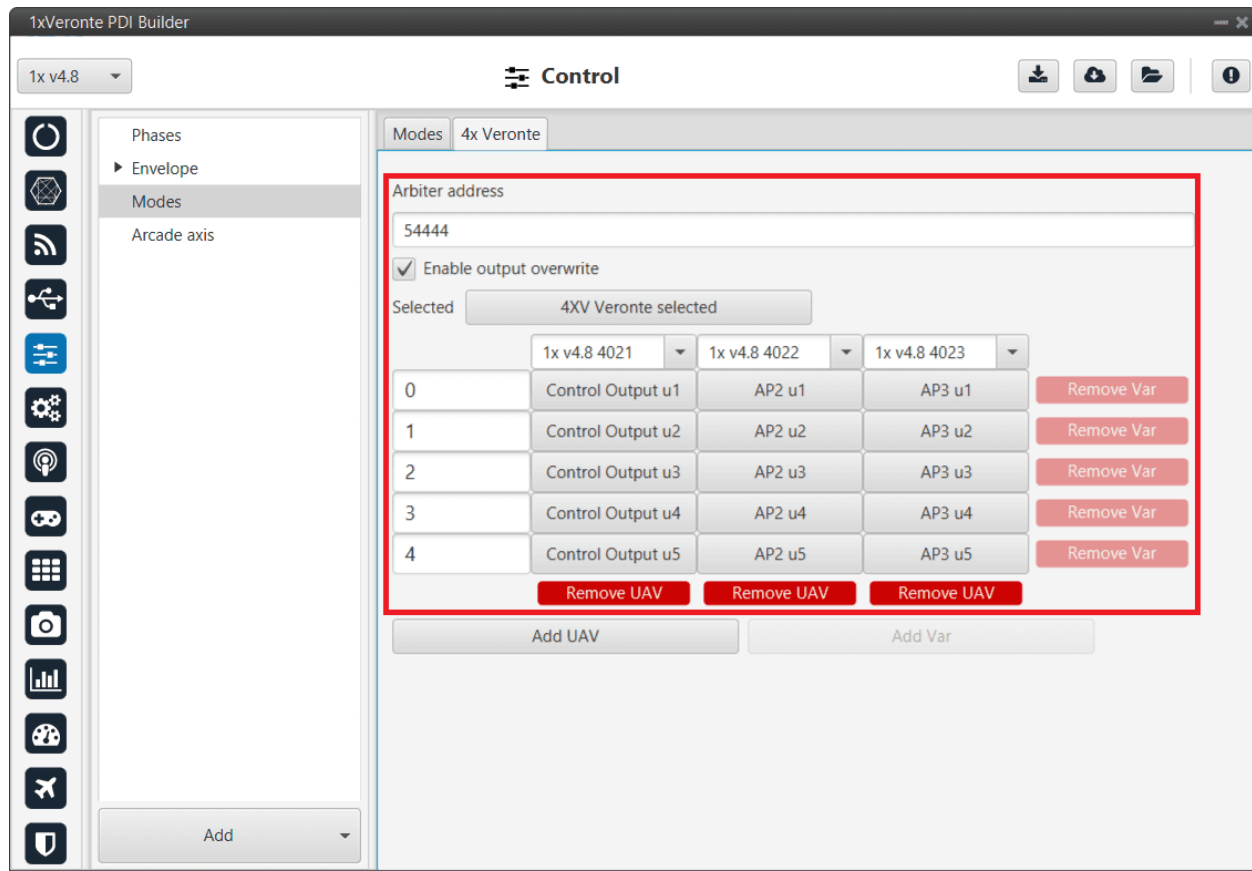


Fig. 126: 4x autopilot menu - Example of use

2.5.4 Arcade axis

The Arcade Axis menu 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 movement to be made with respect him (Ground axes). In this way, if the pilot command a turn right, the aircraft will turn to the right of the pilot, instead the right of the aircraft (Body axes).

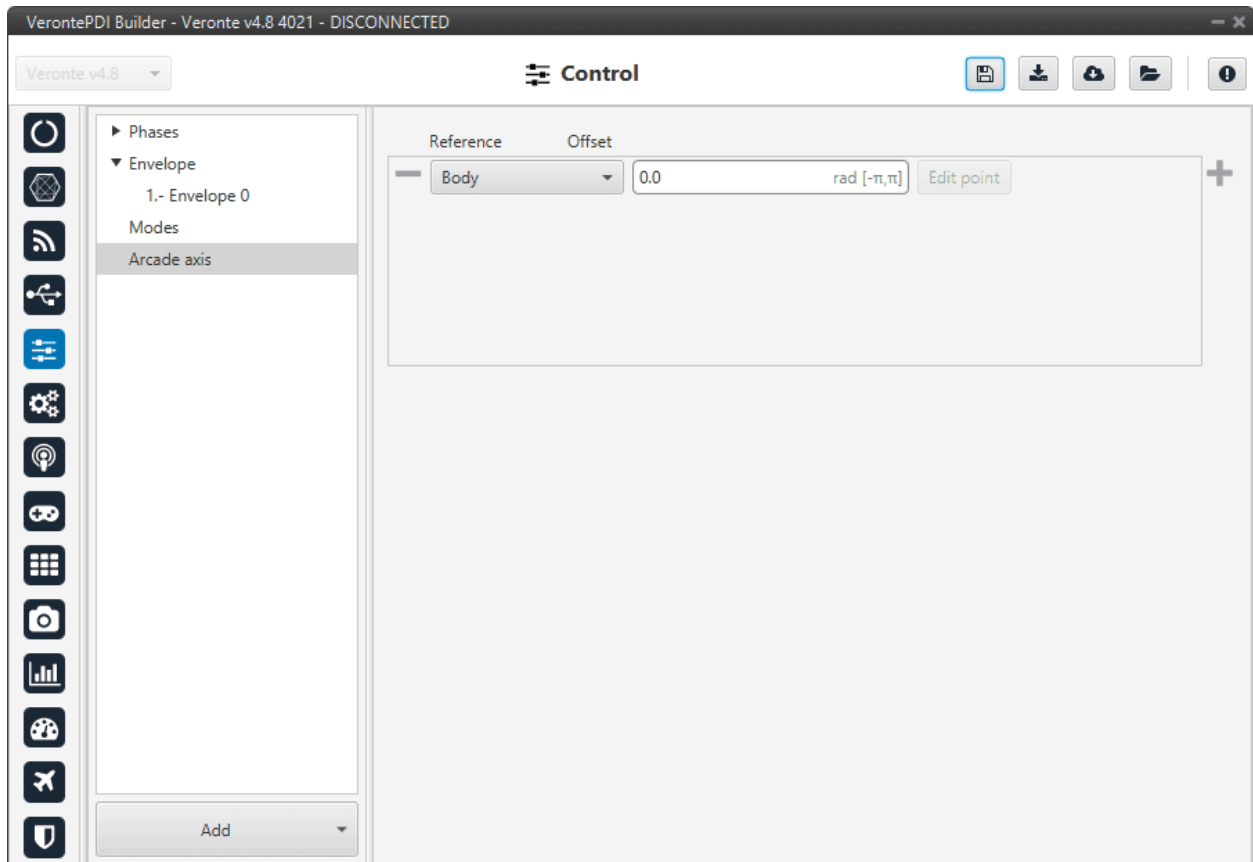


Fig. 127: Arcade axis menu

It is possible to add as many axes system as desired, 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, see [Actions](#) in Automations section.

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.

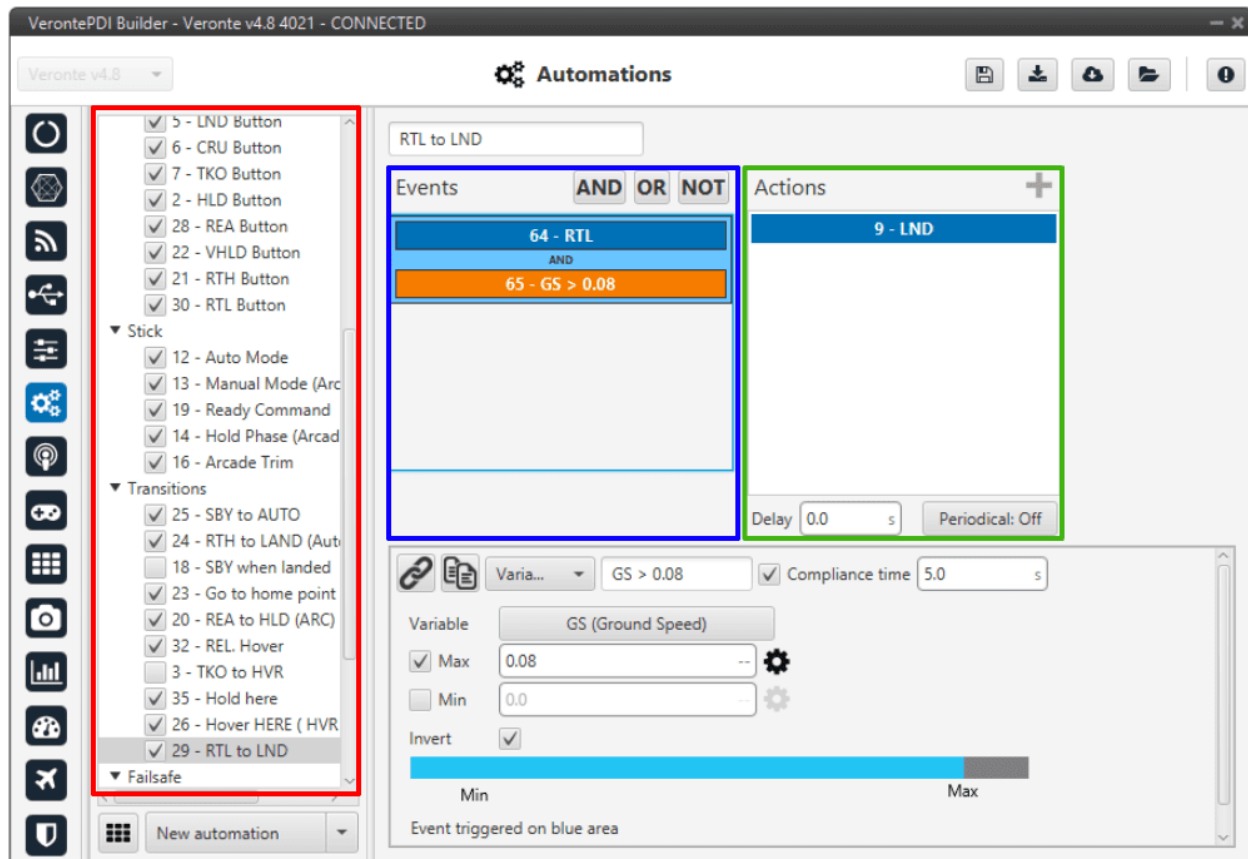


Fig. 128: Automations menu

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.

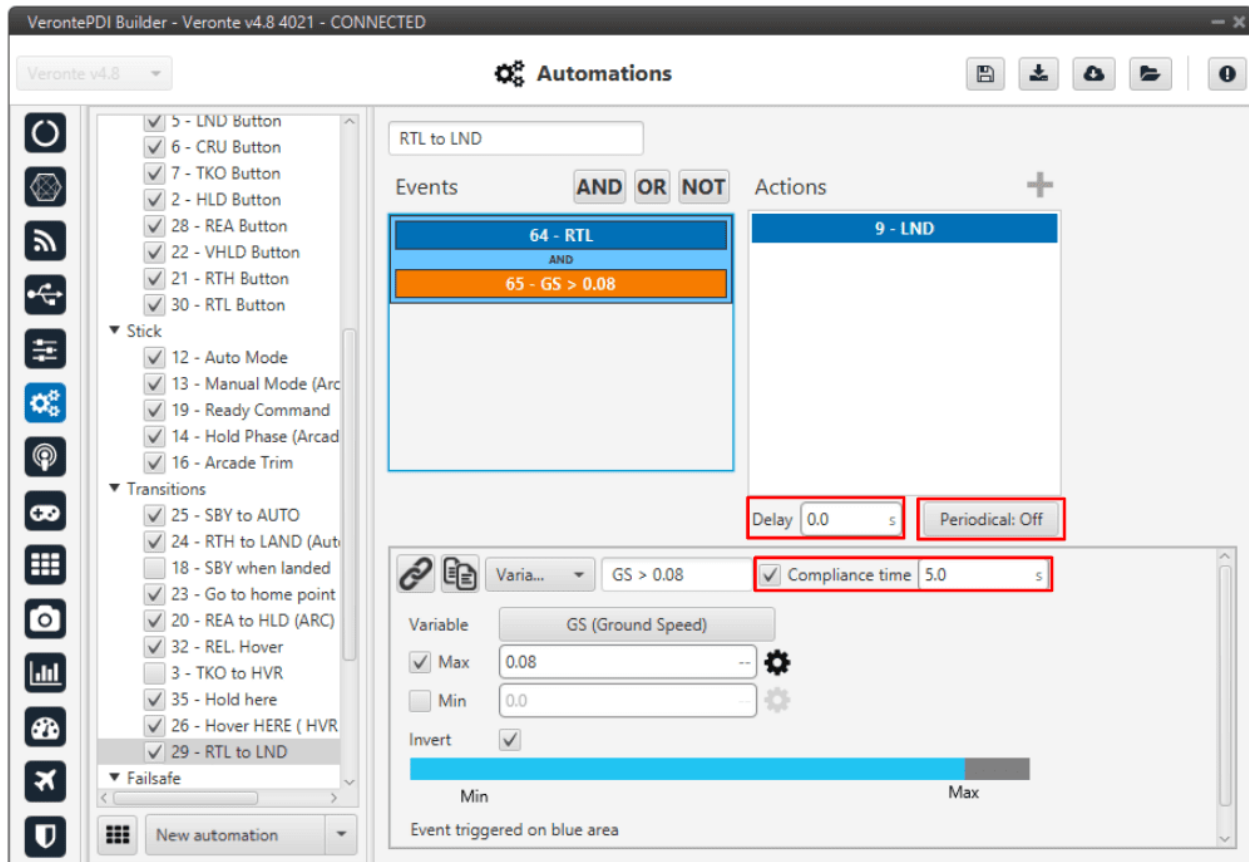


Fig. 129: Automations parameters

- **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:** This menu is used to configure actions to take place periodically during the time that the events are active. 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.

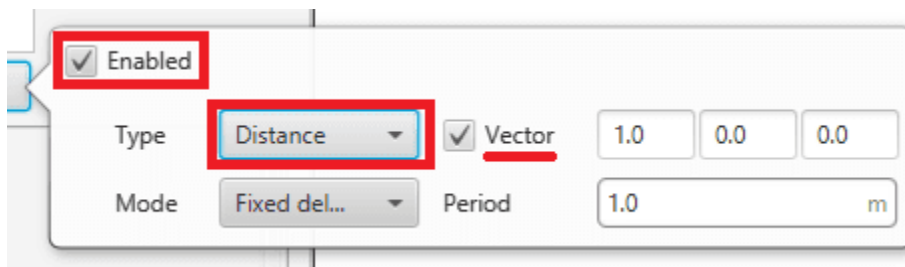


Fig. 130: Periodical distance menu

- Time

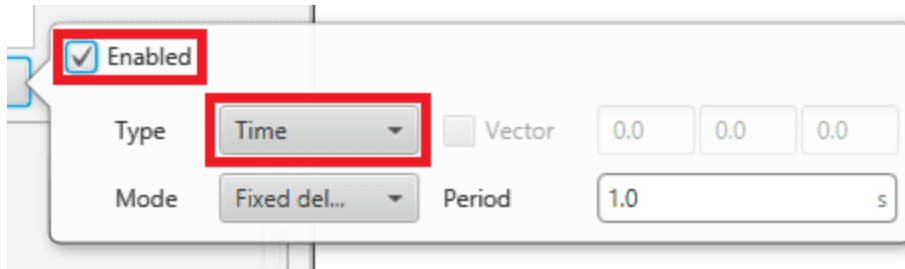


Fig. 131: Periodical time menu

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.

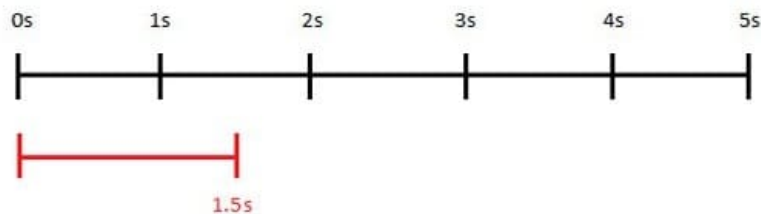


Fig. 132: Periodical modes

Let's consider that the system evaluates the automations each second (black line), and the automation that contains the periodical option is wanted to execute each 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. Users can select a **previous one** (if exists) or **Create new**.

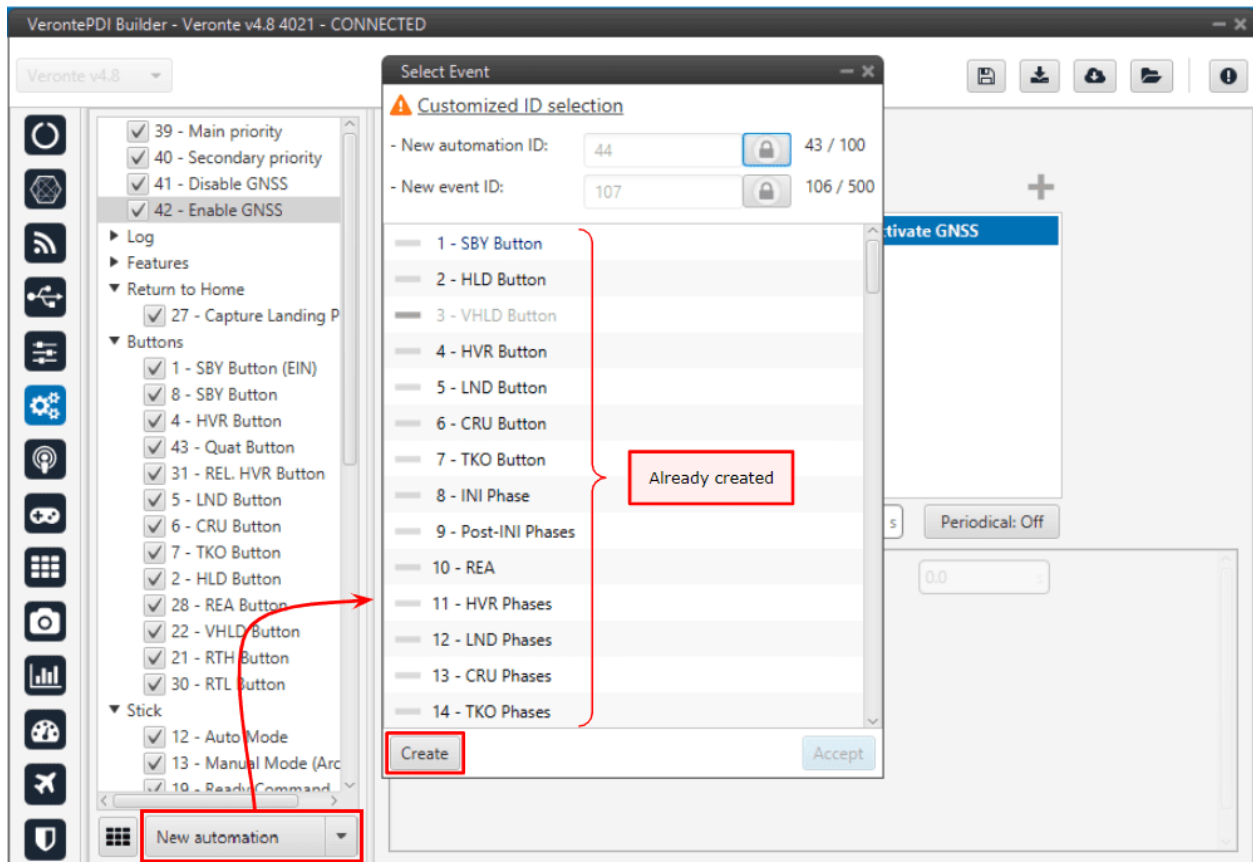


Fig. 133: New automation

To add an action to the automation, press “+” icon and a new window will be displayed. Users can select **previous one** (if exists) or **Create** new.

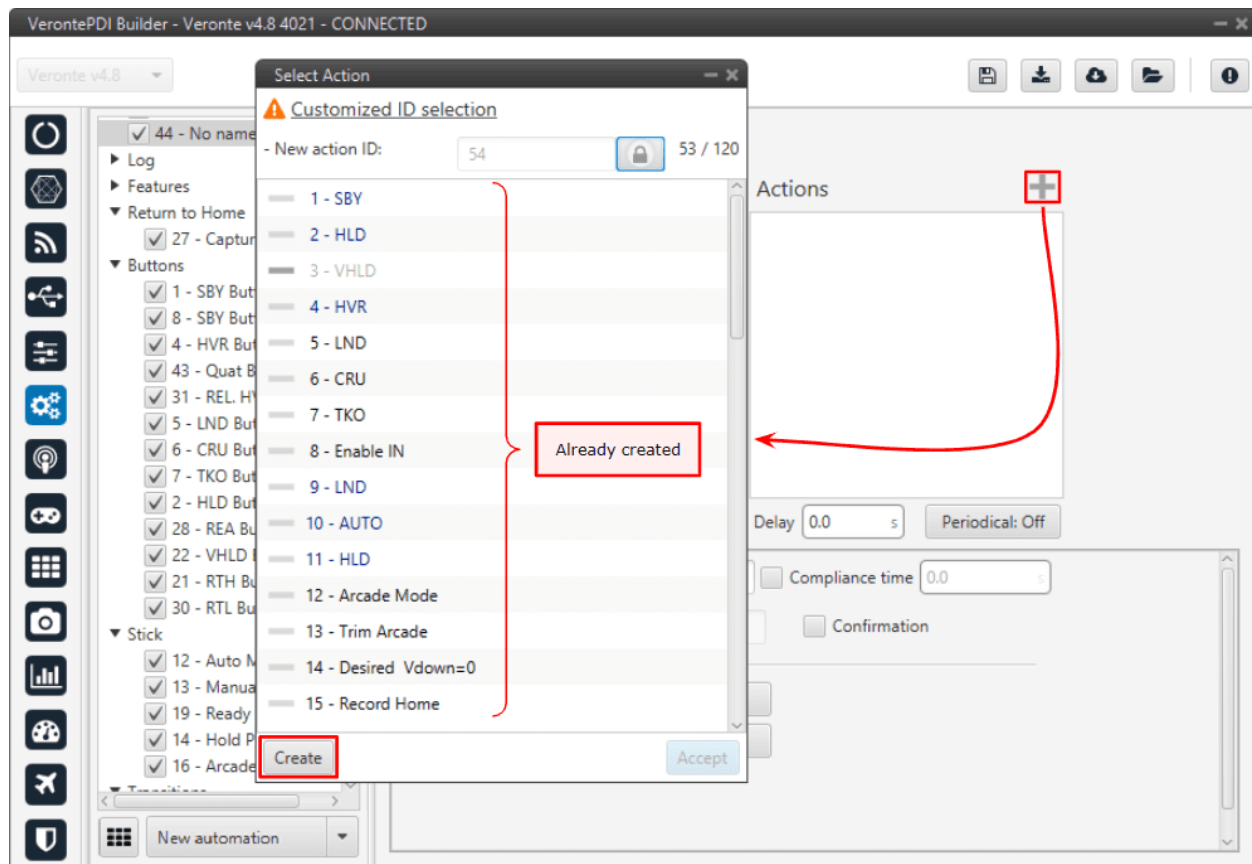


Fig. 134: New action

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

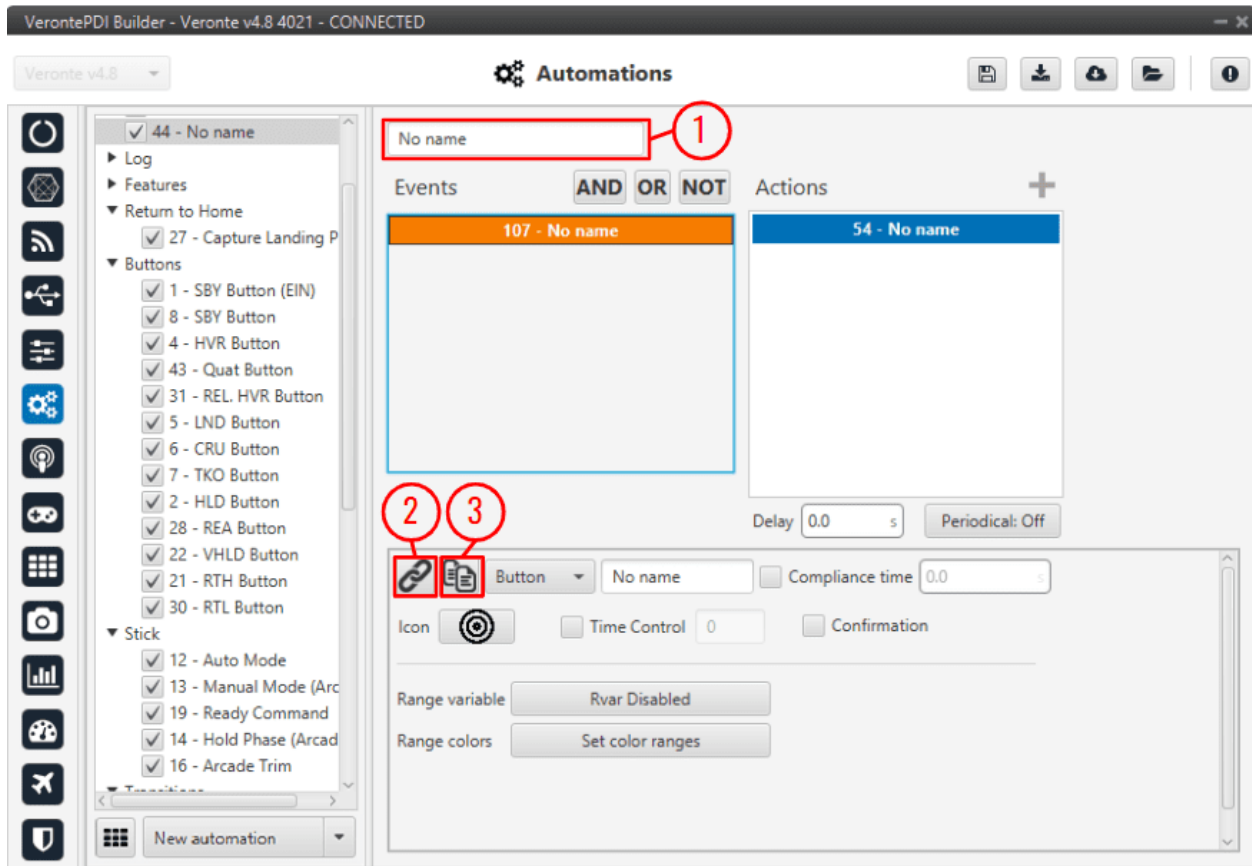




Fig. 135: New automation options

1. The user can rename the automation with the name of his choice.
2. **Use existing** button ( icon): 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 ( icon): 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.

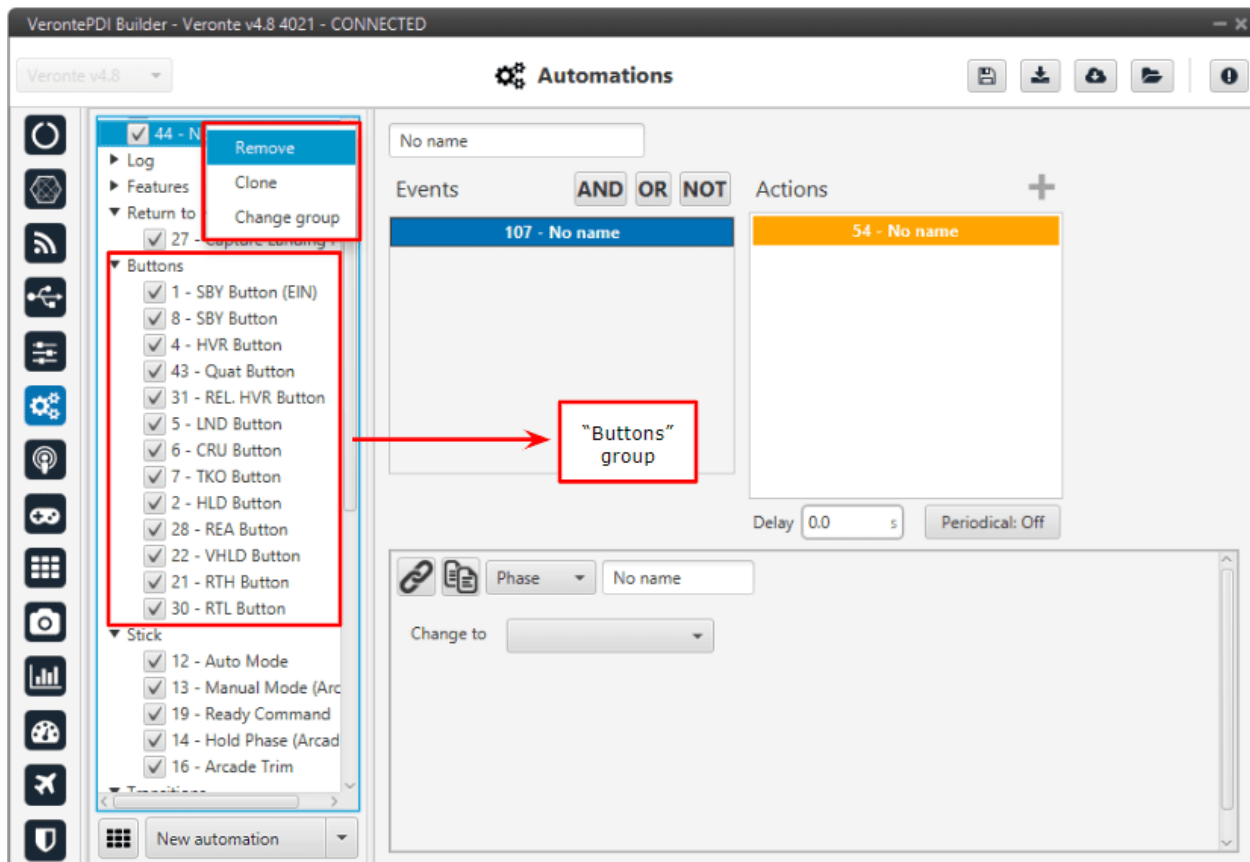


Fig. 136: Automations groups

Warning: When a clone of an automation is created, the 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:

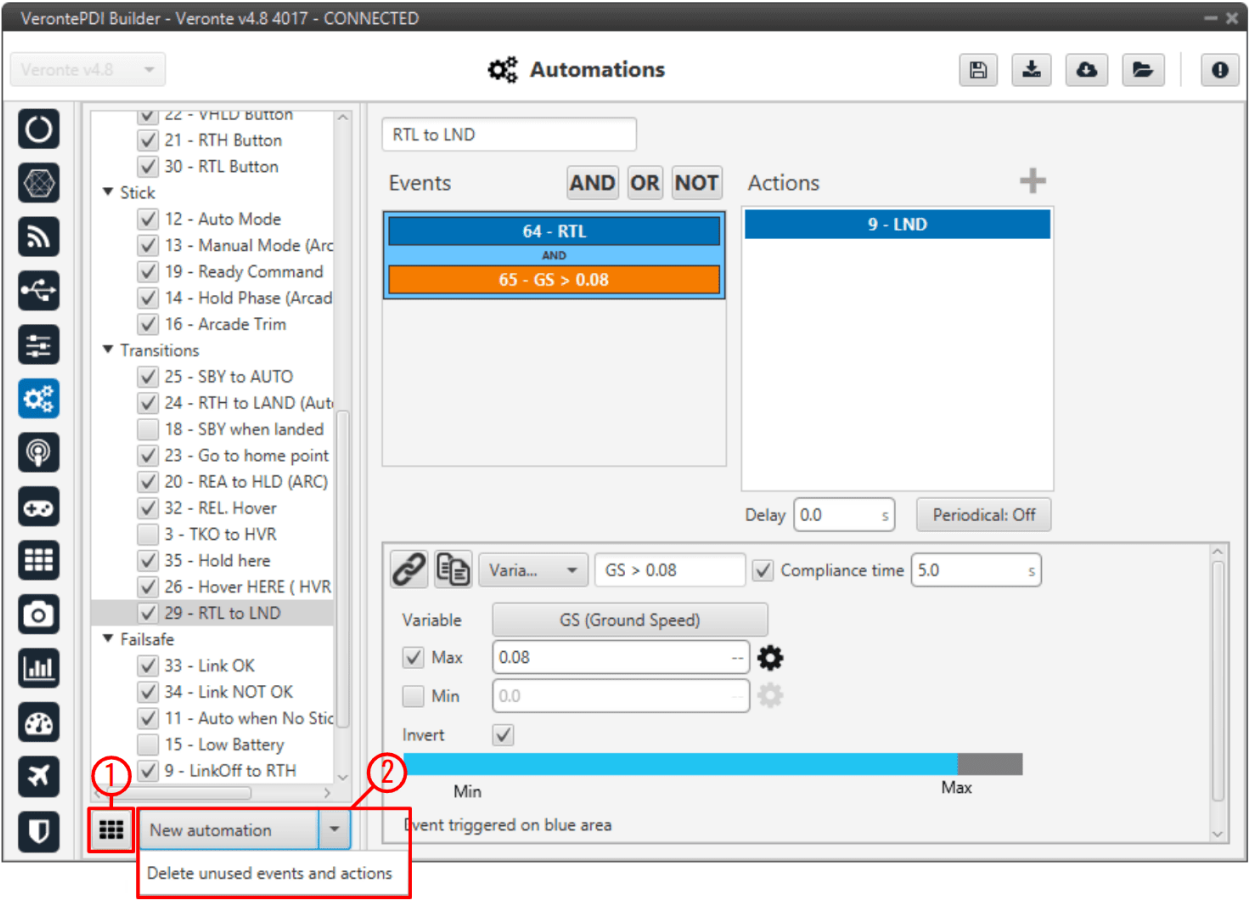


Fig. 137: Automations options

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

Transitions Table														
Origin	Destination													
	Fases	Init	Standby	Hold	VHold	Hover	Cruise	Landing	Takeoff	Ready	Return to Home	Return to Landing	Relative Hover	Height
	Init													
	Standby													
	Hold													
	VHold													
	Hover													
	Cruise													
	Landing													
	Takeoff													
	Ready													
	Return to Home													
	Return to Landing													
	Relative Hover													
	Height													
	Quaternion Test													

Fig. 138: Phase transition table

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

In addition, by clicking on a cell, it is possible to see which automation (and the events) makes the transition

between the two phases possible.

Transitions Table					
Fases	Init	Standby	Hold	VHold	Hover
Init					
Standby					
Hold					
VHold					
Hover					
Cruise					
Landing					
Takeoff					
Ready					
Return to Home					
Return to Landing					
Relative Hover					
Height					
Quaternion Test					

Fig. 139: Phase transition table - automation

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

2.6.2.1 Events

An event is something that has to be accomplished to trigger the actions. All the events can be combined to create a custom event, using the **boolean operations** provided by the software (**AND**, **OR**, **NOT**).

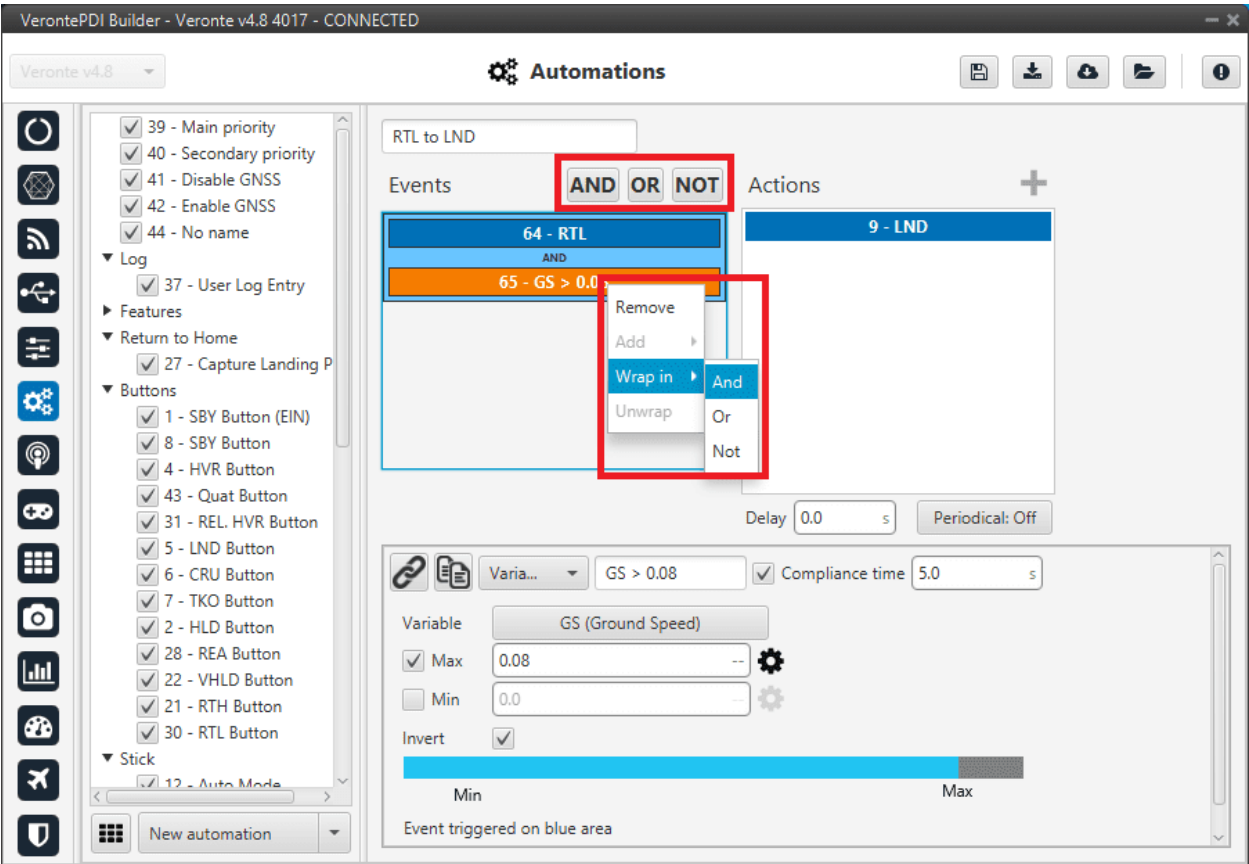


Fig. 140: Events options

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.

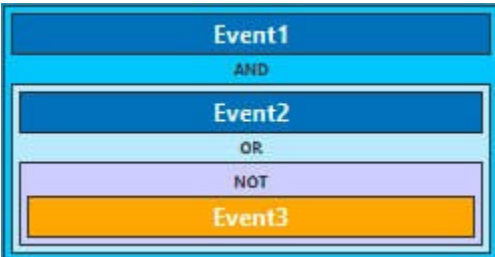


Fig. 141: Events wrapped

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.

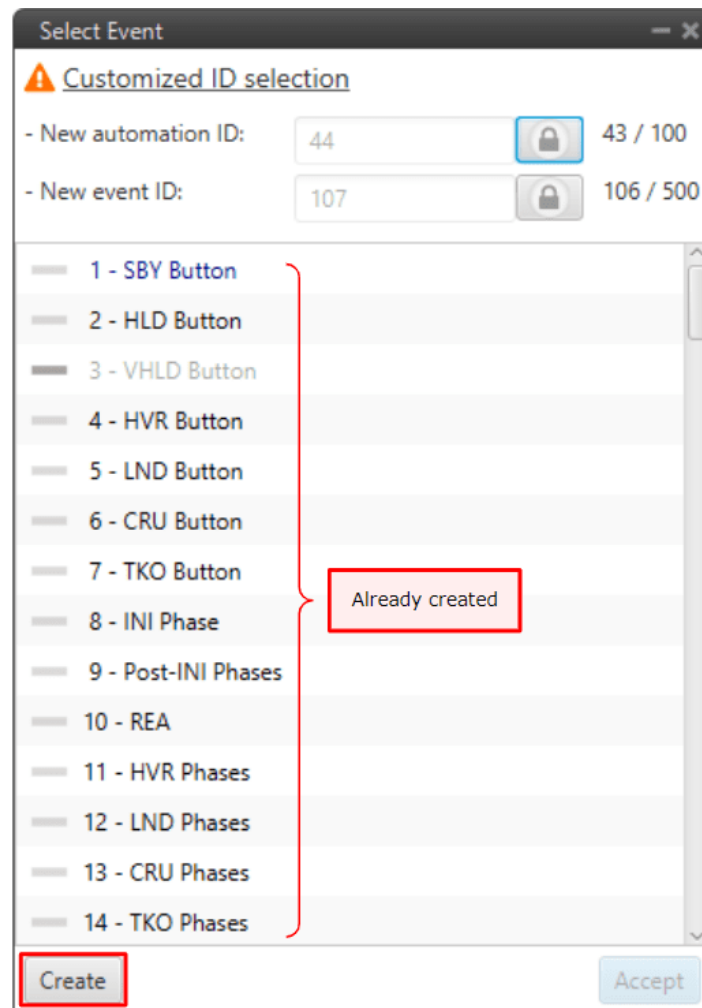


Fig. 142: New event

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

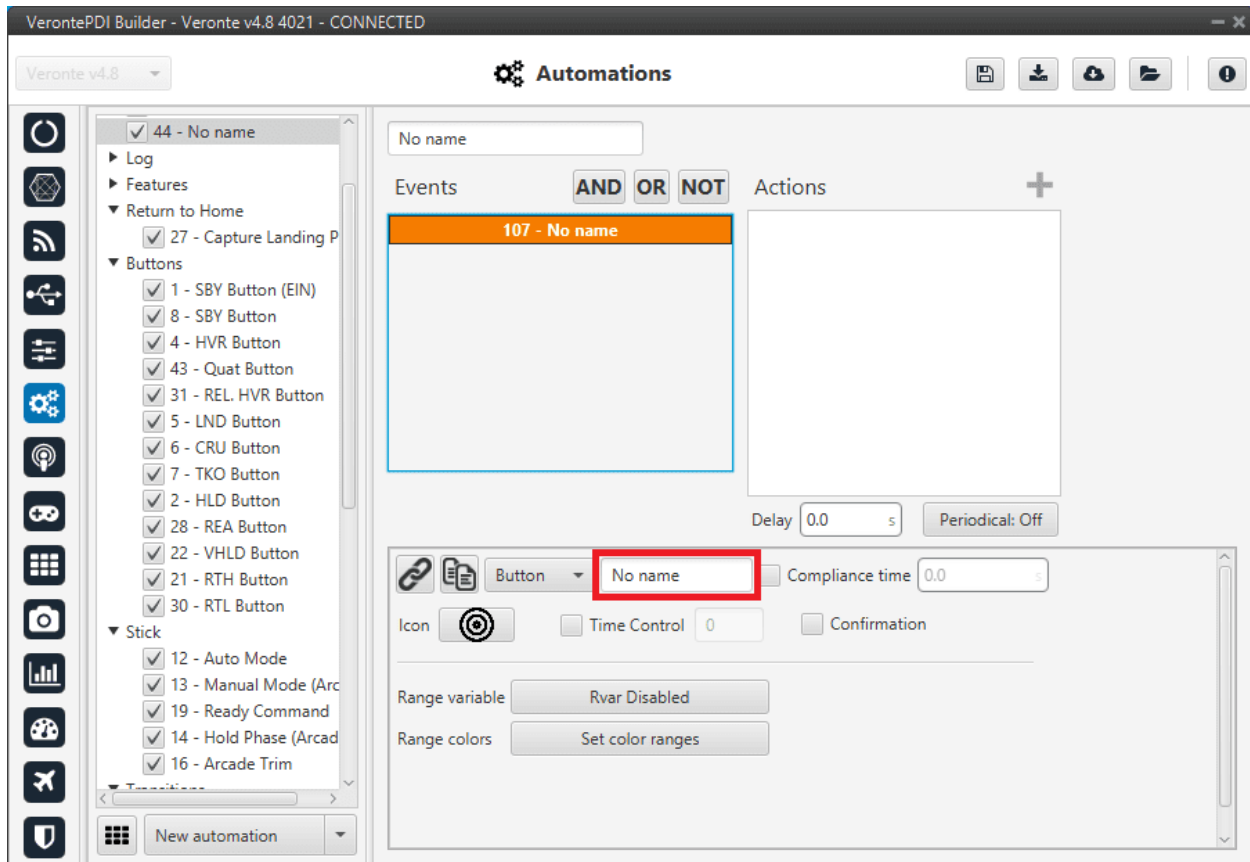


Fig. 143: New event - name

Below are present the different types of events that can be created.

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.

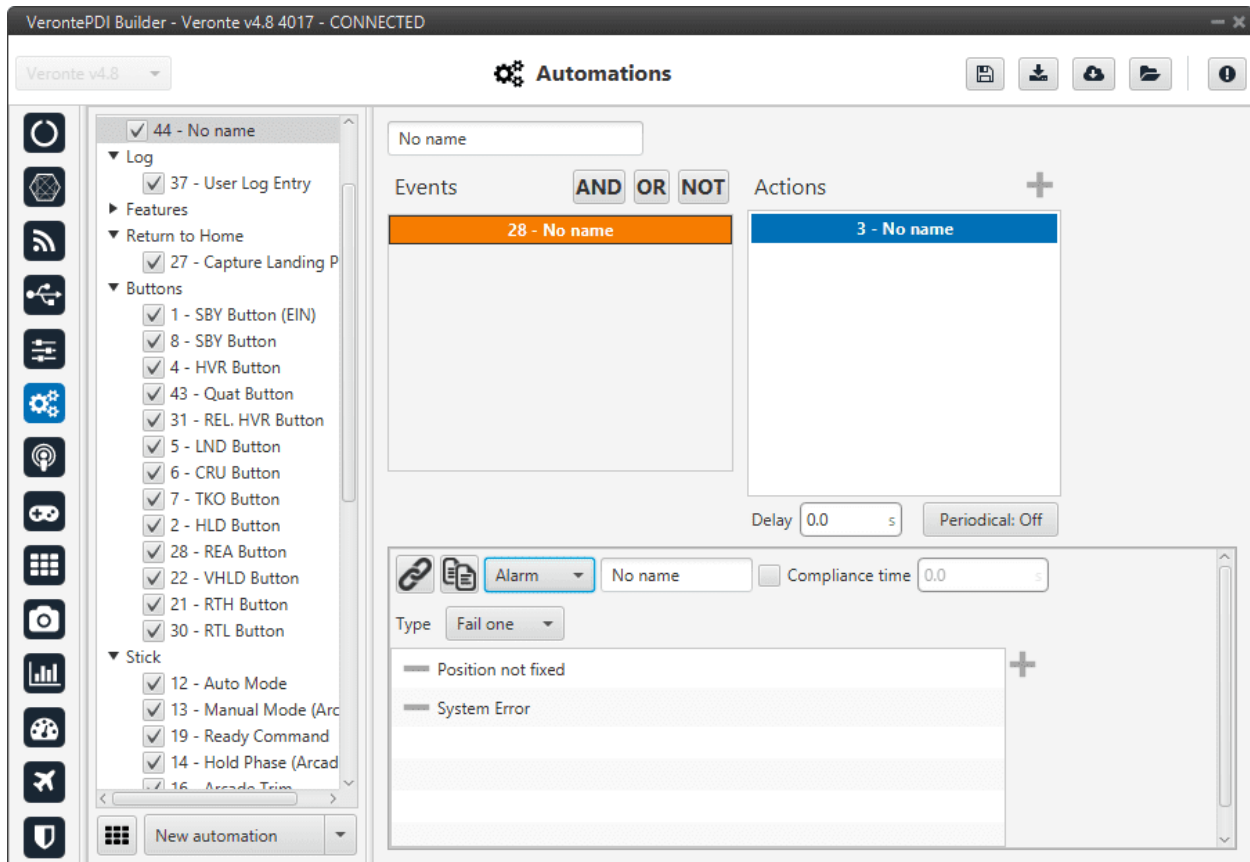


Fig. 144: Alarm event

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.

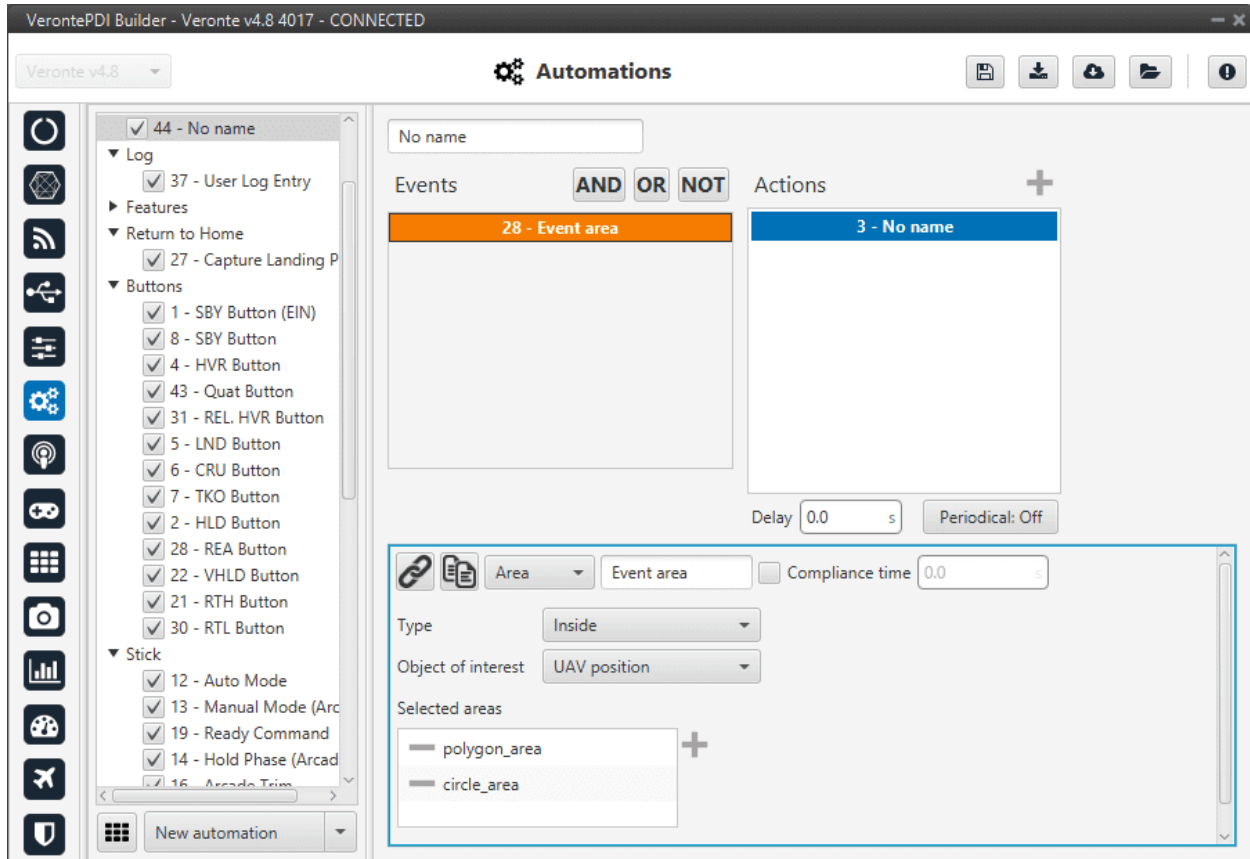


Fig. 145: Area event

- **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 (**polygons** or **circles**) in the *Operation elements* section 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 that will trigger the event when it is clicked.

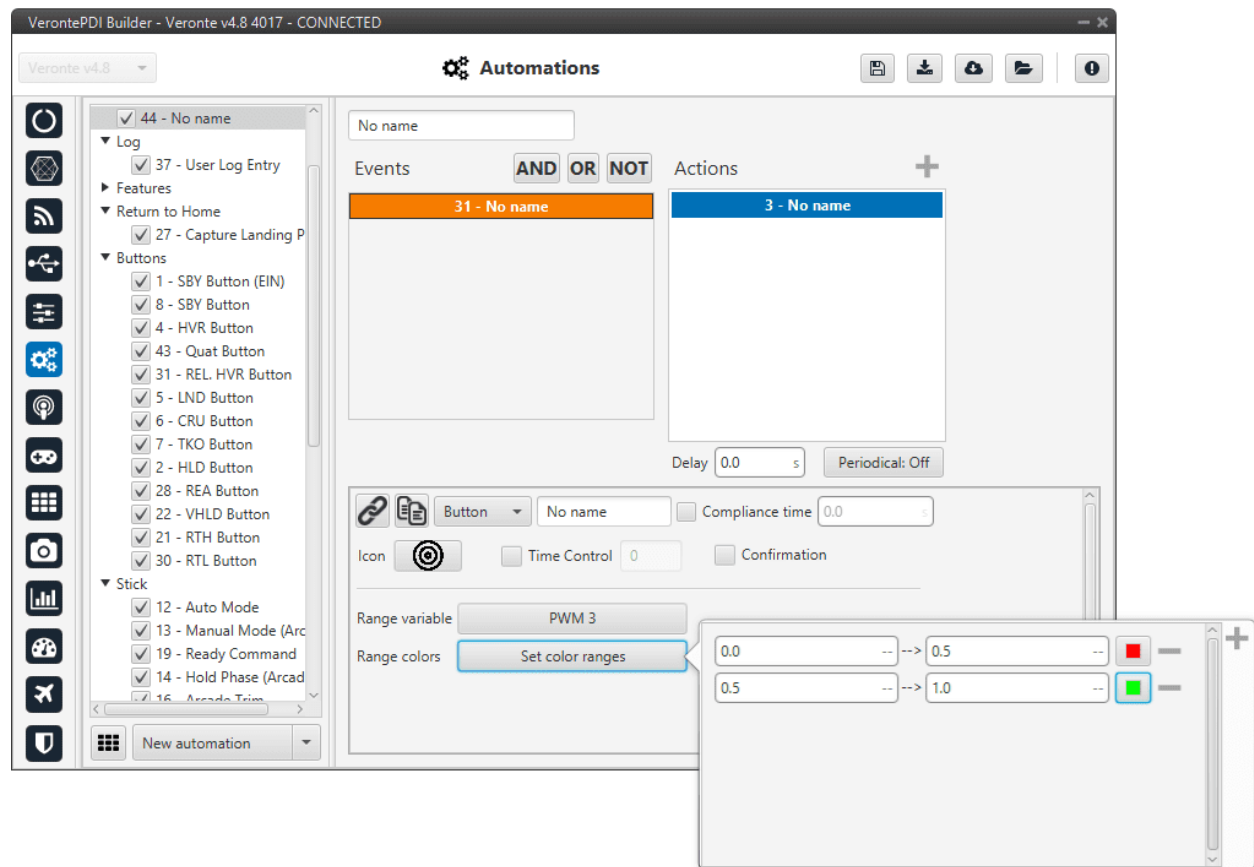


Fig. 146: Button event

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 option is used to trigger the action when the button is being pushed during the time specified in this option.
- **Confirmation:** A pop-up window asking for confirmation will be display 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 the one of the **Veronte Panel** with the name of that phase on it.
 - To create the button for changing to a determined phase, it is only needed to create the button with the same name as the phase.
- If the button event is linked to a different action (servo movement, variable, etc.), it will appear at the top of the **Veronte Panel**, as an **'action button'**, with the icon selected by the user.

2.6.2.1.4 Mode

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

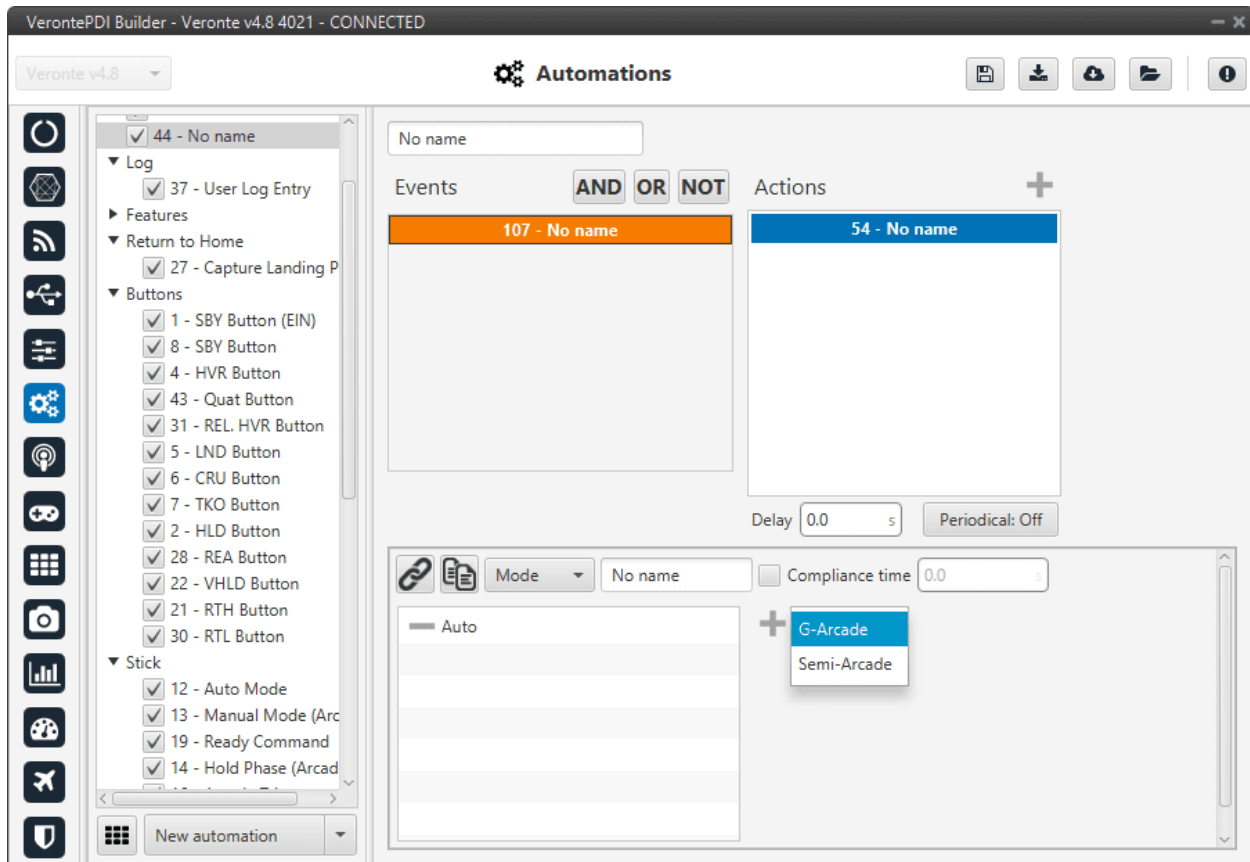


Fig. 147: Mode event

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 the “+” button, being in any of them will trigger the action.

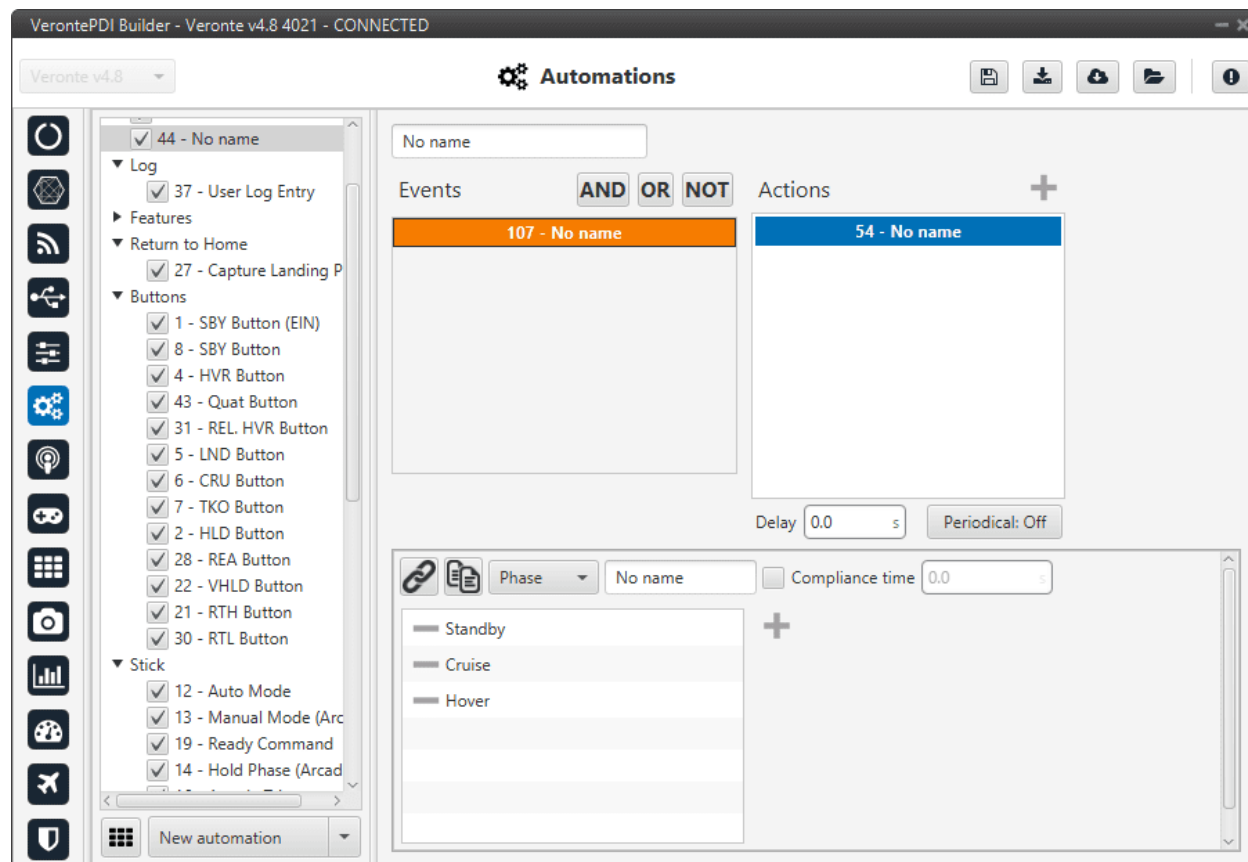


Fig. 148: Phase event

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 section 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](#)).

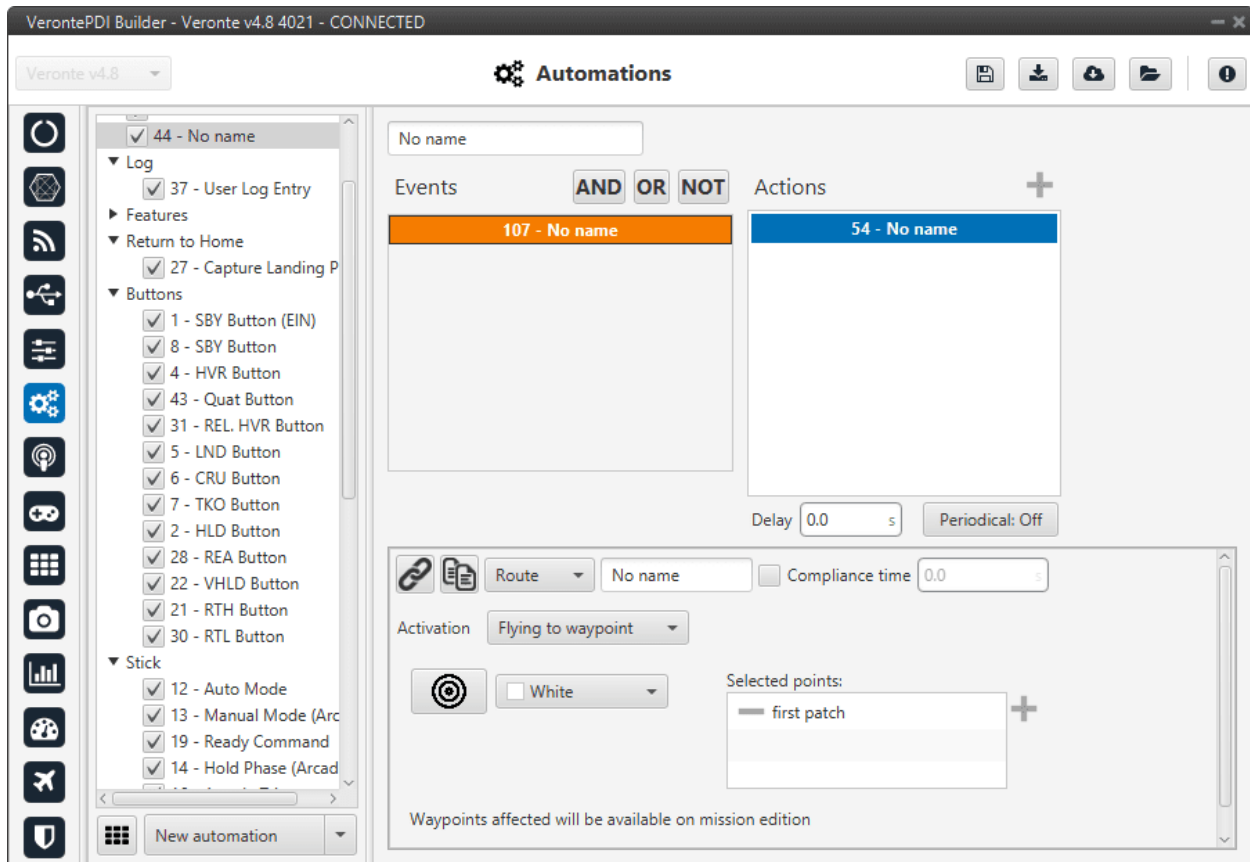


Fig. 149: Route event

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 section 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 configured previously on the action side of another automation (action type *Periodical*).

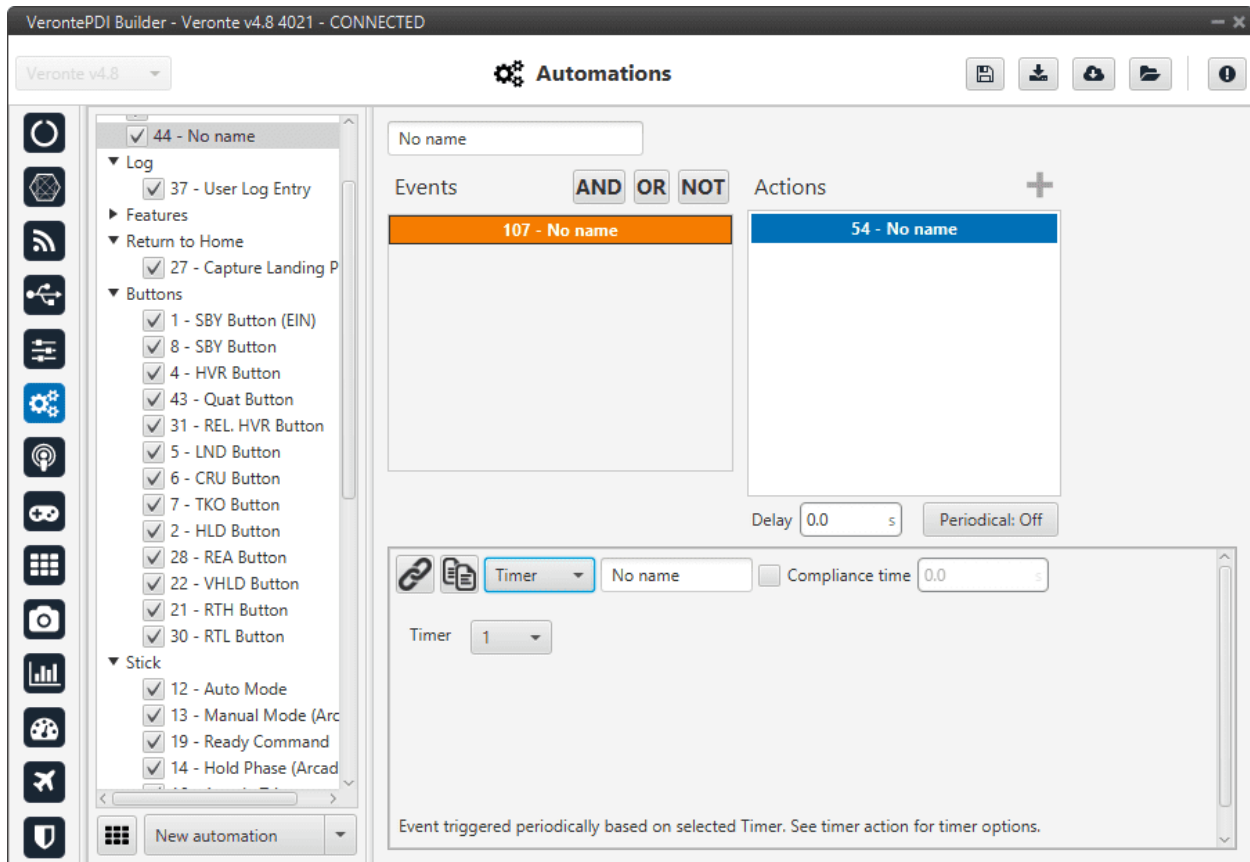


Fig. 150: Timer event

In **Timer** is selected 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.

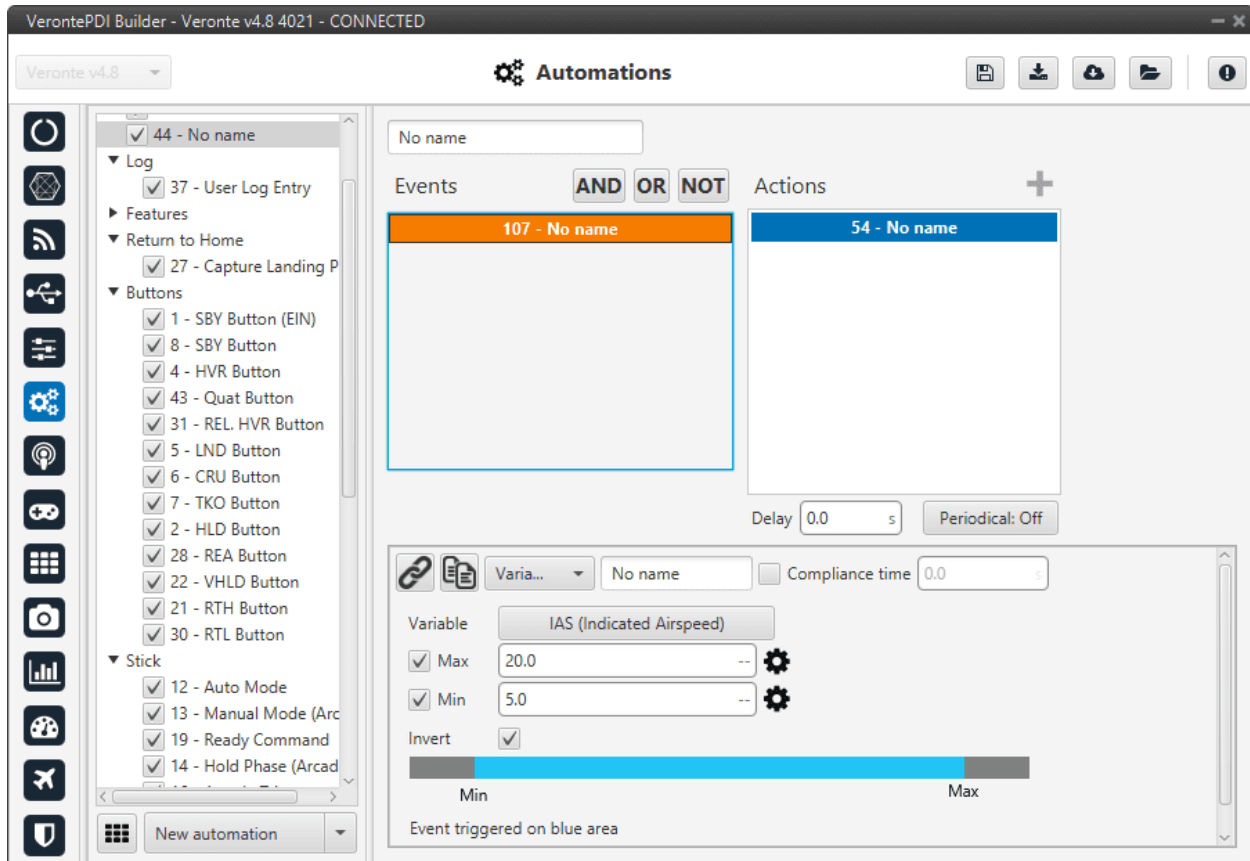



Fig. 151: Variable event

- **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 than 5 m/s or greater than 20 m/s.

2.6.2.2 Actions

An action is something that will be performed when the event (or group of events) has been accomplished. The actions box contains all the actions created.

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

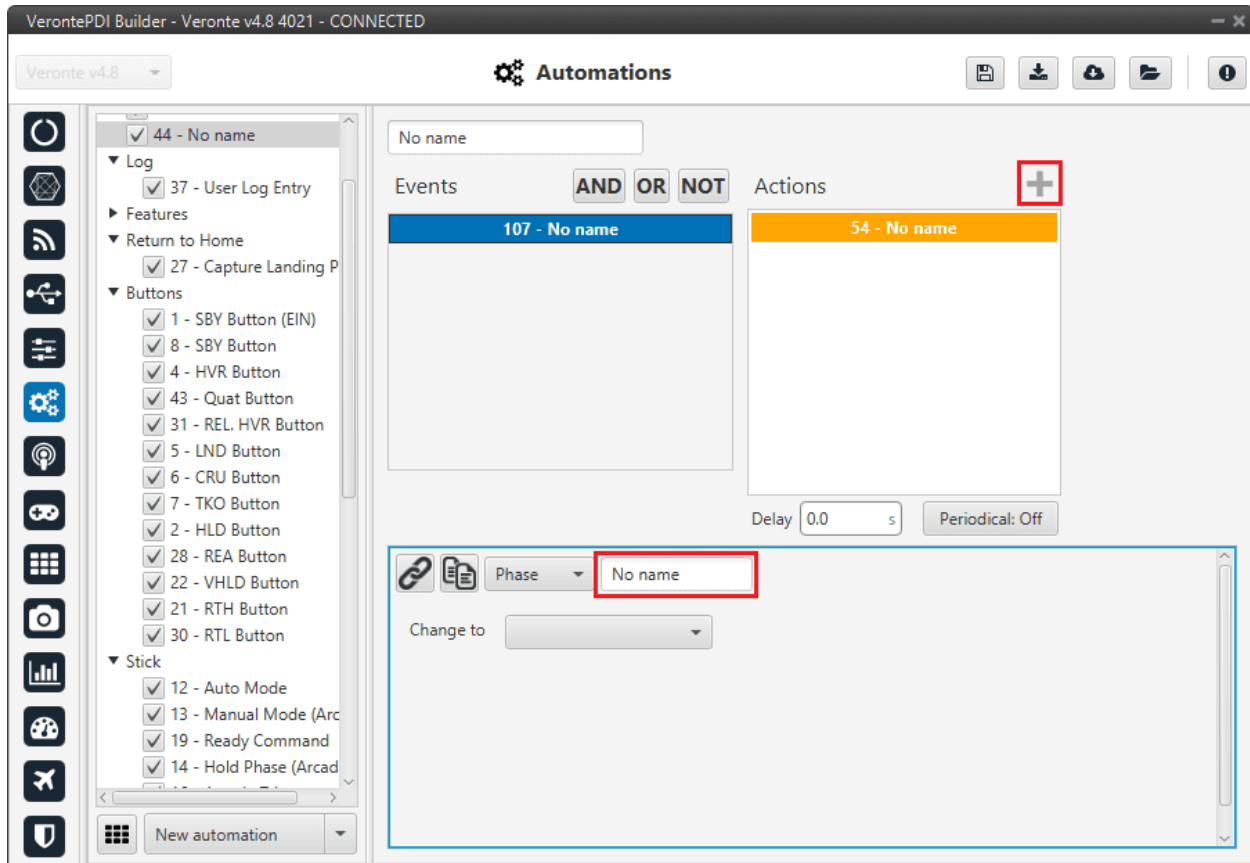


Fig. 152: Actions menu

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.

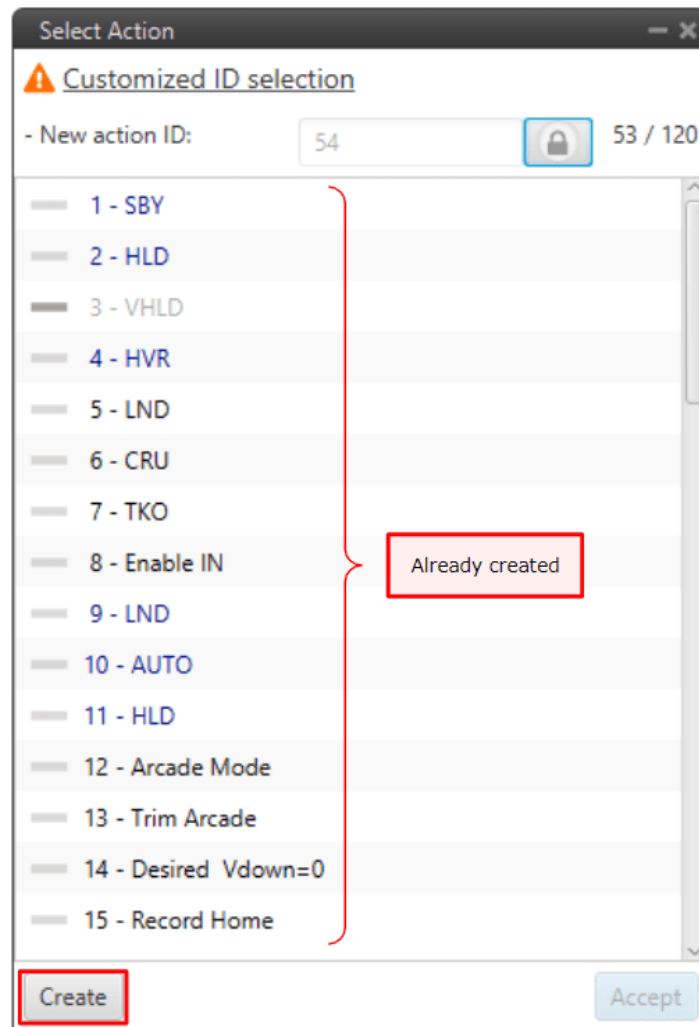


Fig. 153: New action

Below are present the different types of actions that can be created.

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 click [Veronte Ops manual](#)).

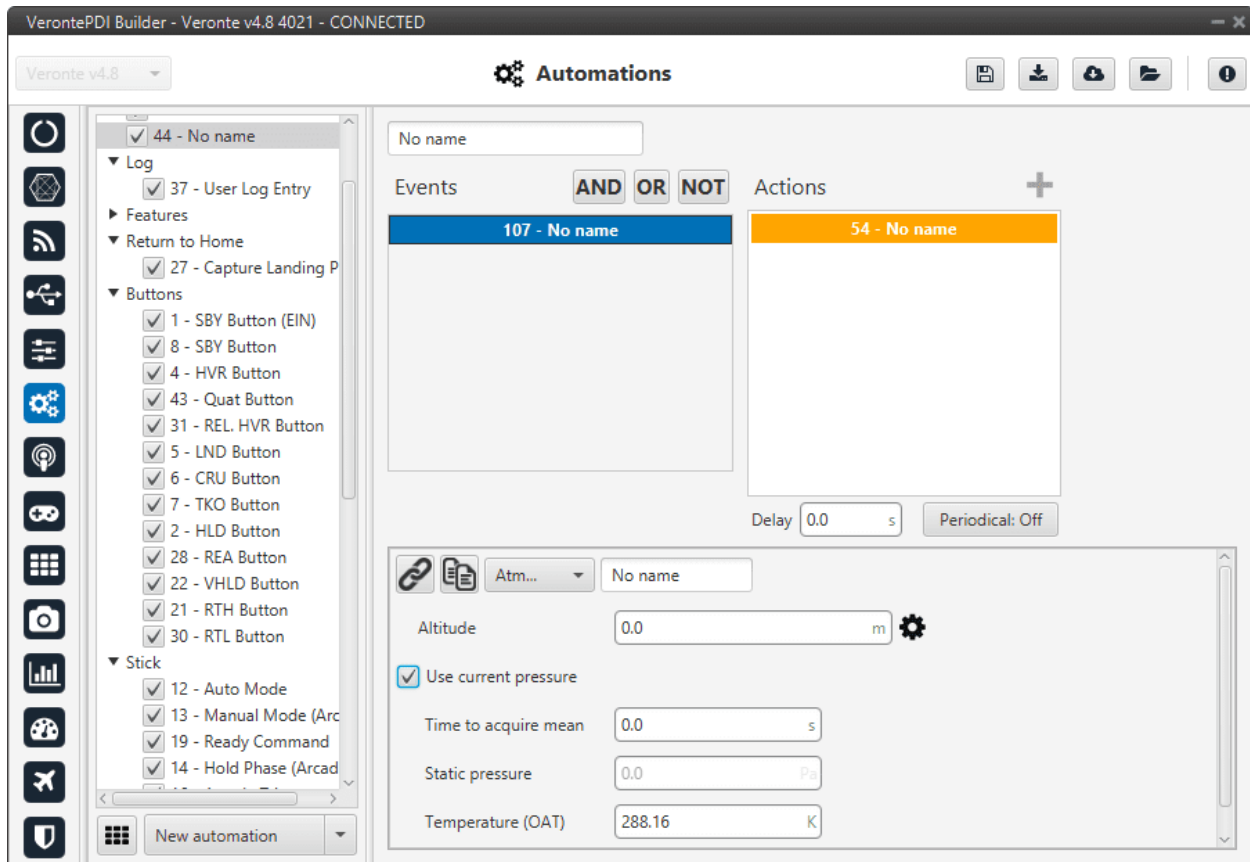


Fig. 154: Atmosphere Calibration action

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 the change of the actual sensor used as accelerometer, gyroscope and dynamic pressure.

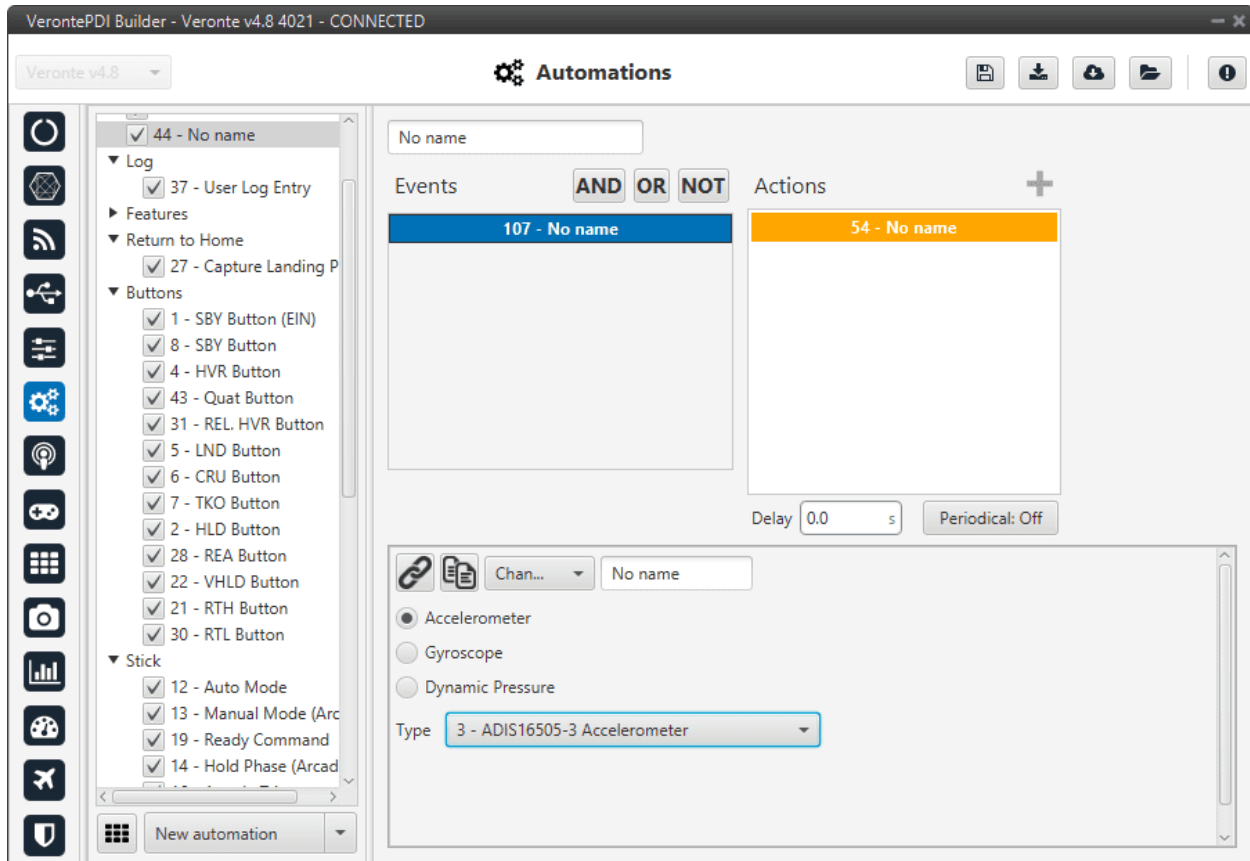


Fig. 155: Change active sensor action

2.6.2.2.3 Command block

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

Note: This action is disabled by default when 1x autopilot is started. To activate it, the user have to create a gimbal block or an arctrim block (see more information about it in [Block Programs](#)).

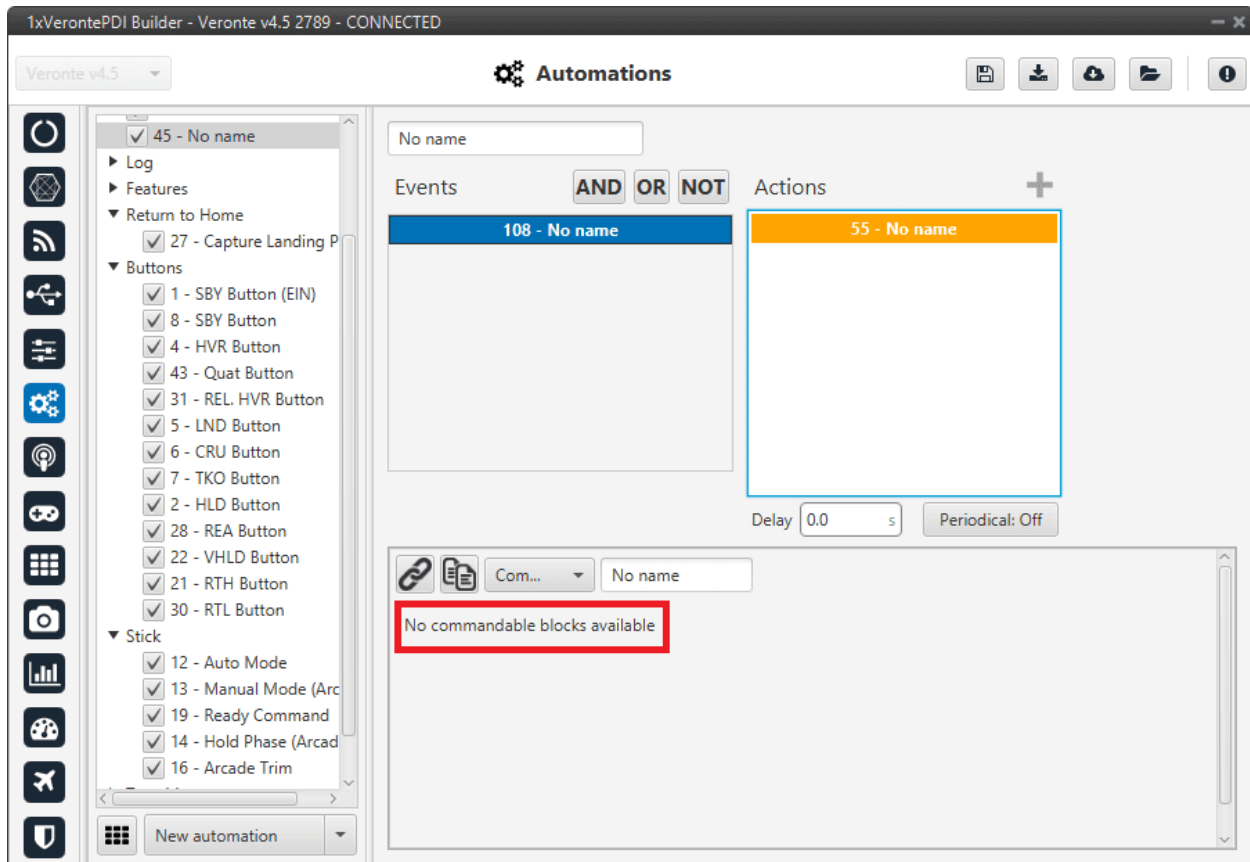


Fig. 156: Command block action

Gimbal

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

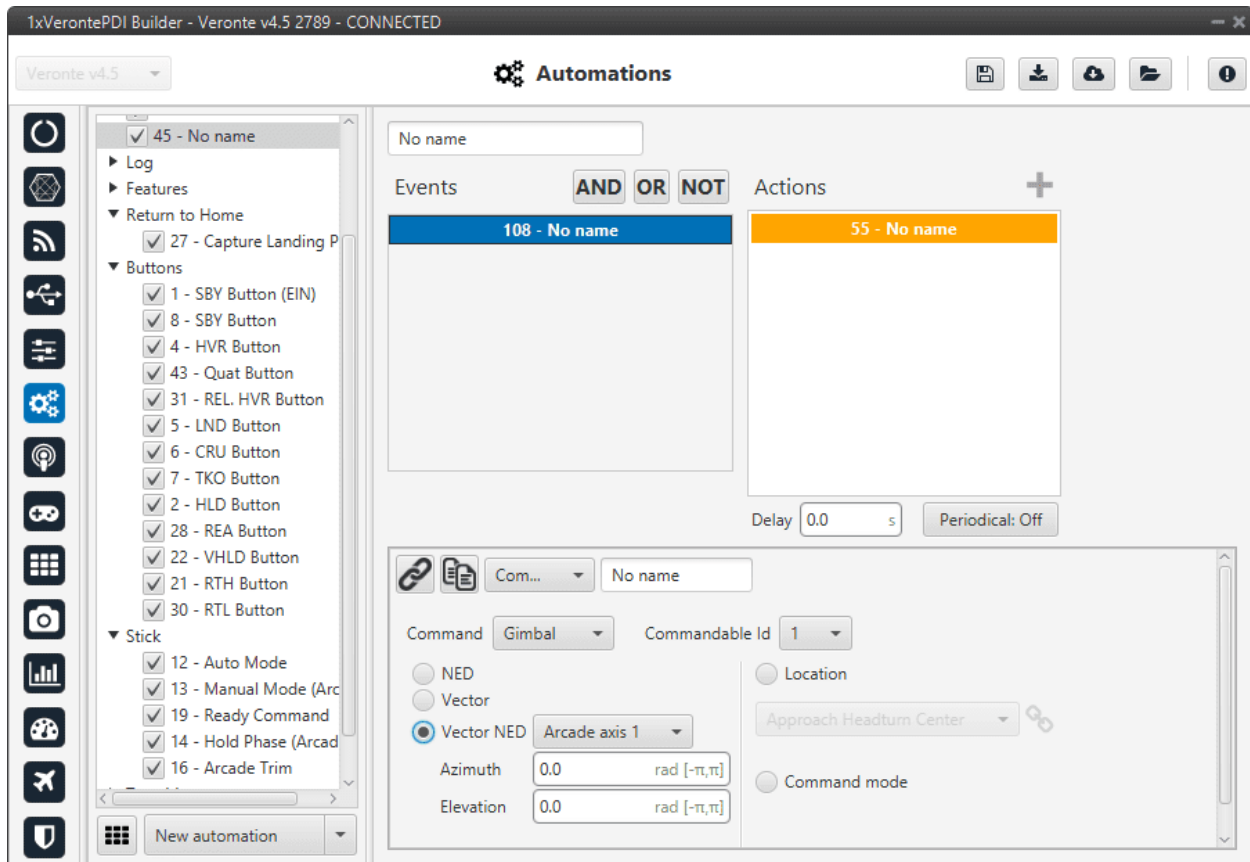


Fig. 157: Command block action - Gimbal

- **Commandable Id:** Id of the commandable Gimbal block (users can look up this Id in the block to be commanded, in *Block Programs* section).
- **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*.
- **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.

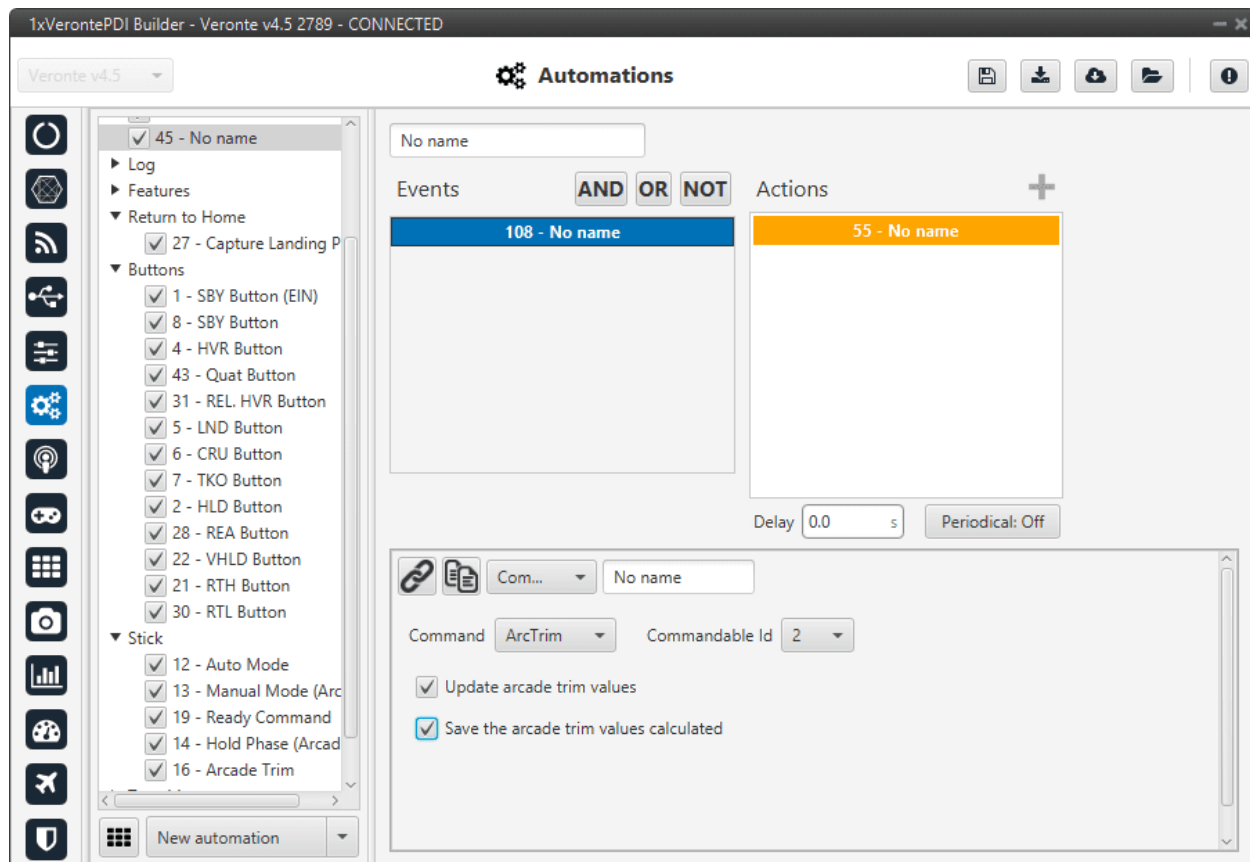


Fig. 158: Command block action - ArcTrim

- **Commandable Id:** Id of the commandable ArcTrim block (users can look up this Id in the block to be commanded, in *Block Programs* section).
- **Update the arcade trim values:** If this option is enabled, the stick is trimmed but not saved in the configuration. This means that **if 1x autopilot 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* section 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**.

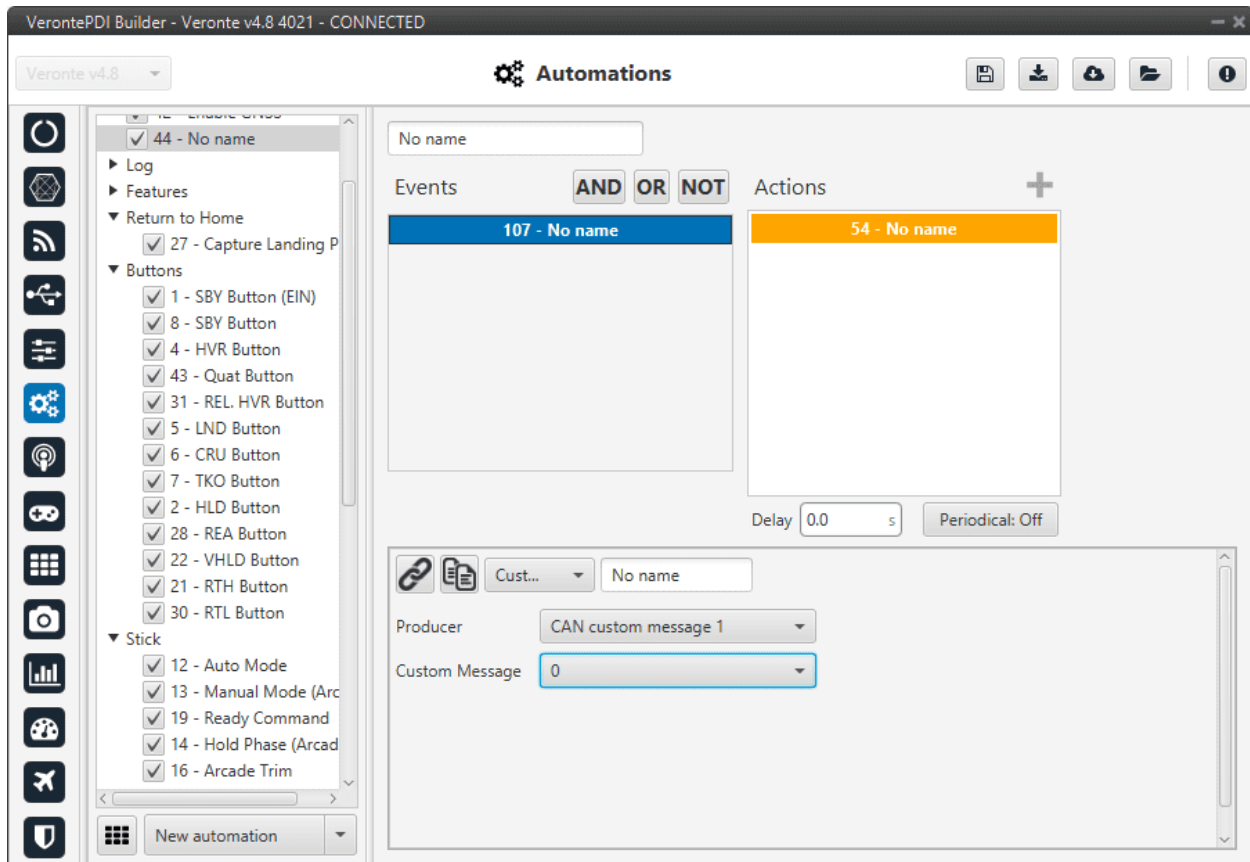


Fig. 159: Custom CAN TX action

The two parameters to configure in this action are:

- **Producer:** The user has to specify where the custom message is located: CAN custom message 1, 2 or 3.
- **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* section 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**.

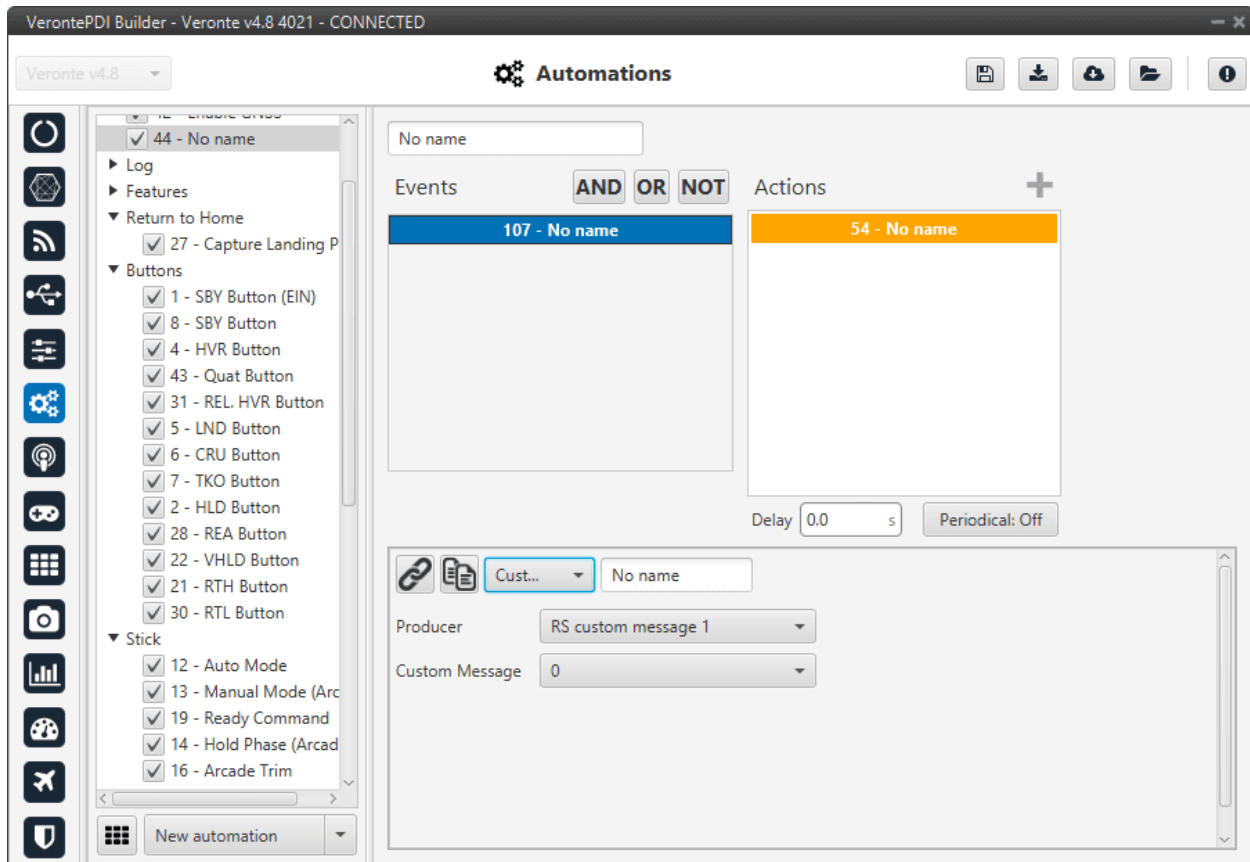


Fig. 160: Custom Serial TX action

The two parameters to configure in this action are

- **Producer:** The user has to specify where the custom message is located: RS custom message 1, 2 or 3.
- **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](#)).

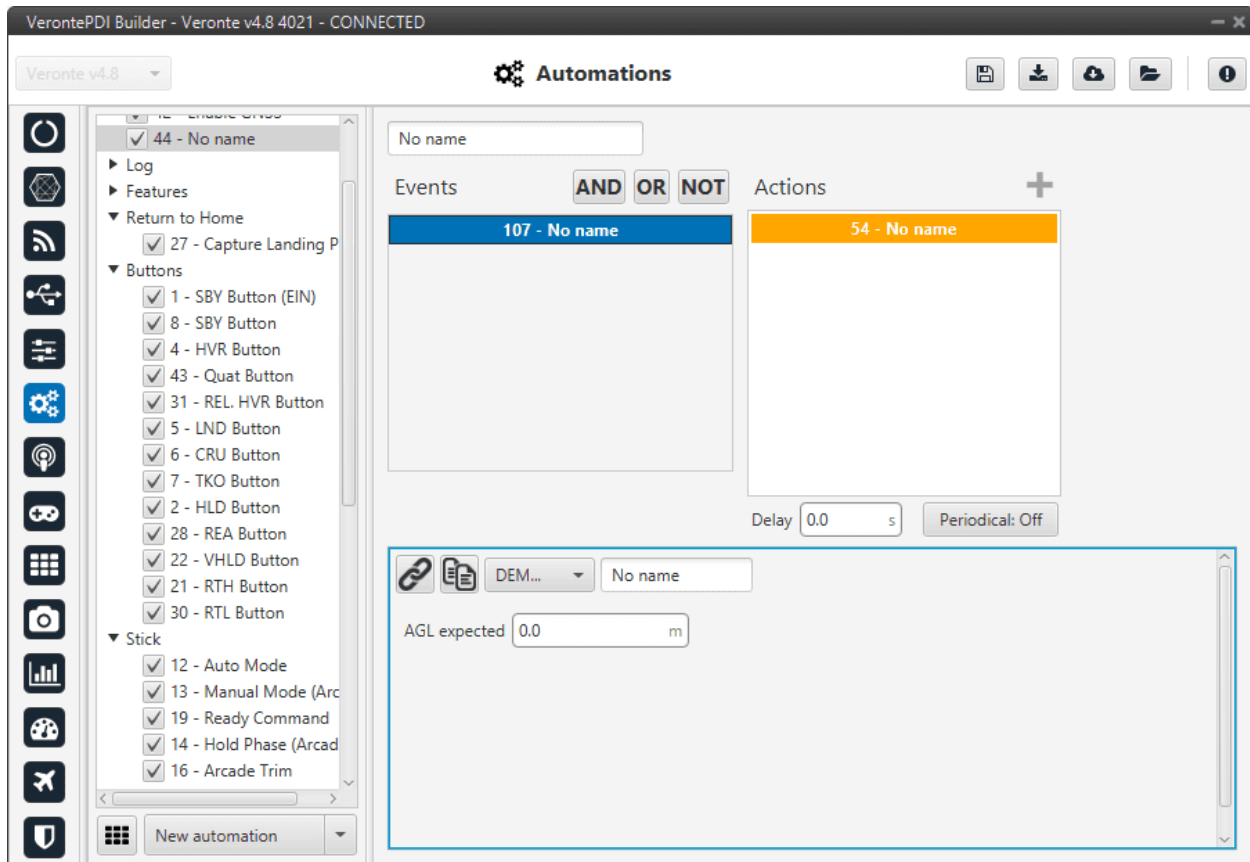


Fig. 161: DEM calibration action

2.6.2.2.7 Enable/Disable Wind Estimation

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

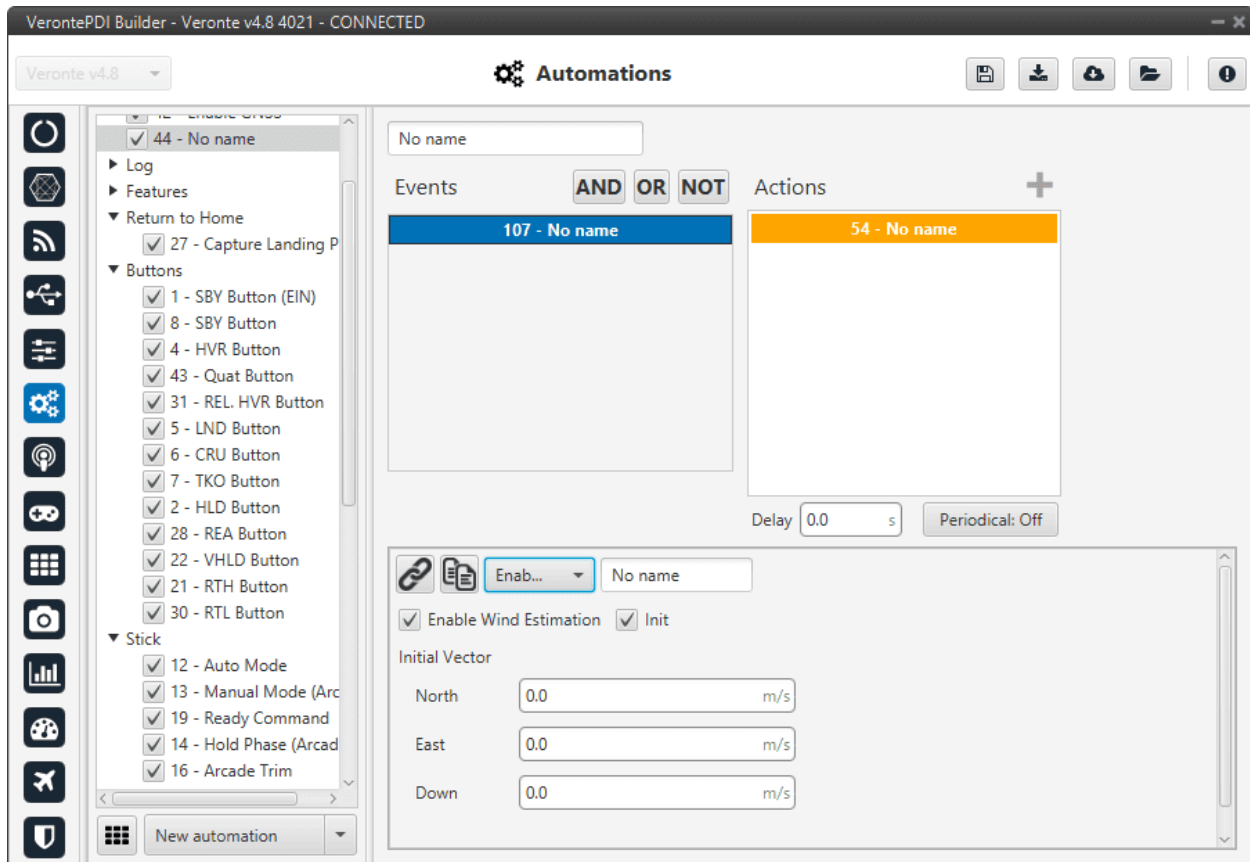


Fig. 162: Enable/Disable Wind Estimation action

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:** Initial wind vector.

2.6.2.2.8 Envelope

This action is used to change the envelope during the flight mission. Envelopes are created in the Control Menu, see section [Envelope](#).

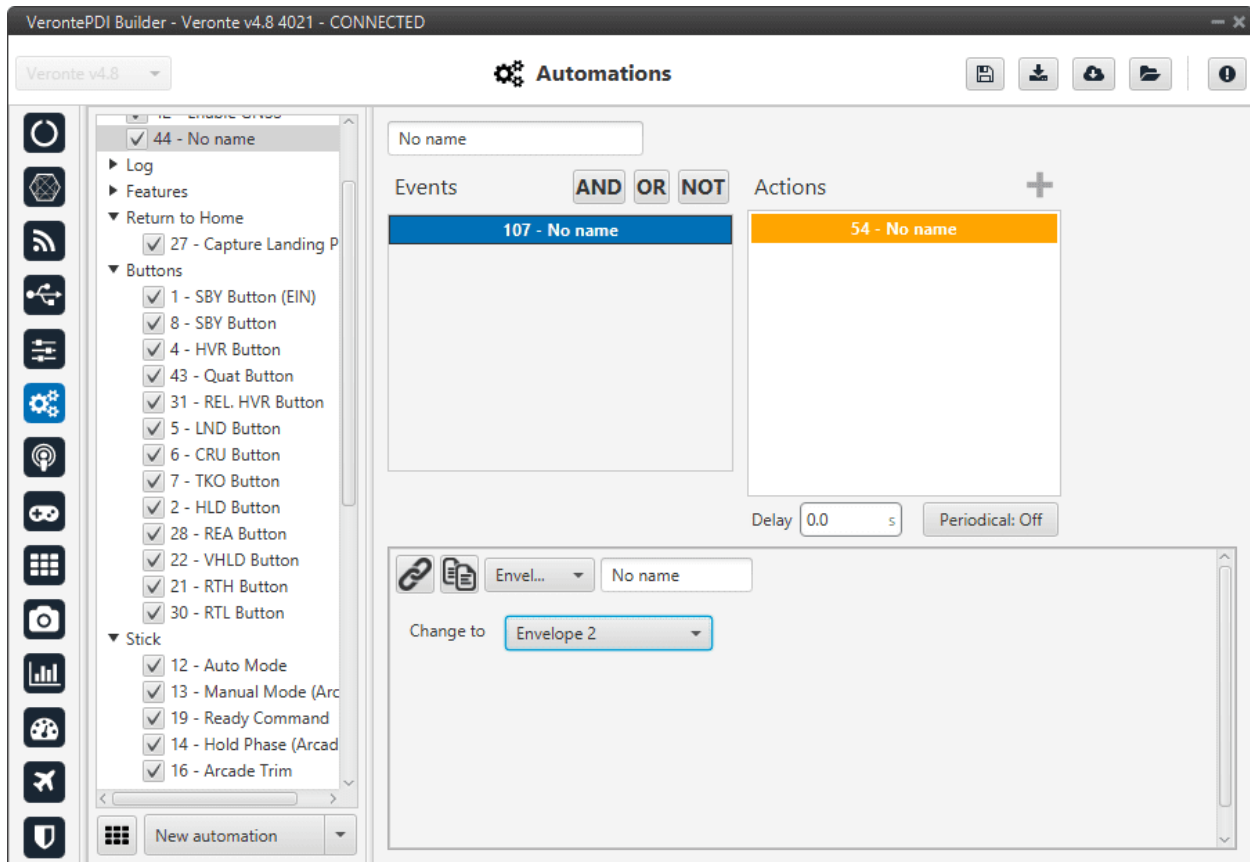


Fig. 163: Envelope action

This is useful with a hybrid platform, being possible to change the envelope when the aircraft changes its configuration. Envelopes are selected from those created previously.

2.6.2.2.9 FTS Activation

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

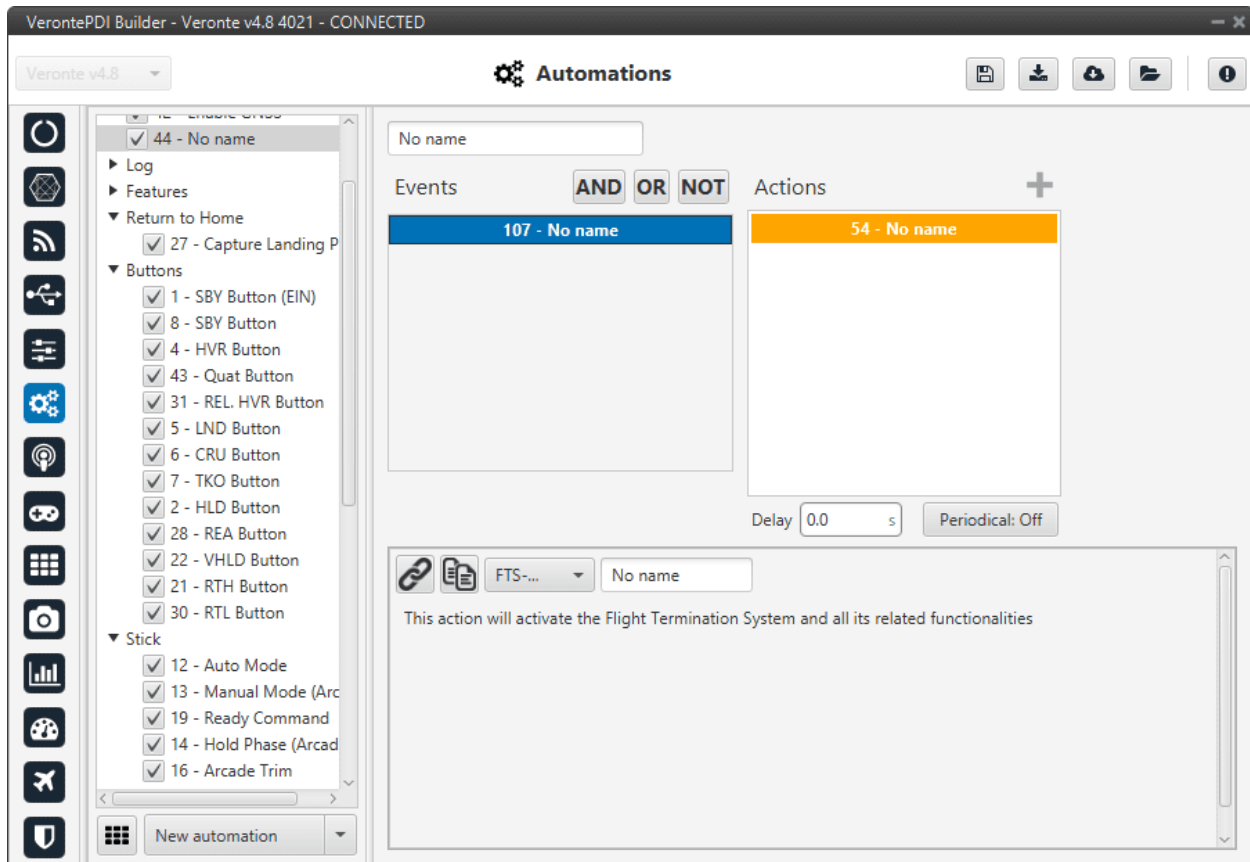


Fig. 164: FTS Activation action

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

2.6.2.2.10 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):

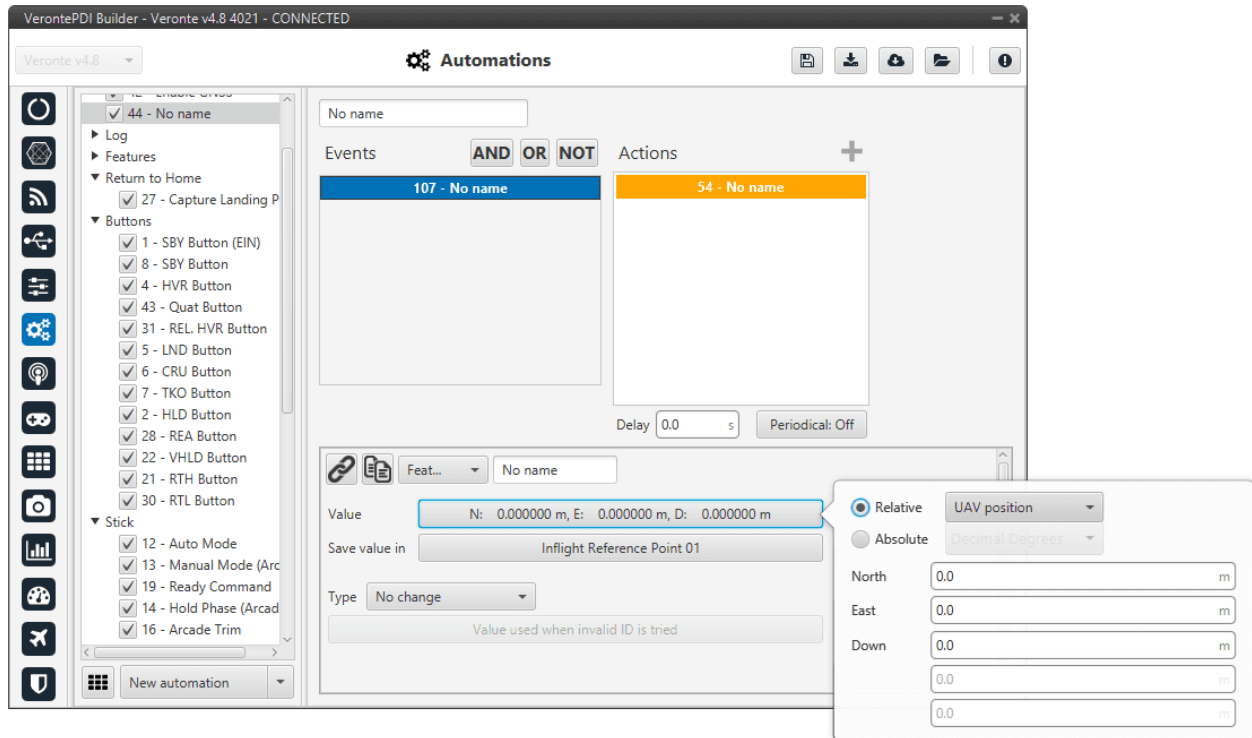


Fig. 165: Feature action

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 ° ' ' '. 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 a 1x autopilot.
- **Save value in:** The position has to be saved in an 'Inflight Reference Point'.
- **Types:** 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.

2.6.2.2.11 Format SD

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

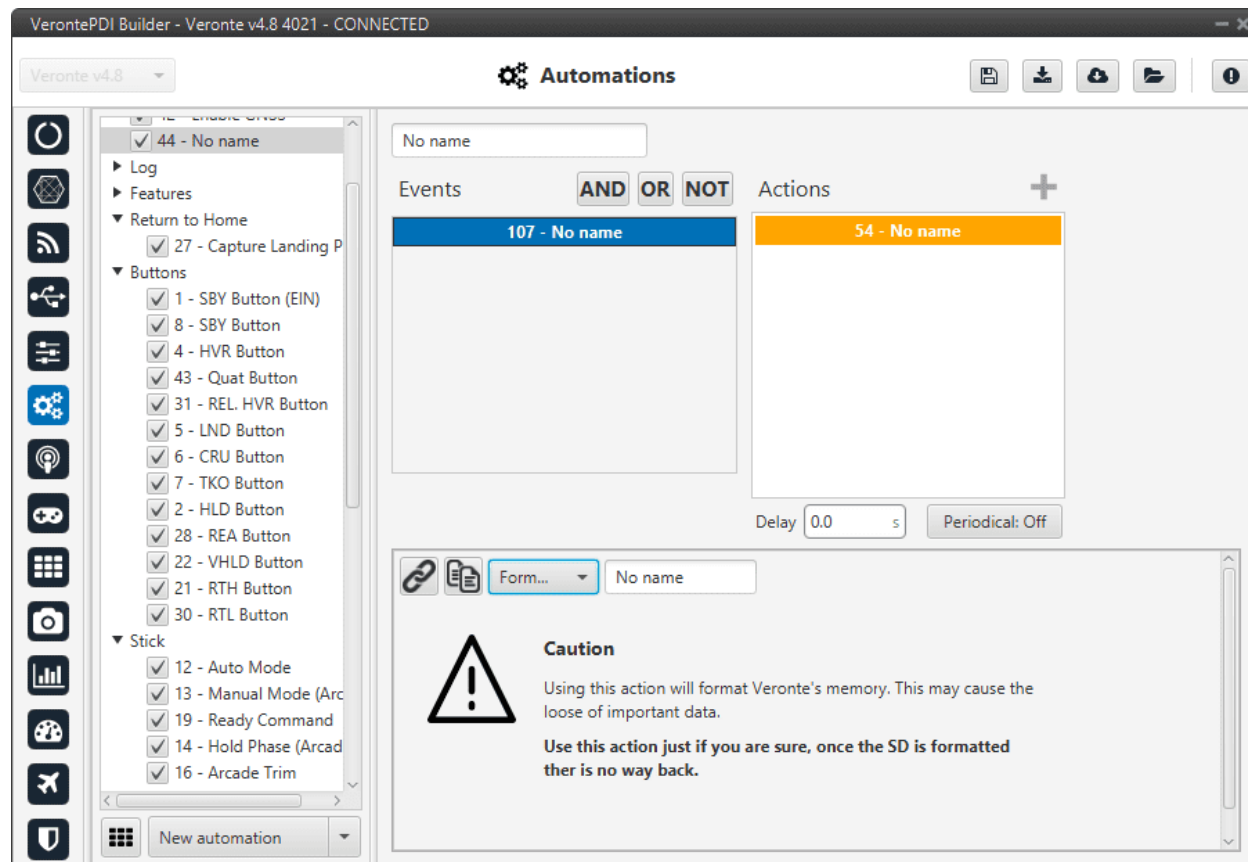


Fig. 166: Format SD action

2.6.2.2.12 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.

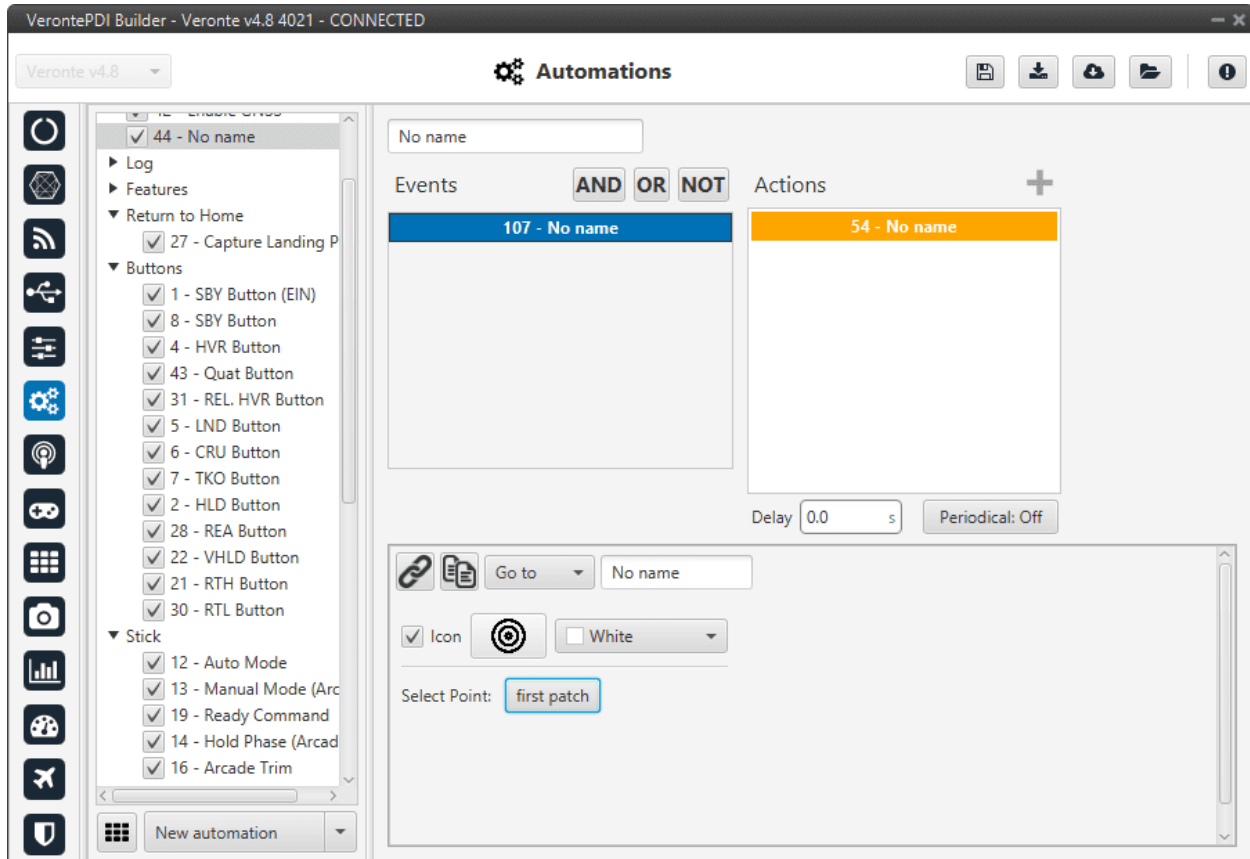


Fig. 167: Go to action

In this action two parameters can be configured:

- **Select Point:** To select point (**patch**), first define it in the *Operation elements section 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.

2.6.2.2.13 Mode

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

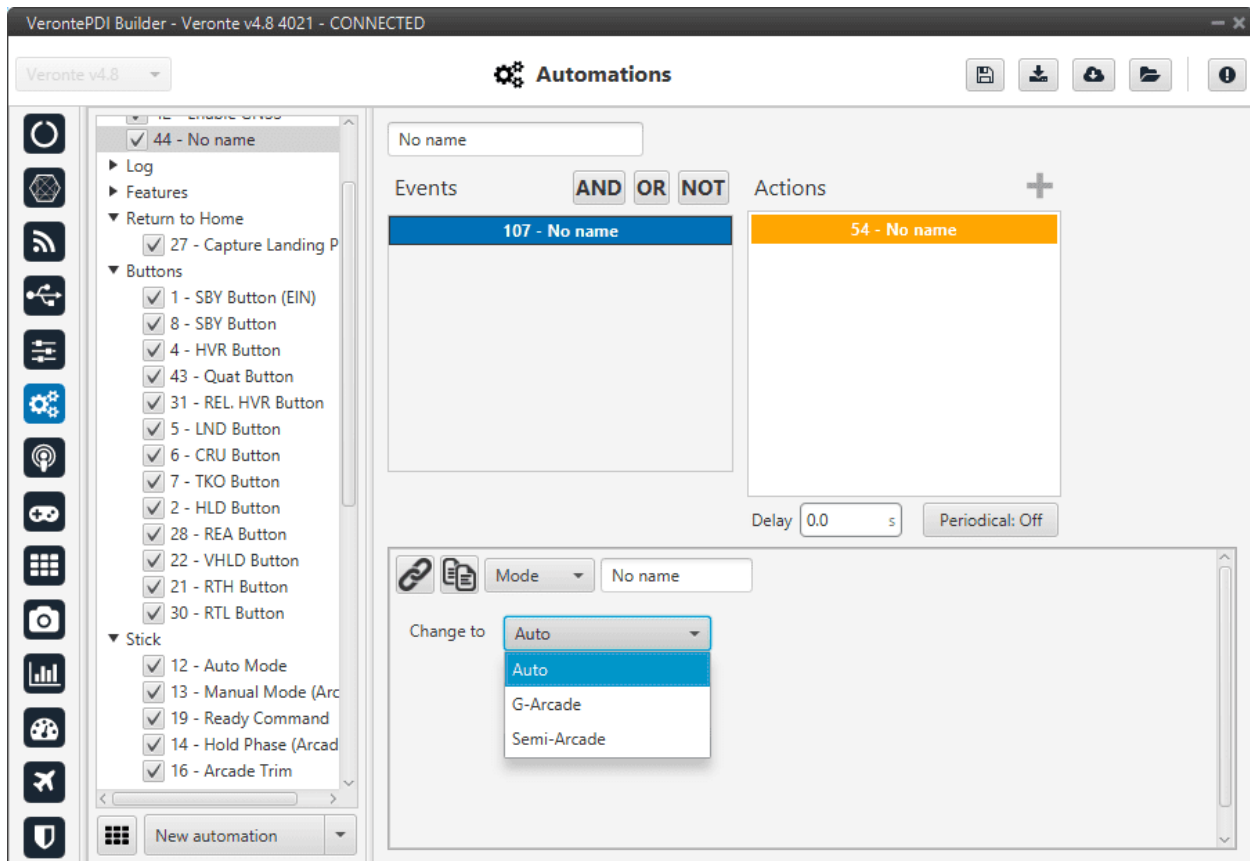


Fig. 168: Mode action

These modes have been created previously. See section [Modes](#), for more information about creating modes.

2.6.2.2.14 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 1x autopilot 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 **behaviour** 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.

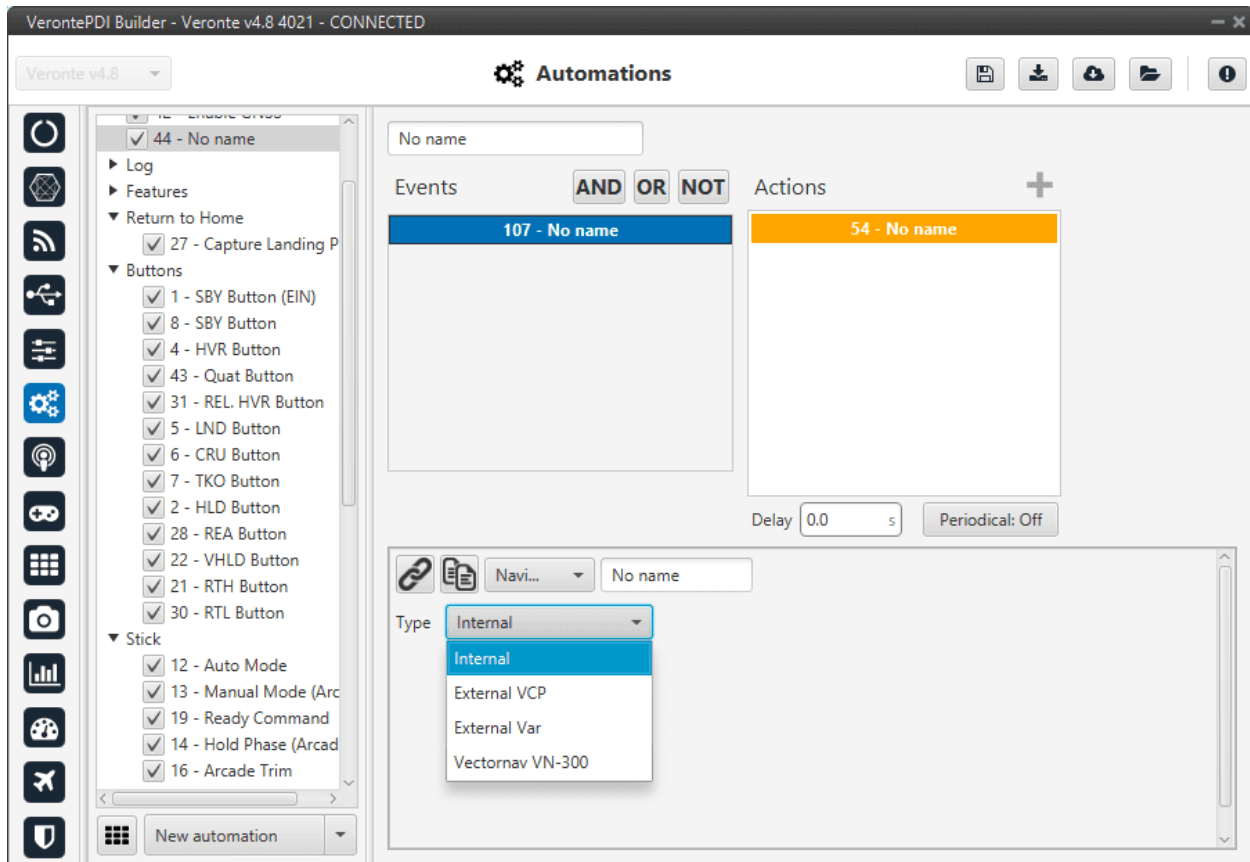


Fig. 169: Navigation action

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. Data (position, attitude, etc.) is provided by Var.
- **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.

2.6.2.2.15 Obstacle avoidance

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

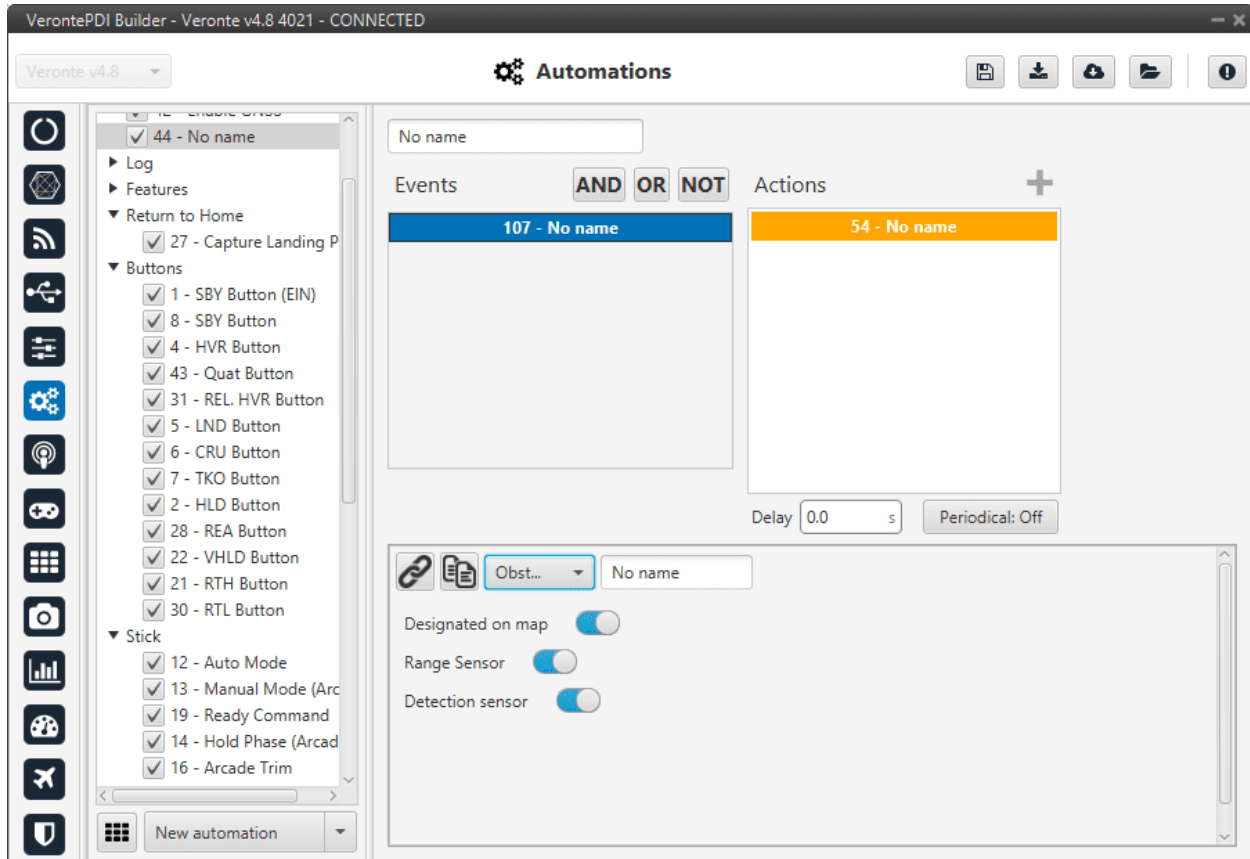


Fig. 170: Obstacle avoidance action

Three types of obstacles can be enabled:

- **Designated on map:** Those obstacles that appear on the map, obstacle zones and aircraft received by ADS-B.
- **Range sensor:** Obstacles previously defined in **Veronte Ops**.
- **Detection sensor:** Obstacles previously defined in **Veronte Ops**.

2.6.2.2.16 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 section [GPIO](#)).

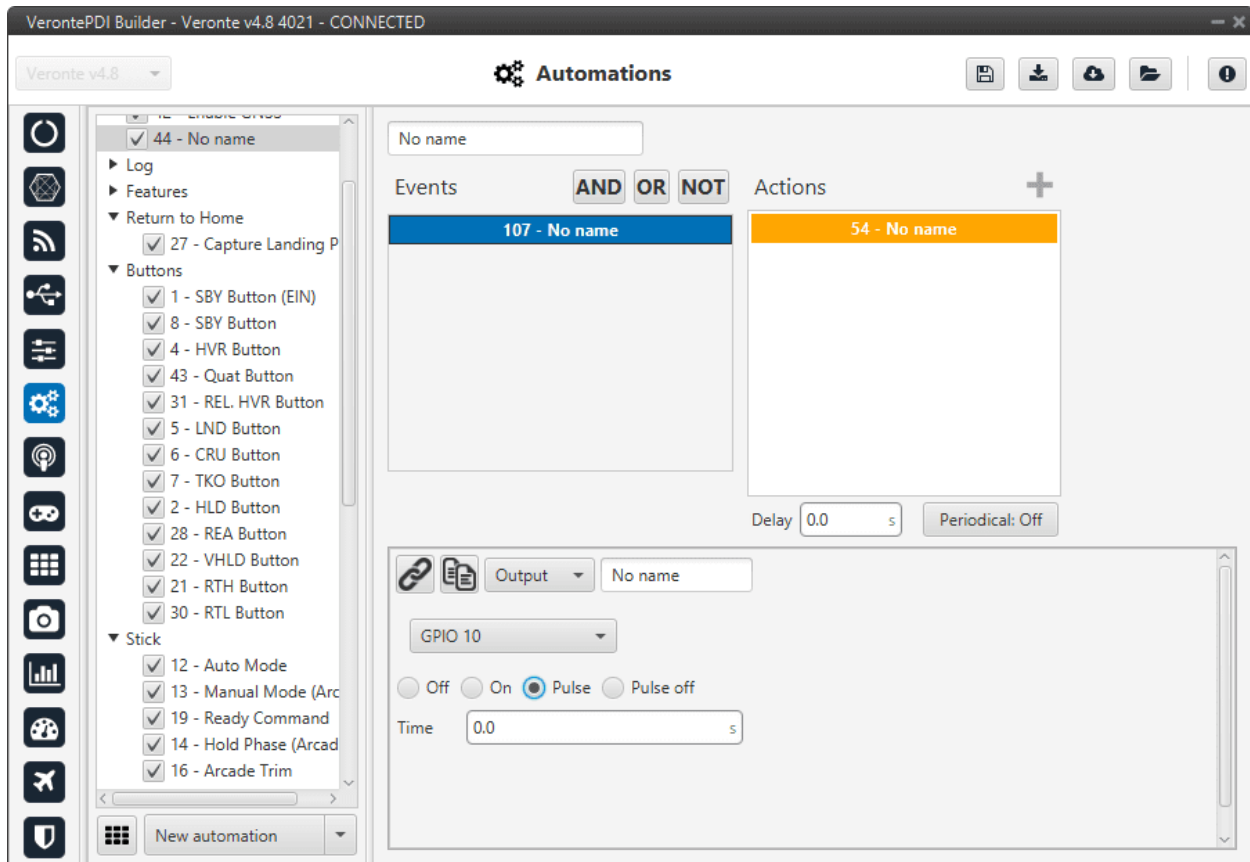


Fig. 171: Output action

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.

2.6.2.2.17 Periodical

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

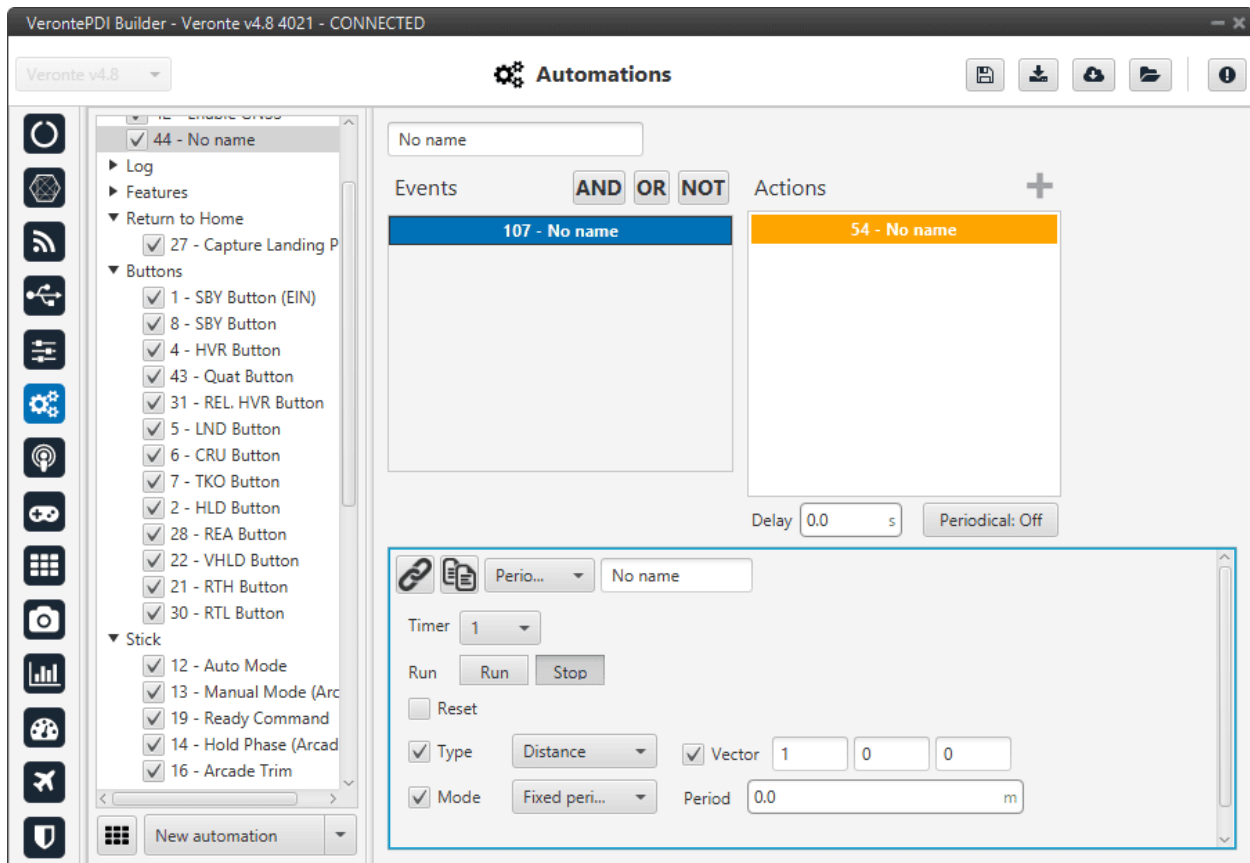


Fig. 172: Periodical action

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:** 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 + Resert:** The timer will be stopped and set back to zero.
- **Type:** These available options have been explained in [Automations](#).
- **Mode:** The difference between fixed delay and fixed period has been 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.

2.6.2.2.18 Phase

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

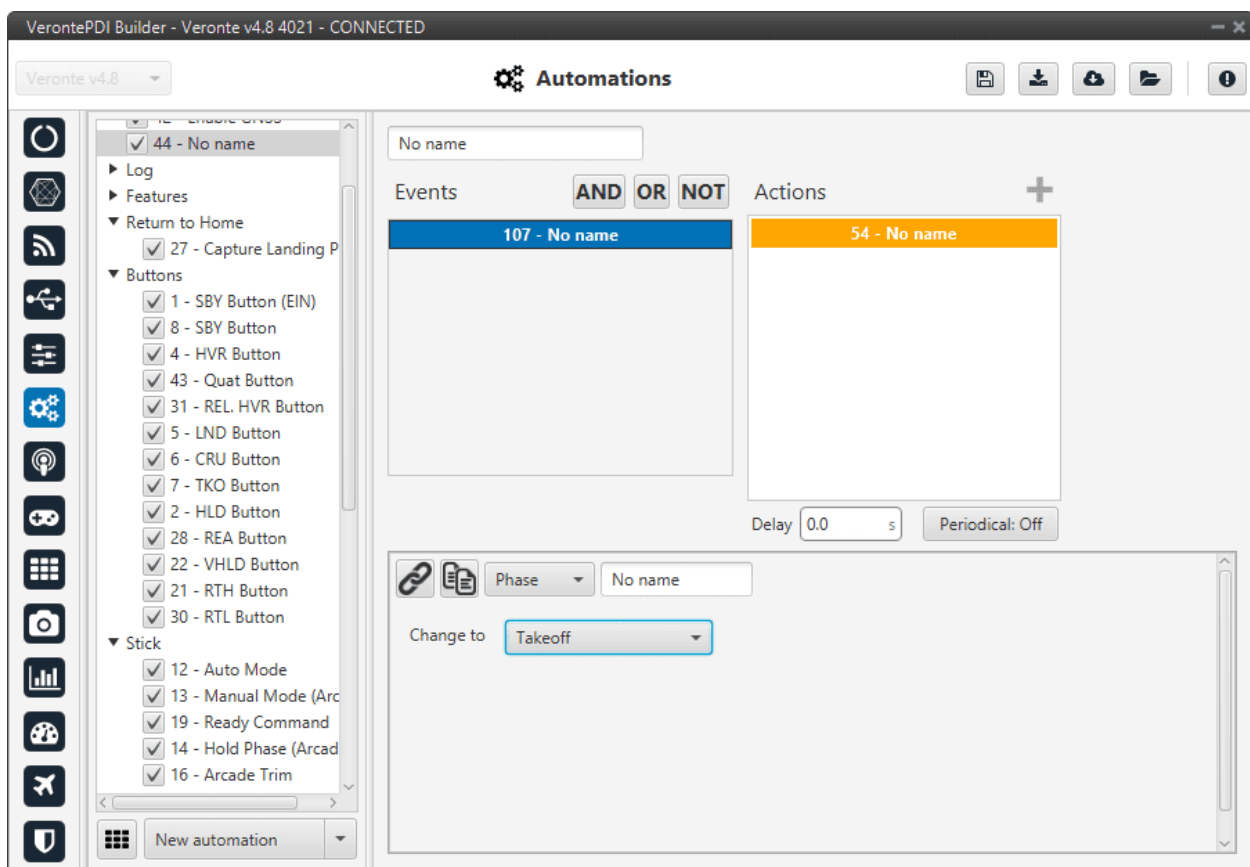


Fig. 173: Phase action

These phases have been created previously. See section [Phases](#), for more information about creating phases.

2.6.2.2.19 Ports

This action allows the user to switch between 4 pre-set configurations defined in the [Ports](#) section of the Communication menu.

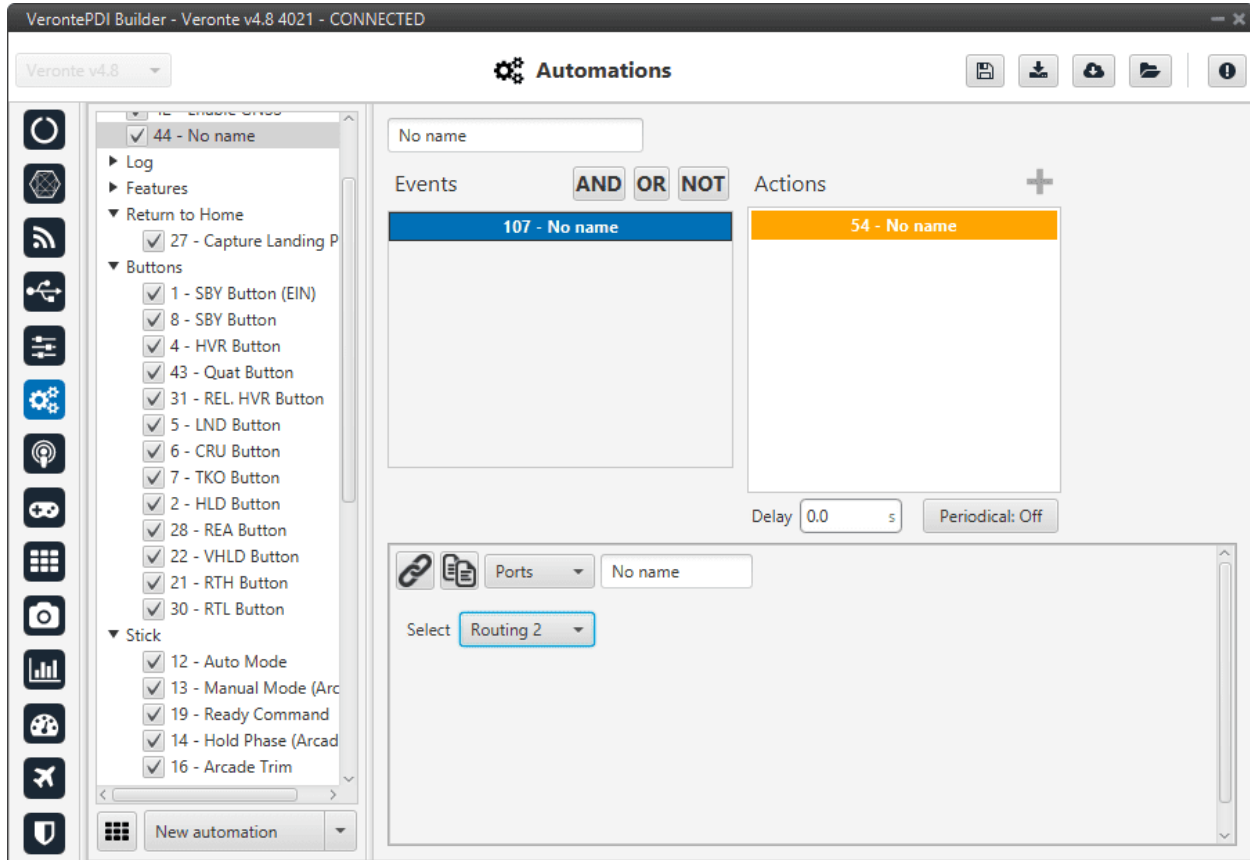


Fig. 174: Ports action

2.6.2.2.20 Run block program

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

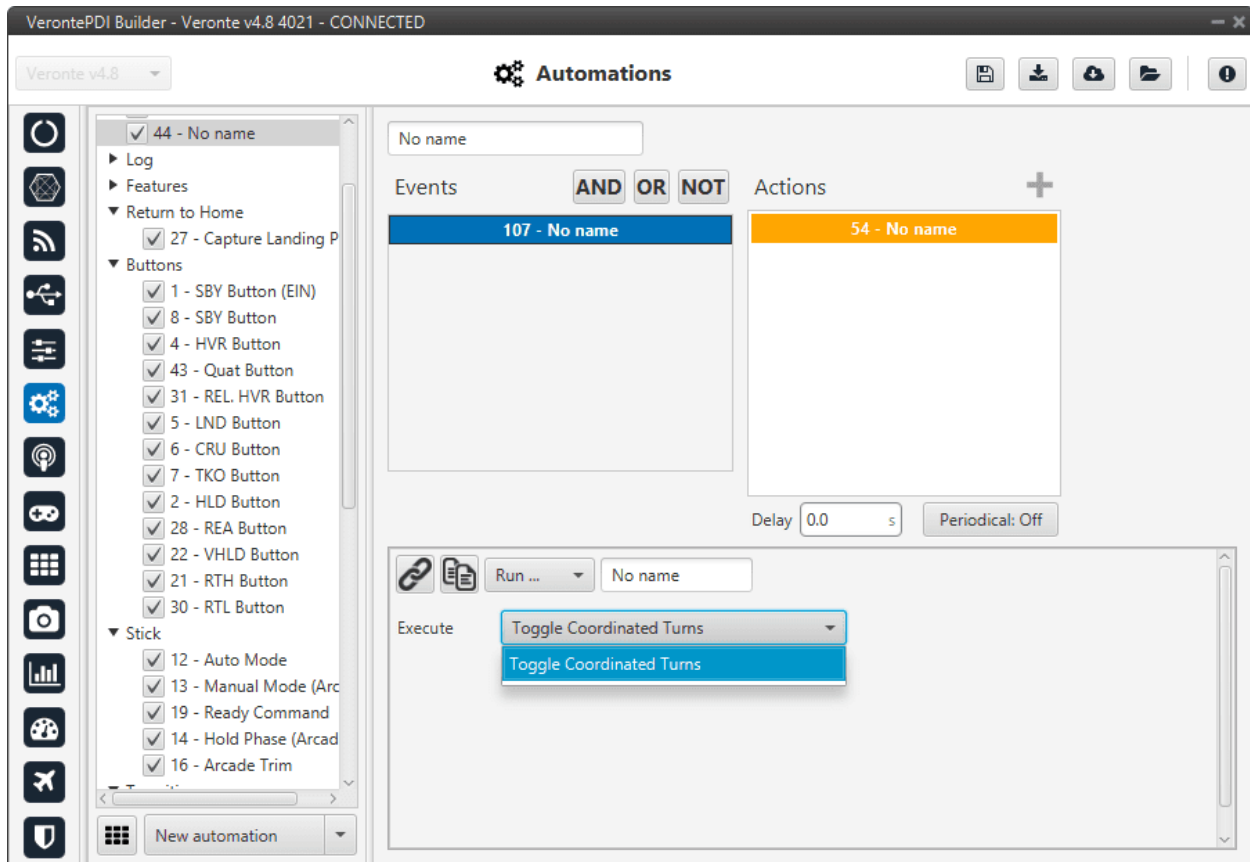


Fig. 175: Run block program action

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 continuously. For more information about programs, see section [Block Programs](#).

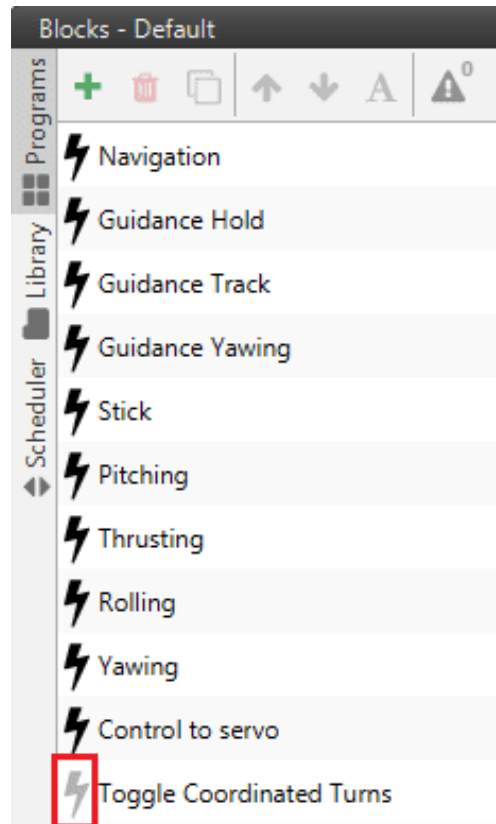


Fig. 176: Grey lightning icon

2.6.2.2.21 Safety Bits

This action selects a predefined safety bits list.

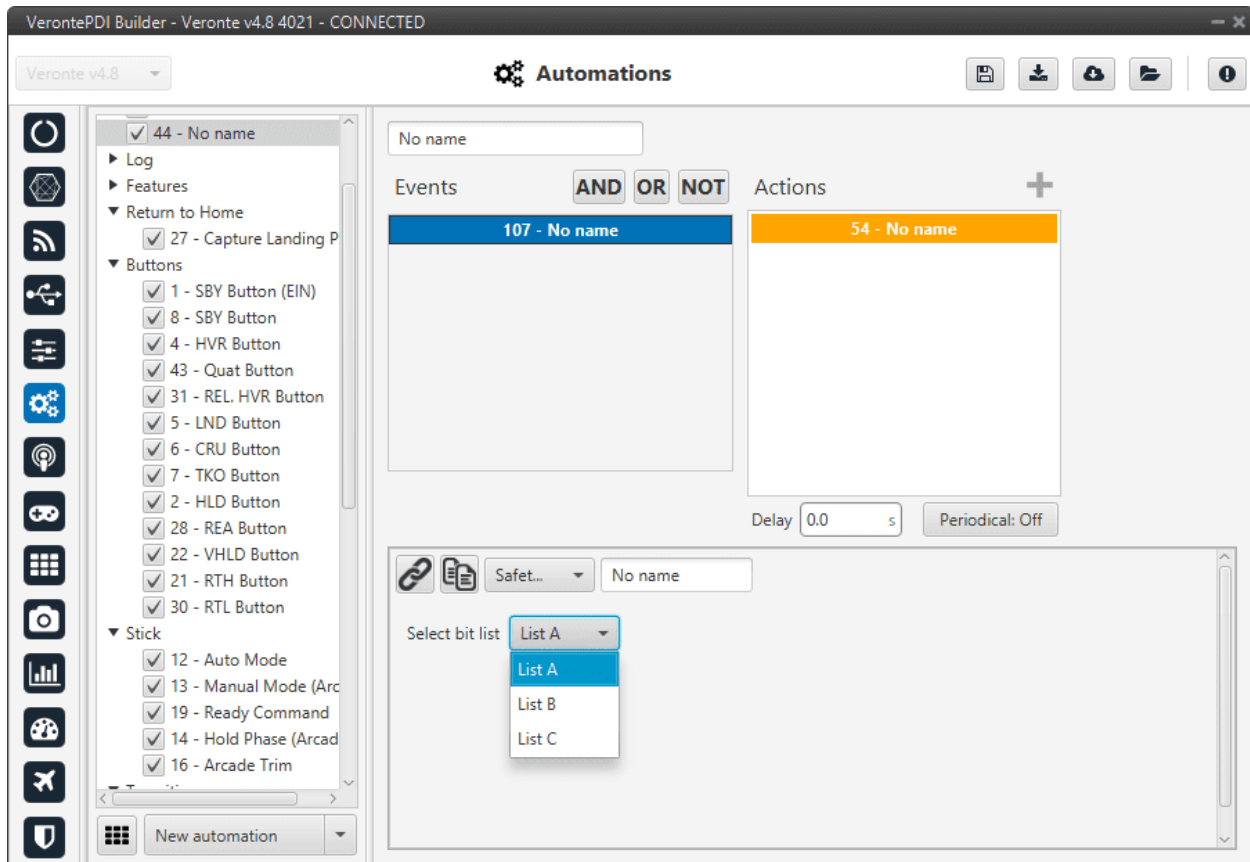


Fig. 177: Safety Bits action

This list must be configured previously. For more information about this, see section [Safety bits](#).

2.6.2.2.22 Select Arcade axis

The axes system of the aircraft is changed to one that has been previously created, see section [Arcade Axis](#) of the Control menu.

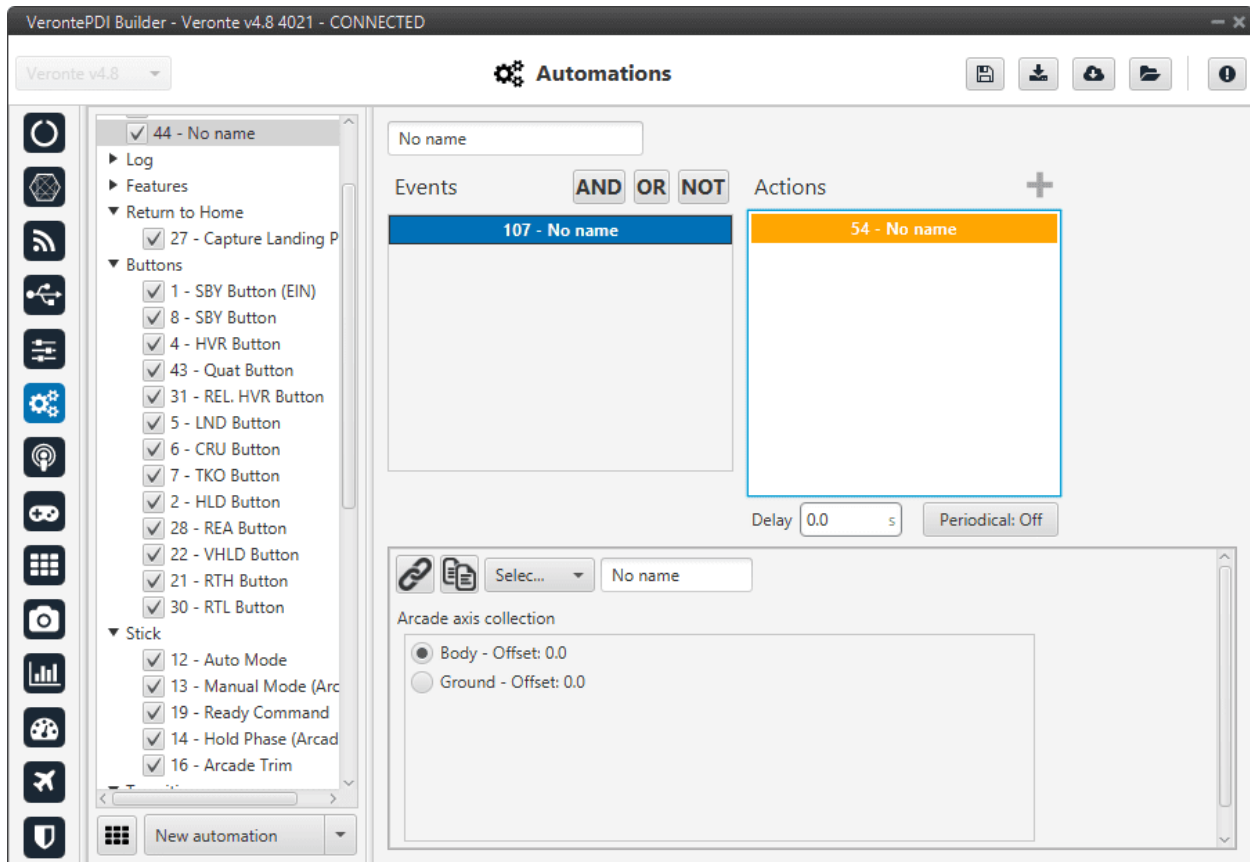


Fig. 178: Select Arcade axis action

2.6.2.2.23 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). By default priority table 0 is selected when 1x autopilot starts.

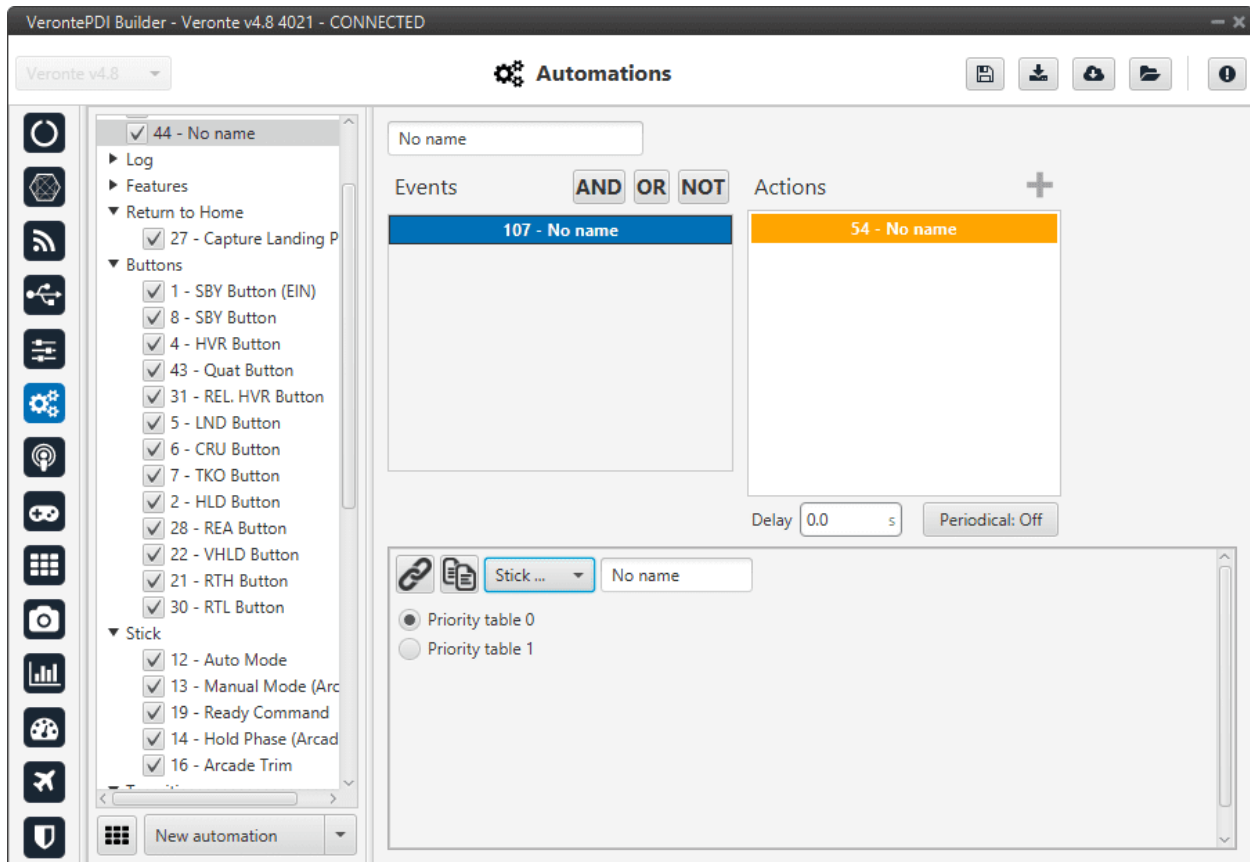


Fig. 179: Stick priority action

2.6.2.2.24 Terrain obstacle

This option is used to make the aircraft climb when is reaching an altitude of zero meters, for example, when flying towards a mountain. This option can't be activated all the time because it will not allow the aircraft to land.

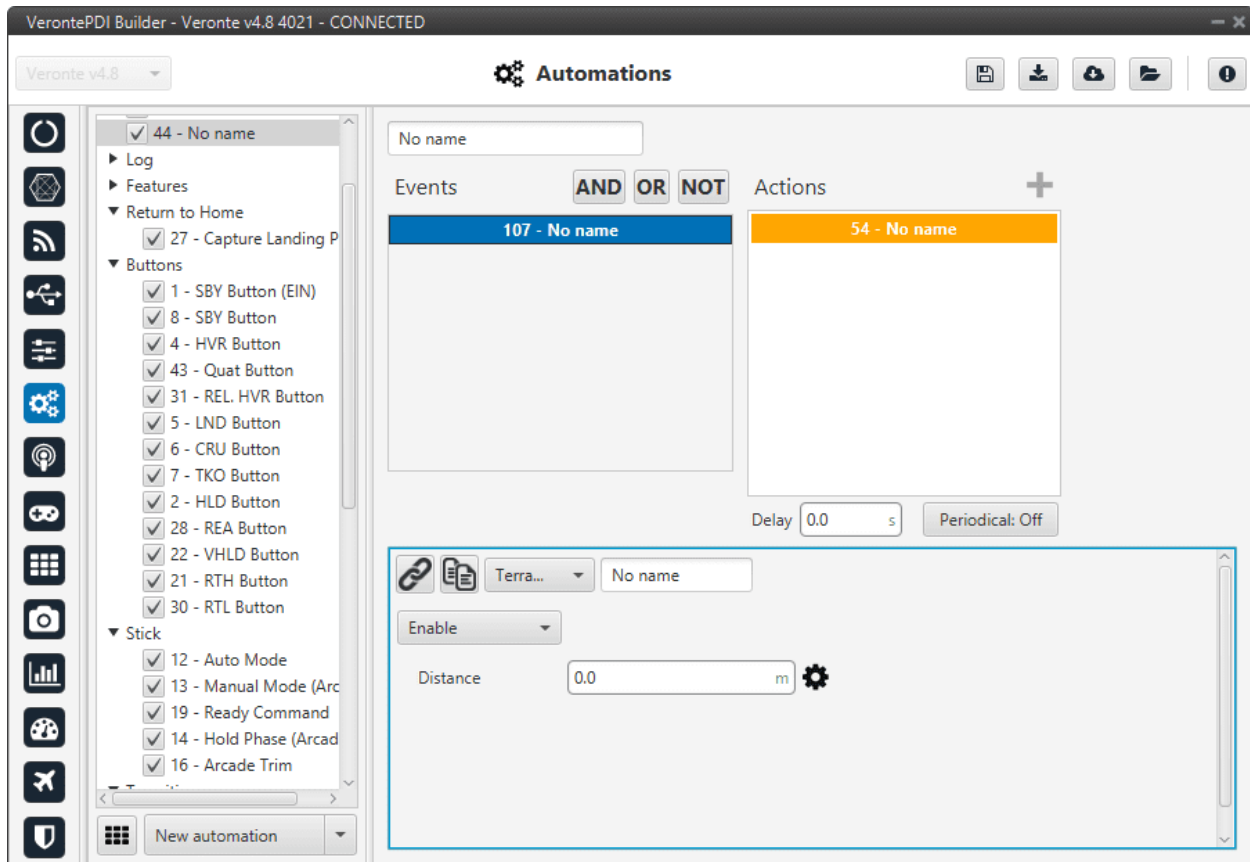


Fig. 180: Terrain obstacle action

- **Distance:** Establish how the aircraft will climb, it can be said to be a repulsion value. High values made the platform ascent quickly. This effect is more noticeable when the aircraft is close to the ground.

2.6.2.2.25 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.

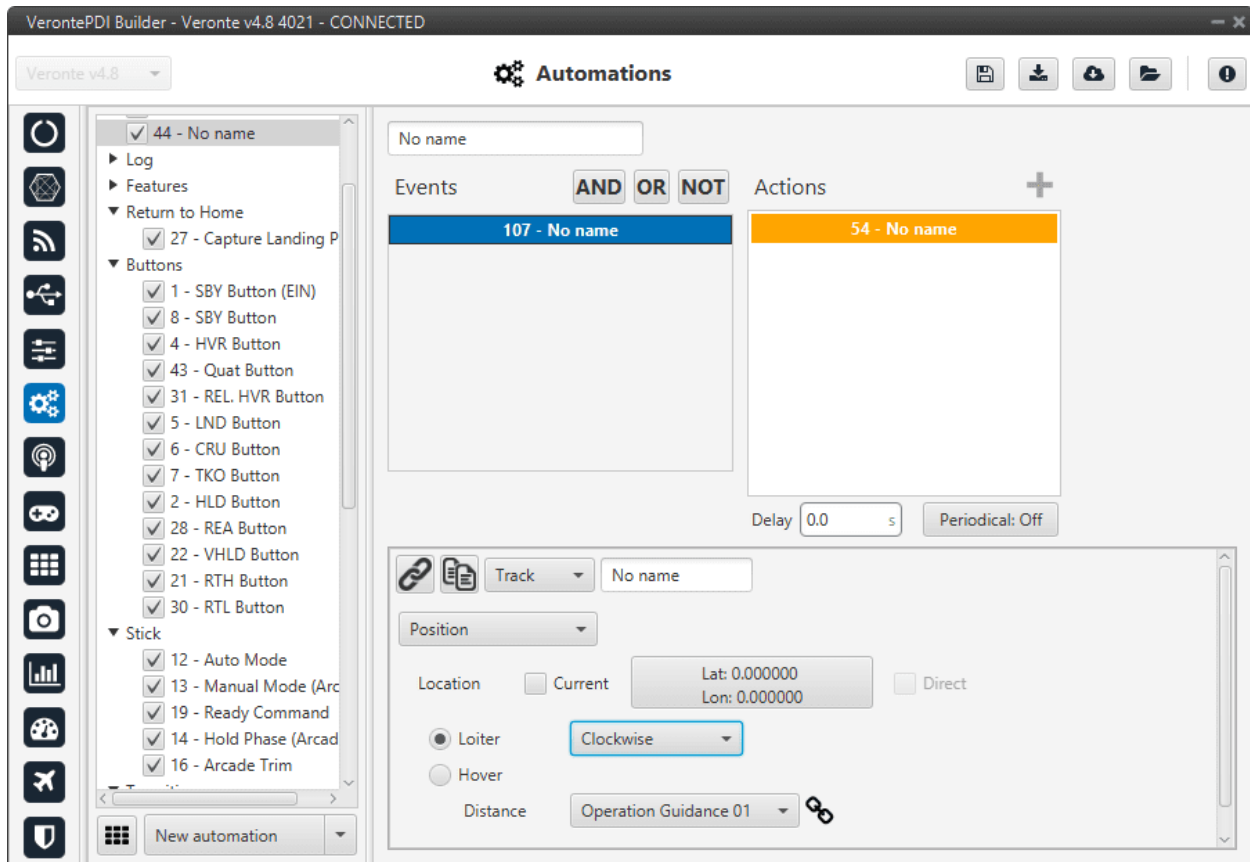


Fig. 181: Track action - Position

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 centre. If the UAV position is inside this circle, then 1x autopilot considers it is hovering correctly and will keep the position. If the centre of the hover changes its position and the UAV position is out of the hovering area, 1x will fly to the hovering centre, and once it is inside the circle, the hover will start.

Follow Leader

The platform will follow a moving object.

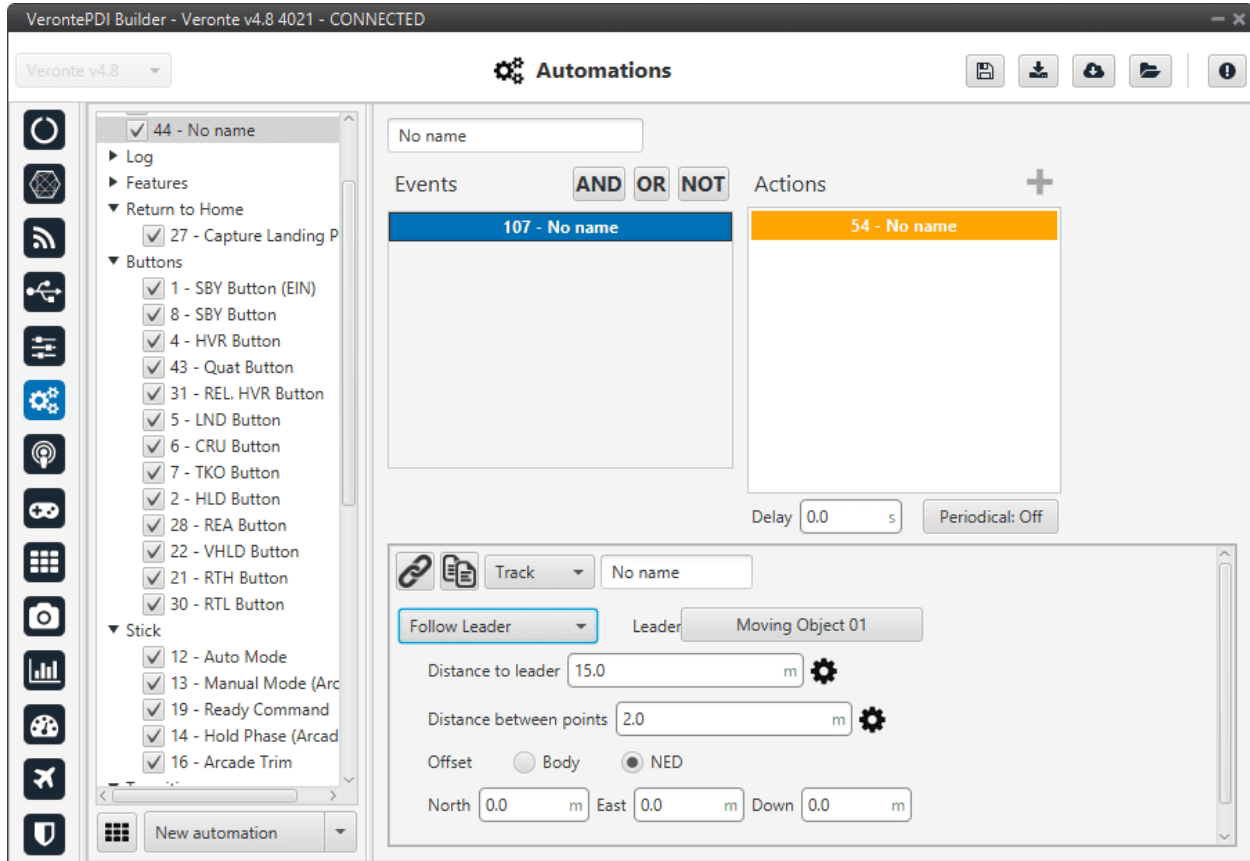


Fig. 182: Track action - Follow Leader

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, user has to follow the next steps:

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

2.6.2.2.26 User Log

An entry, previously configured in *User Log -> Telemetry section*, is added to the on-board log.

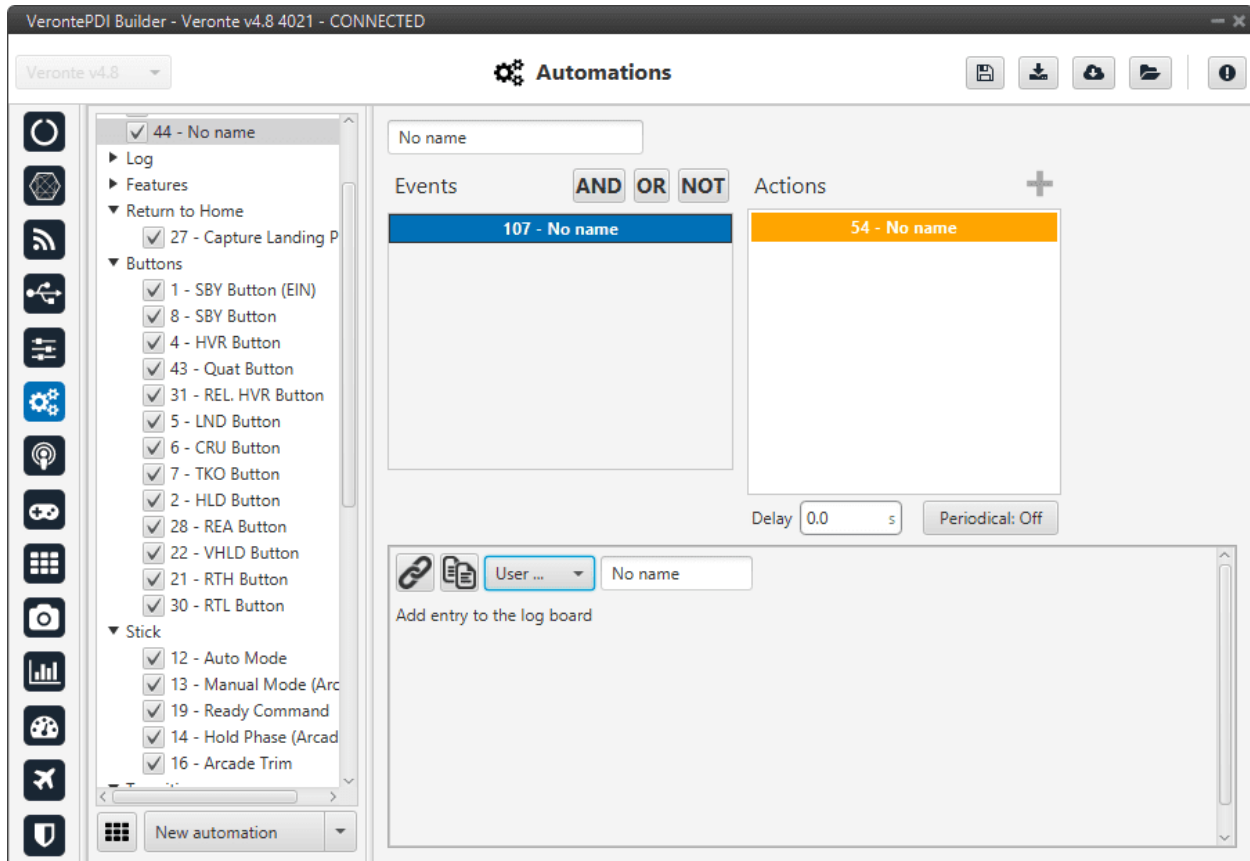


Fig. 183: User Log action

2.6.2.2.27 Variable

This action allows the user to select variables and save them in user variables.

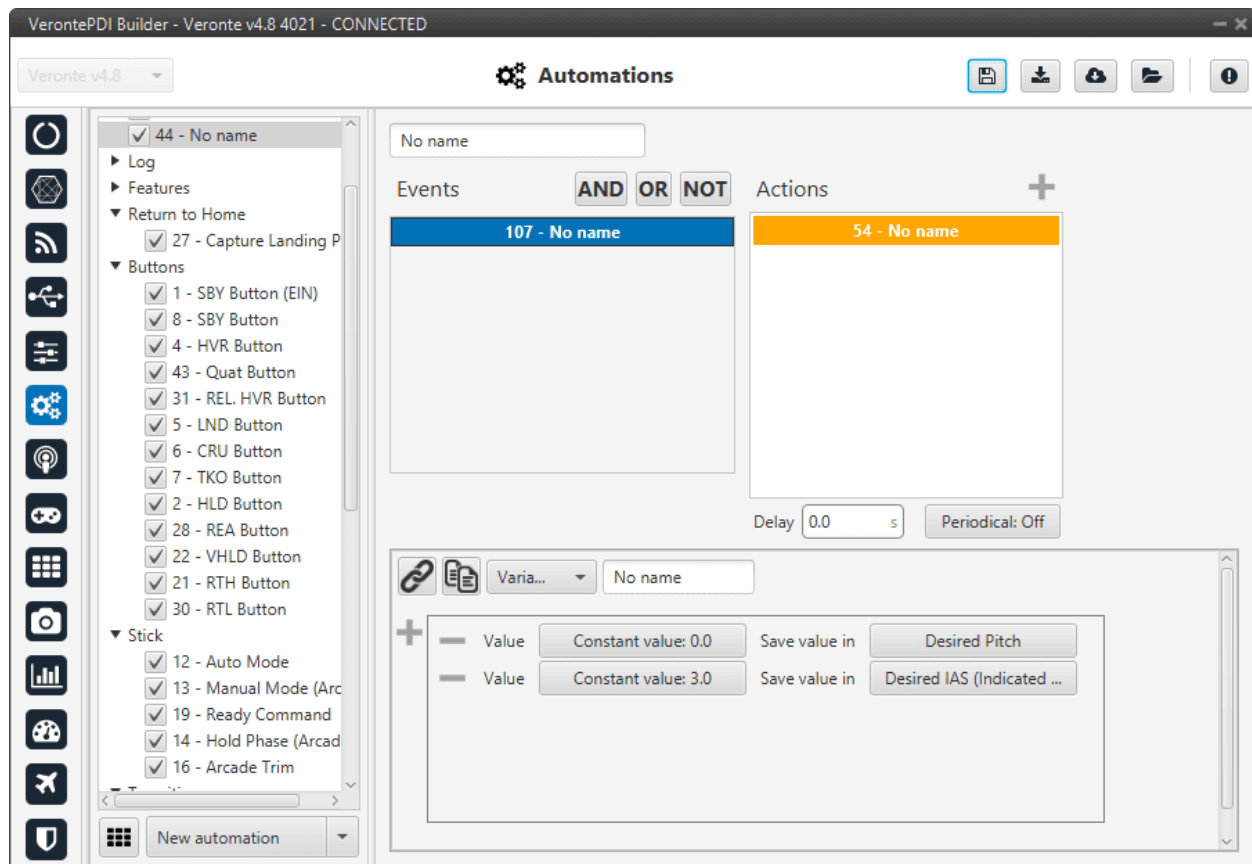


Fig. 184: Variable action

2.6.2.2.28 Yaw

When this action is triggered the actual yaw can be commanded.

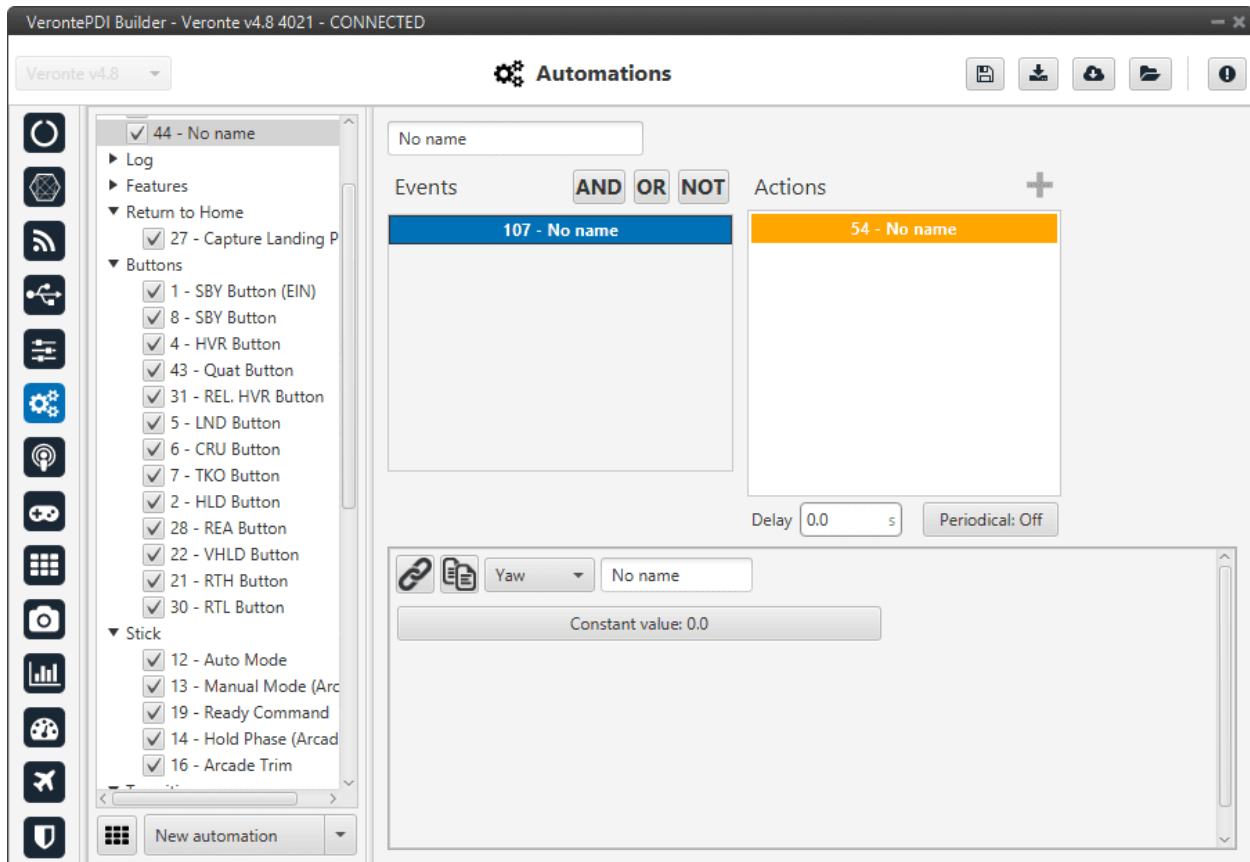


Fig. 185: Yaw action

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, 1x autopilot can be configured to **route** VCP messages for an external Veronte device with a known address (ID) through a given port.

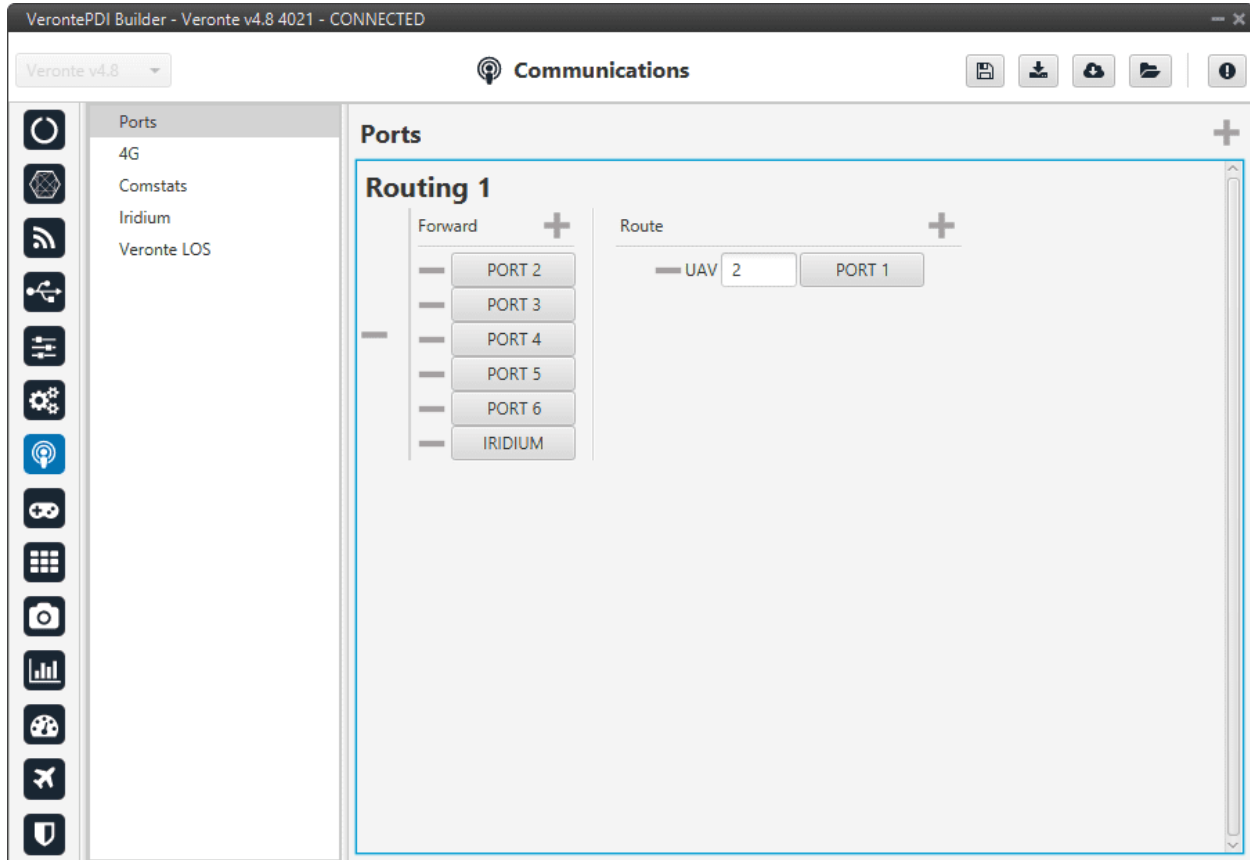


Fig. 186: Ports menu

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 using the *Ports action* of the Automations menu. Routing 1 will always be selected by default when booting 1x autopilot.

Application example

A practical example of the use of this menu are 1x Ground unit configurations. These configurations will have

configured a Routing of Address 2 (1x PDI Builder) through Port 1. This way, any messages that are received through a Commgr Port (i.e. through Veronte LOS), will be re-routed through Port 1 (USB) and received by 1x PDI Builder 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 [Maintenance mode - Troubleshooting section](#).

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*.

ESIM

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

The connection between the air unit and the ground station is stablished 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.

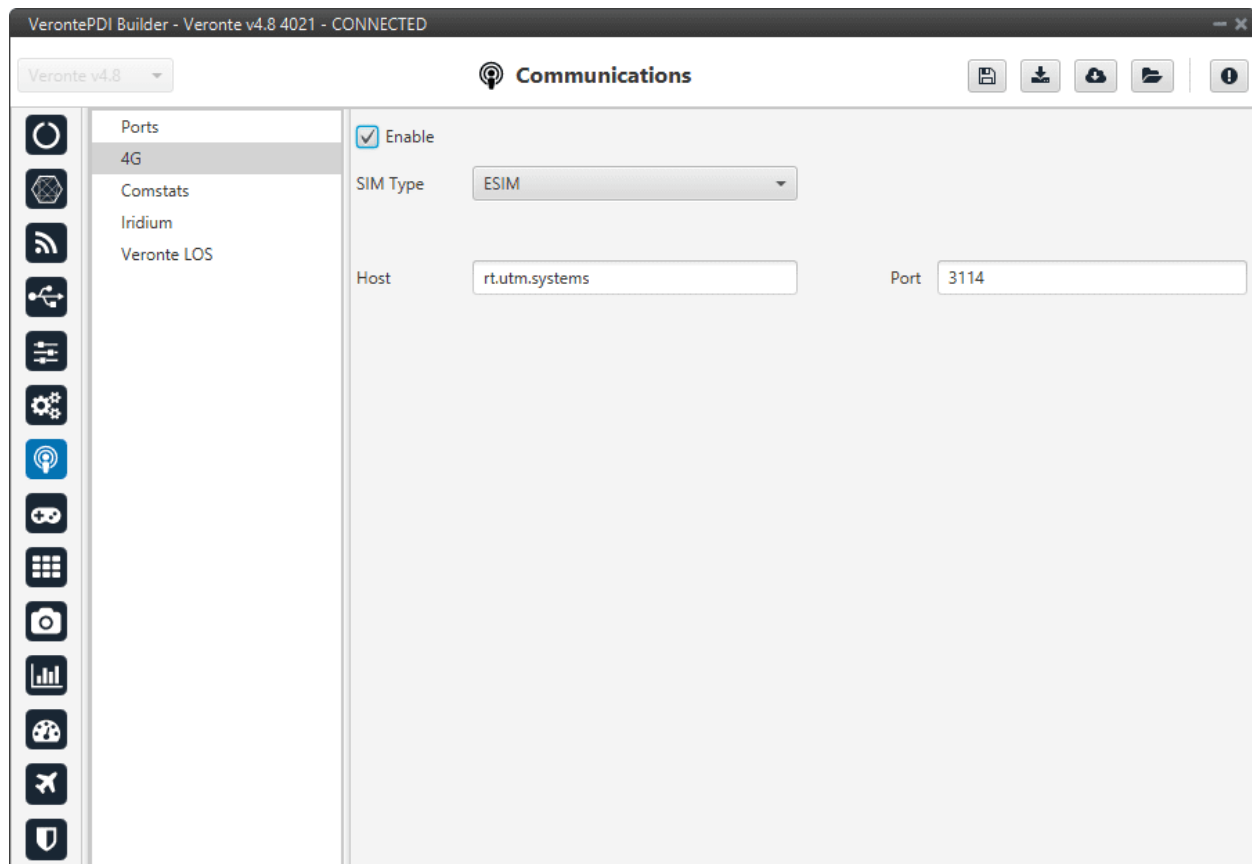


Fig. 187: ESIM menu

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 1x autopilot. PIN number and APN (Access Point Name) of the SIM card provider must be defined before enabling the 4G communication

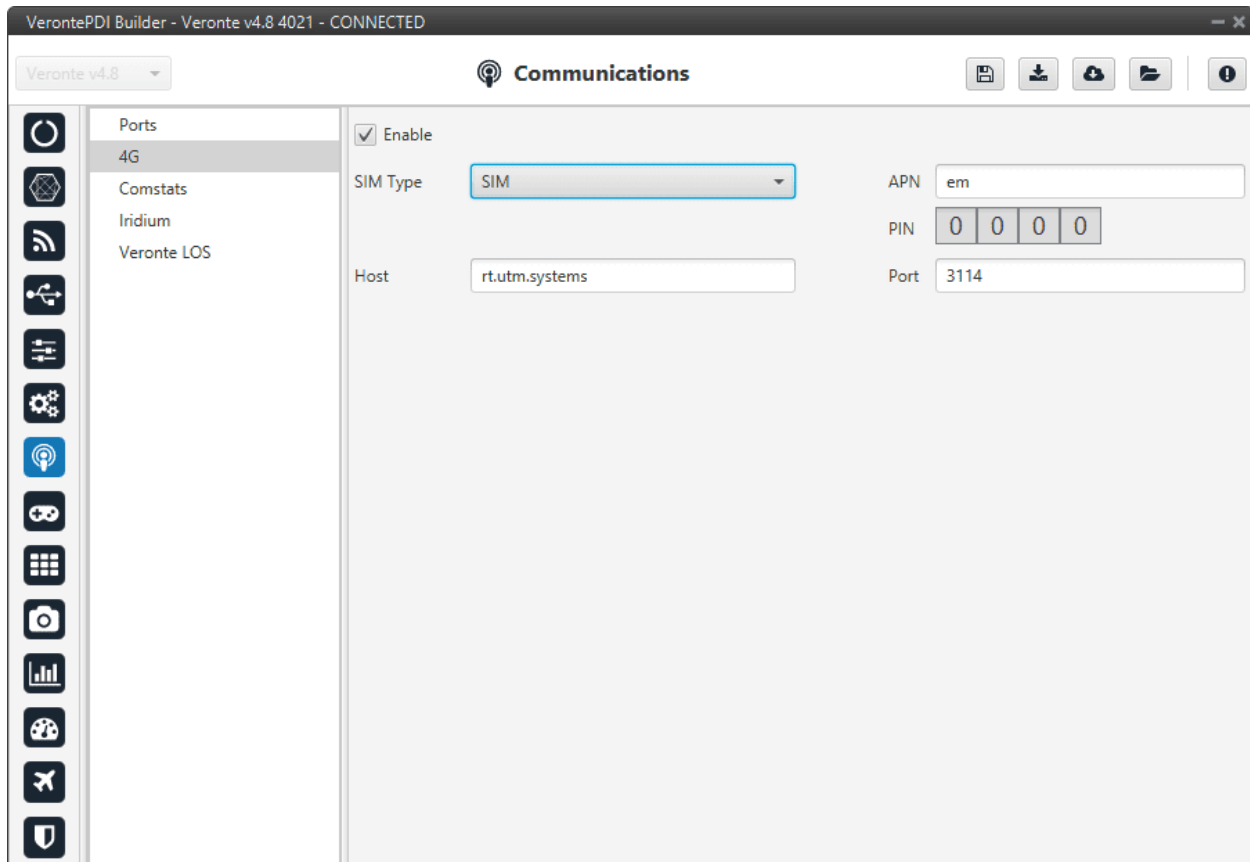


Fig. 188: SIM menu

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 1x autopilot 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.

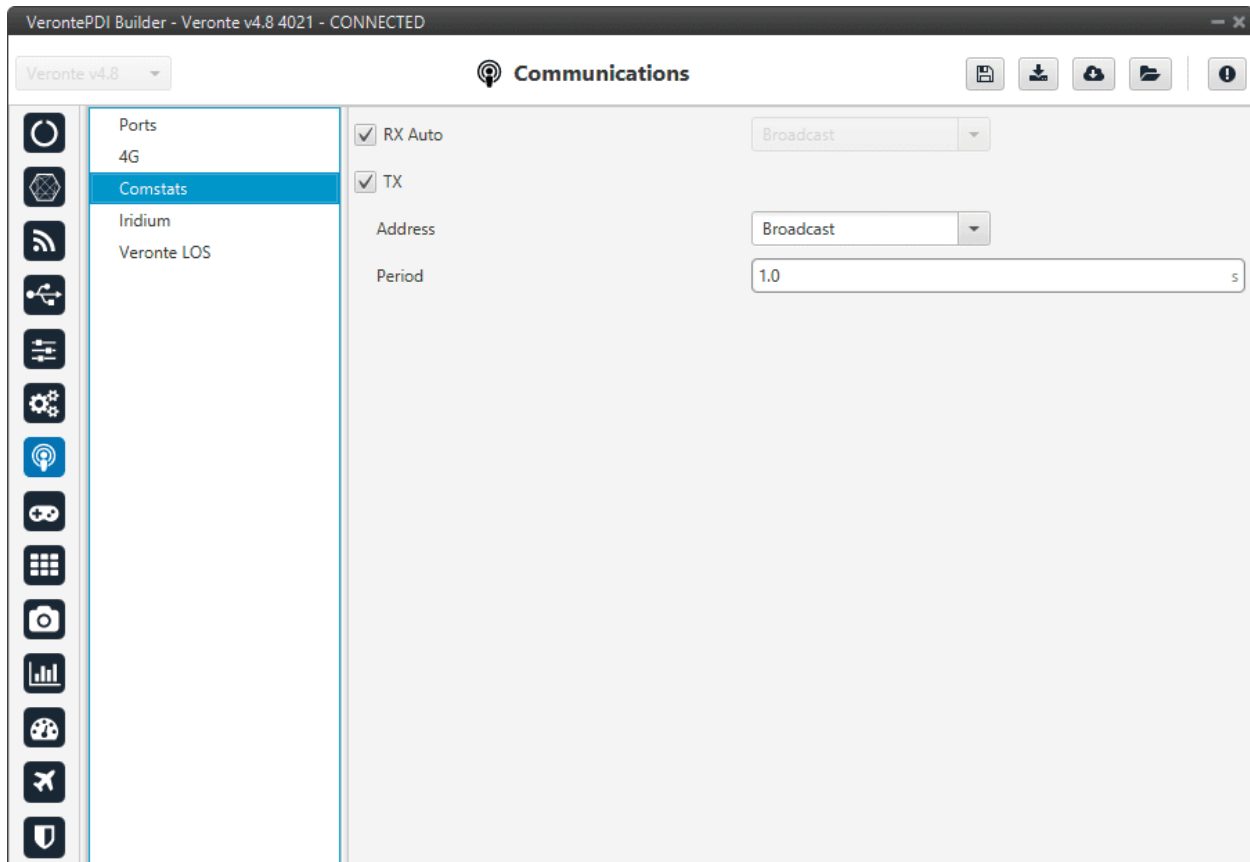


Fig. 189: Comstats menu

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 choose manually the address of the unit used for Comstats calculation.
- **TX:** When enabled, the unit will periodically send its Comstats message (set the period). Select the address to which the message should be sent:
 - App 2: Veronte Ops address.
 - Broadcast: All units on the network.
 - Veronte v4.X XXXX: To a specific unit.

Note: Enabling Tx will enable autopilot 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 possible 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. Please find more information about the registration process in this link: <https://docs.rockblock.rock7.com/docs/rockblock-management-system>.

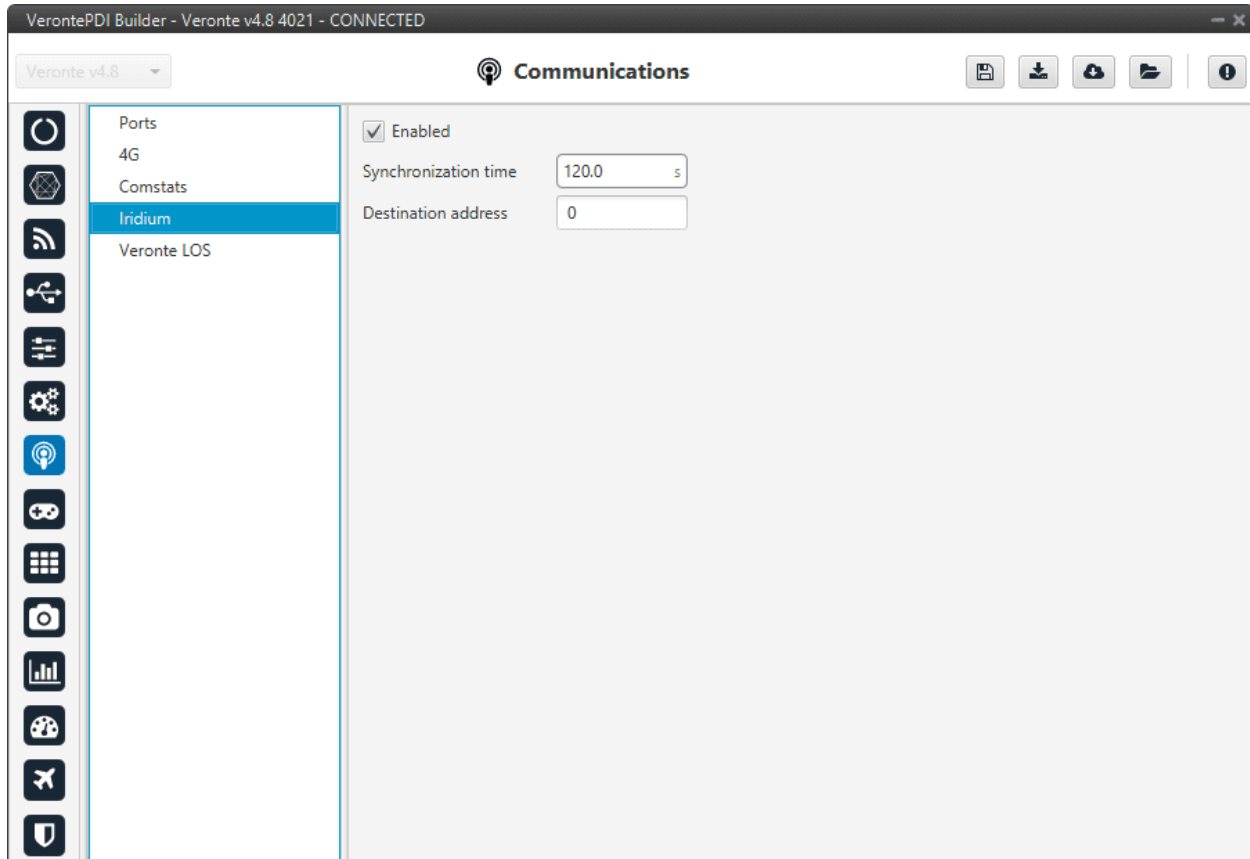


Fig. 190: Iridium menu

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 synchronization 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 synchronization time can be configured in order not to run out of credits.

2.7.5 Veronte LOS

In this section, 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.

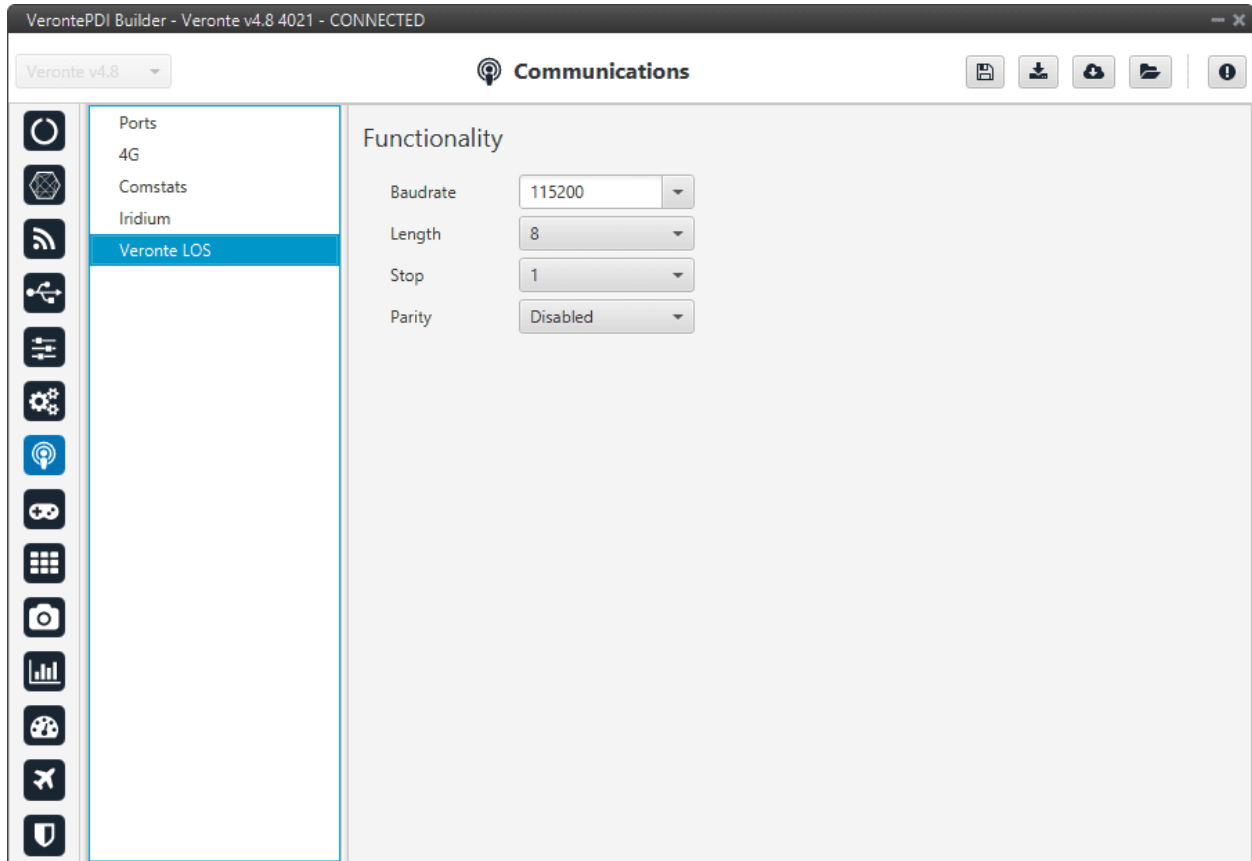


Fig. 191: Veronte LOS menu

- **Baudrate:** This specifies how fast data is sent over a serial line.
- **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, 1x autopilot should match with them in order to be able to communicate.

2.8 Stick

In this section, the stick configuration on 1x PDI Builder is explained.

This 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 (1-4)

The wired connected transmitters are configured through the following tabs.

2.8.1.1 PPM

This tab provides the options to configure a Pulse Position Modulation (PPM) radio controller to control the platform fitted with the autopilot.

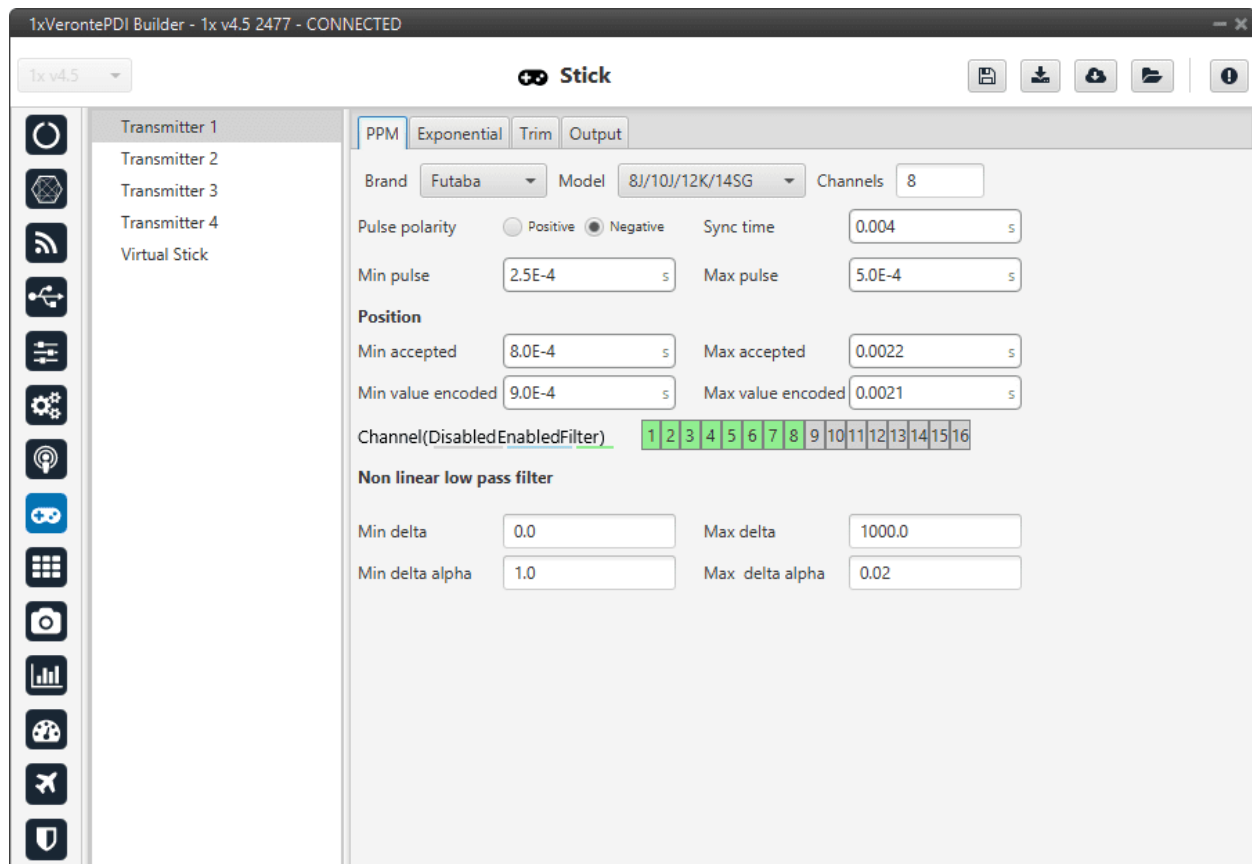


Fig. 192: Stick - PPM menu

- **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
	12K/14SG	12
	T18SZ	8
Jeti	DC 16/DC 24	16
FrSky	Taranis X9D	8
	Horus X12S	8
TBS	Croosfire	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

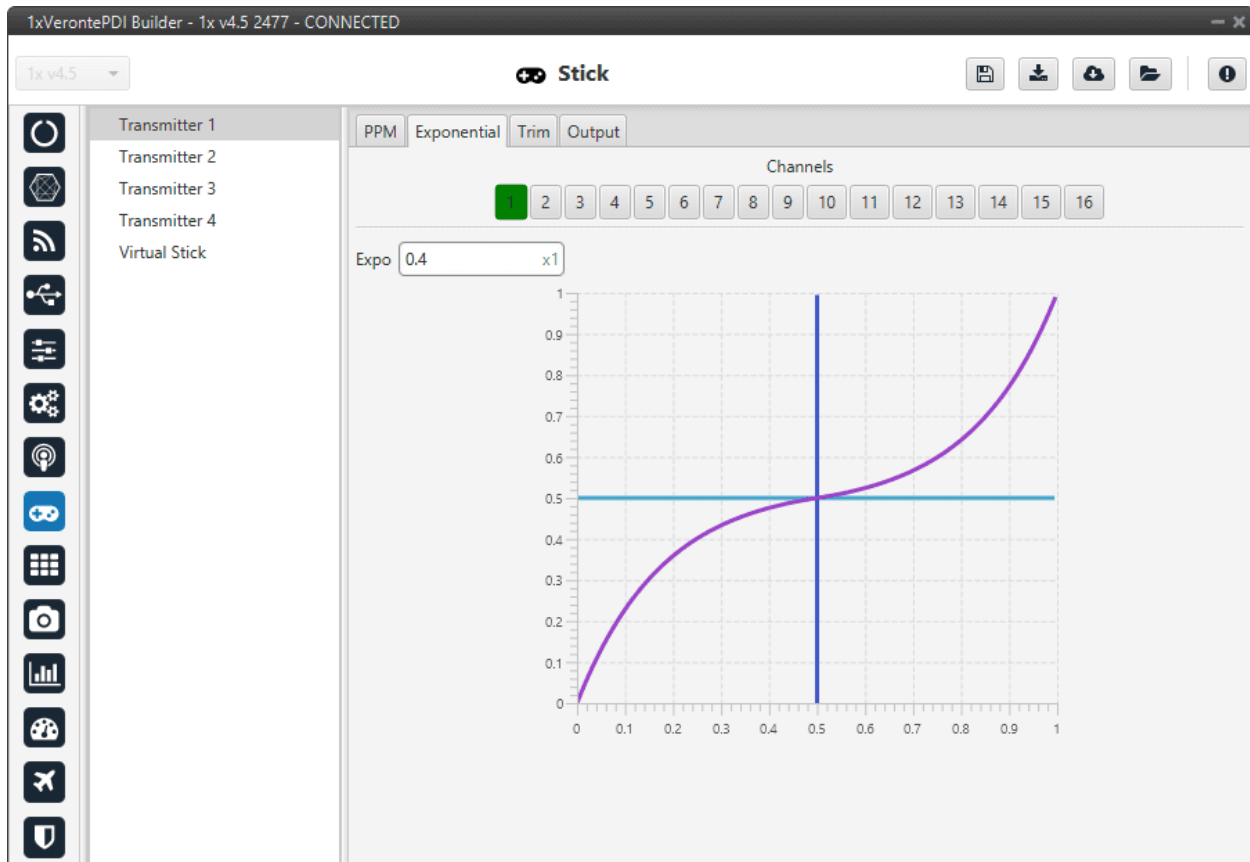


Fig. 194: Stick - Exponential menu

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

The third tab available is the Trim option.

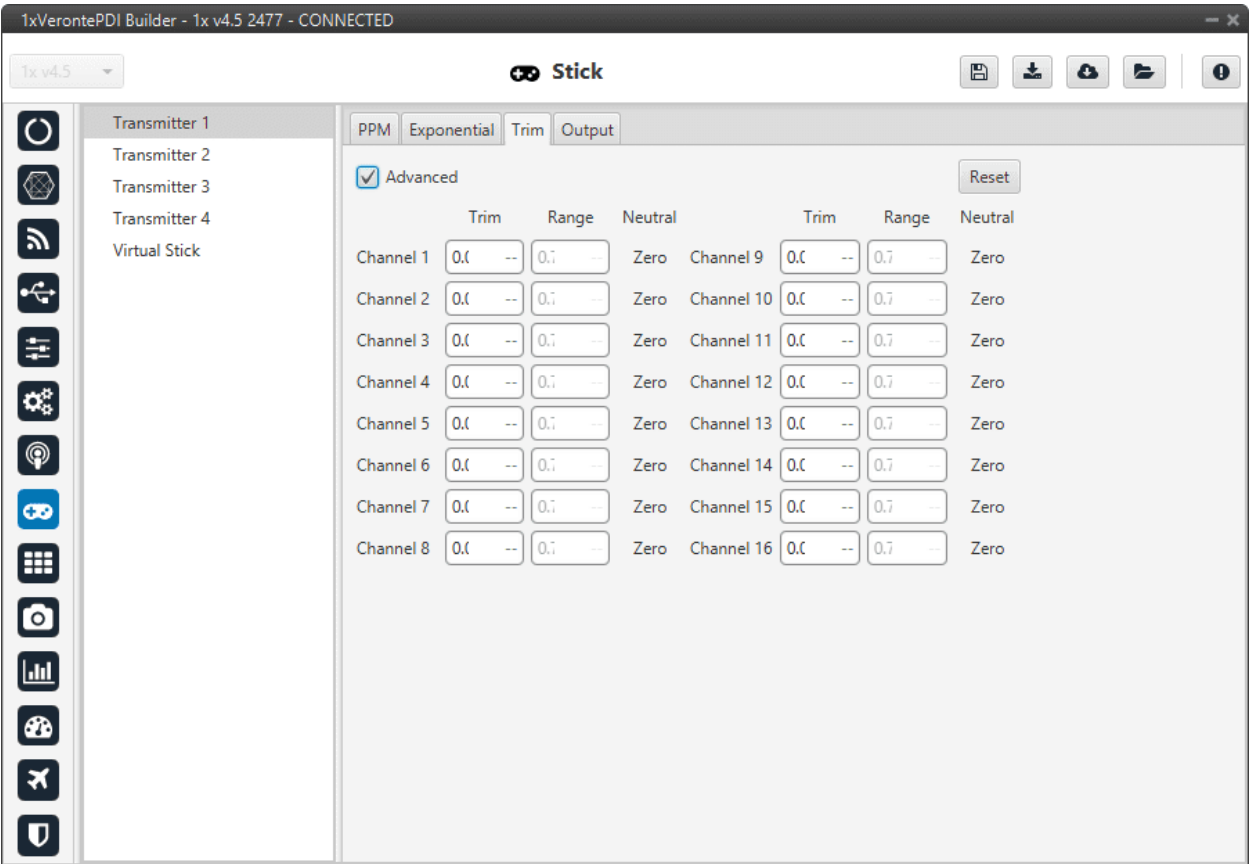


Fig. 195: Stick - Trim menu

By enabling the **Advanced** 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.

2.8.1.4 Output

In this menu 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.

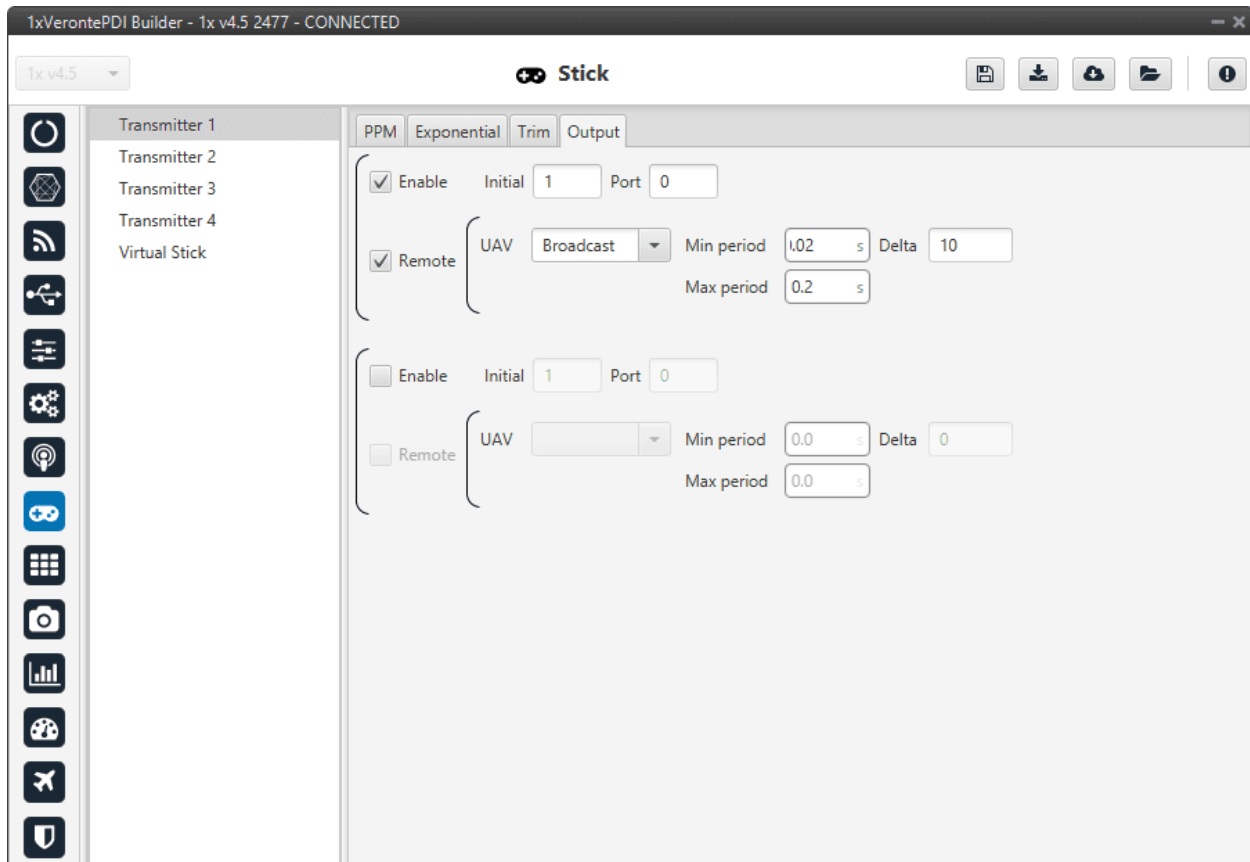


Fig. 196: Stick - Output menu

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 available:
 - * App 2: Veronte Ops address.
 - * Broadcast: The commands are sent to all units on the network. **We recommend this option.**
 - * Veronte v4.X XXXX: The address of a specific air unit.
 - **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**. We recommend **0.02s**.

- **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**. We recommend **0.2s**.
- **Delta:** This parameter determines whether the frequency is set to the minimum or maximum period set above.

If 1x Autopilot 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). We recommend **10**.

Note: An example of the *Stick integration* can be found in the Integration examples section.

2.8.2 Virtual Stick

In this menu a virtual stick can be defined. It can be useful when:

- Testing the autopilot configuration (not flight tests).
- Using a USB stick. Please find more information about this in the *USB -> Integration examples section*.
- The information of a stick is received through a different channel than PPM, so that the virtual stick will process that information and input it into the system.

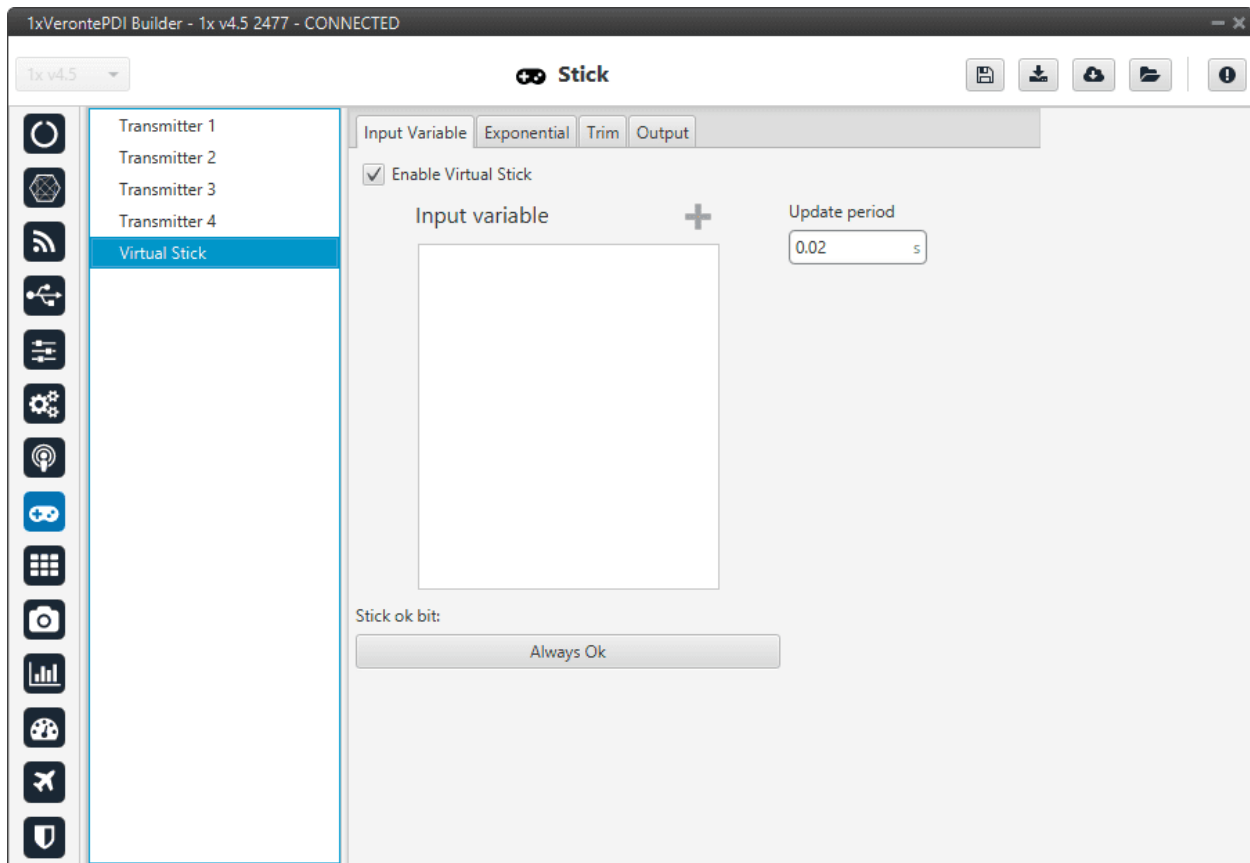


Fig. 197: Virtual stick menu

In this menu the user can configure:

- **Enable Virtual Stick**
- **Input variables:** Place here the variables containing the stick information. It is not necessary when using a USB stick or simply virtual stick.
- **Update Period:** Configure the period required. We recommend **0.02 s**.

In addition, a **virtual stick widget** must be configured in **Veronte Ops**. Please find more information in the [Stick -> Widgets](#) section of the **Veronte Ops** manual.

The tabs **Trim** and **Output** are the same as the Transmitter ones, so refer to the [Transmitter](#) menu for more information.

Note: An example of the [Virtual stick integration](#) can be found in the Integration examples section.

2.9 Block Programs

Block programs are the **core of 1x autopilot**. 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 1x autopilot. 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.

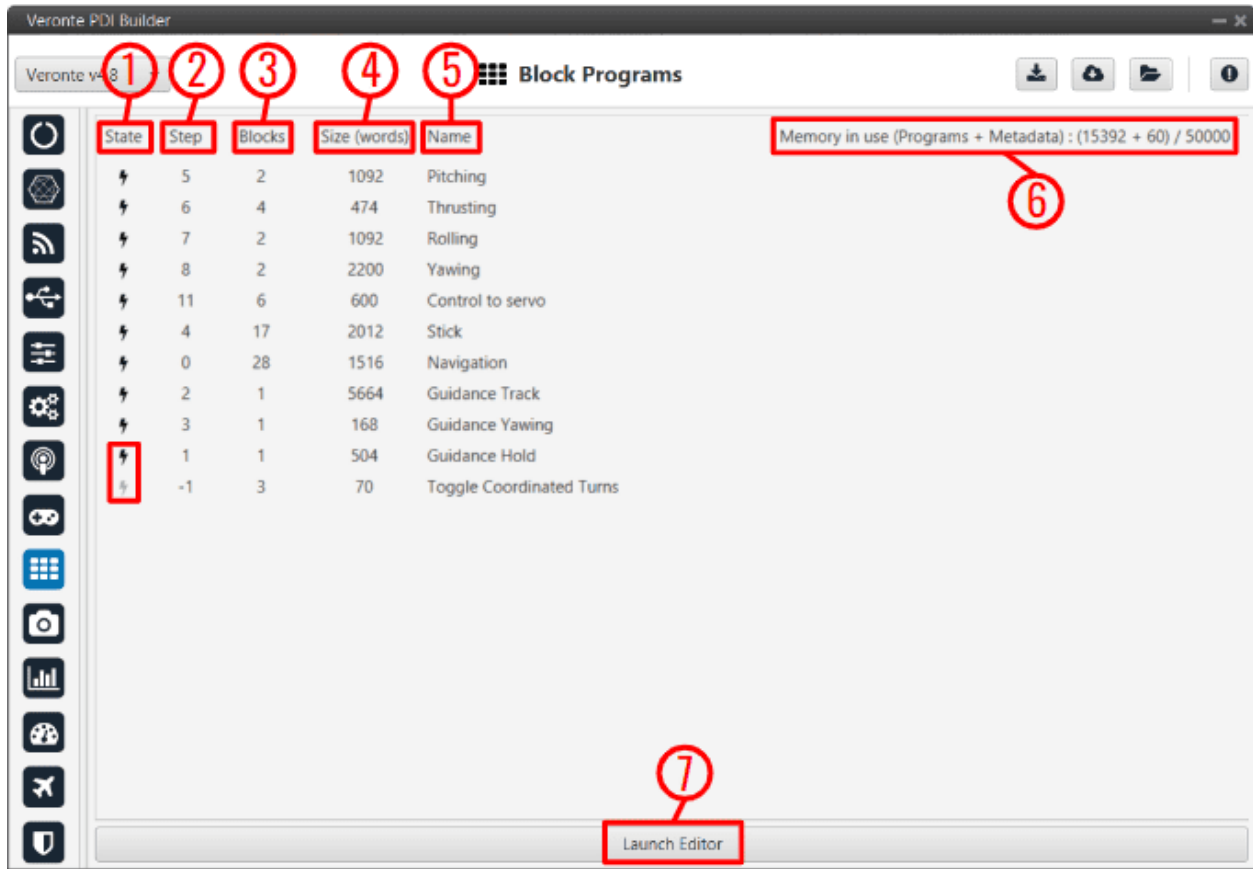
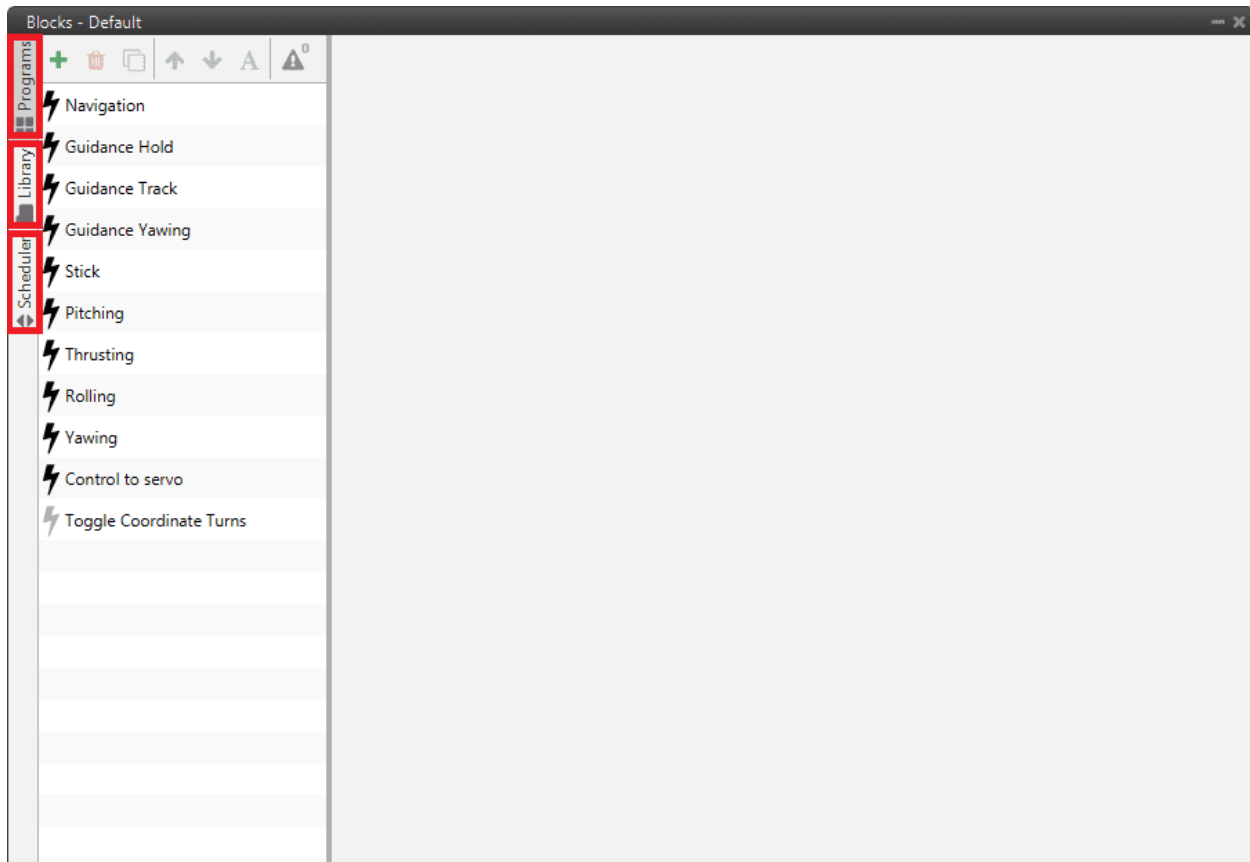


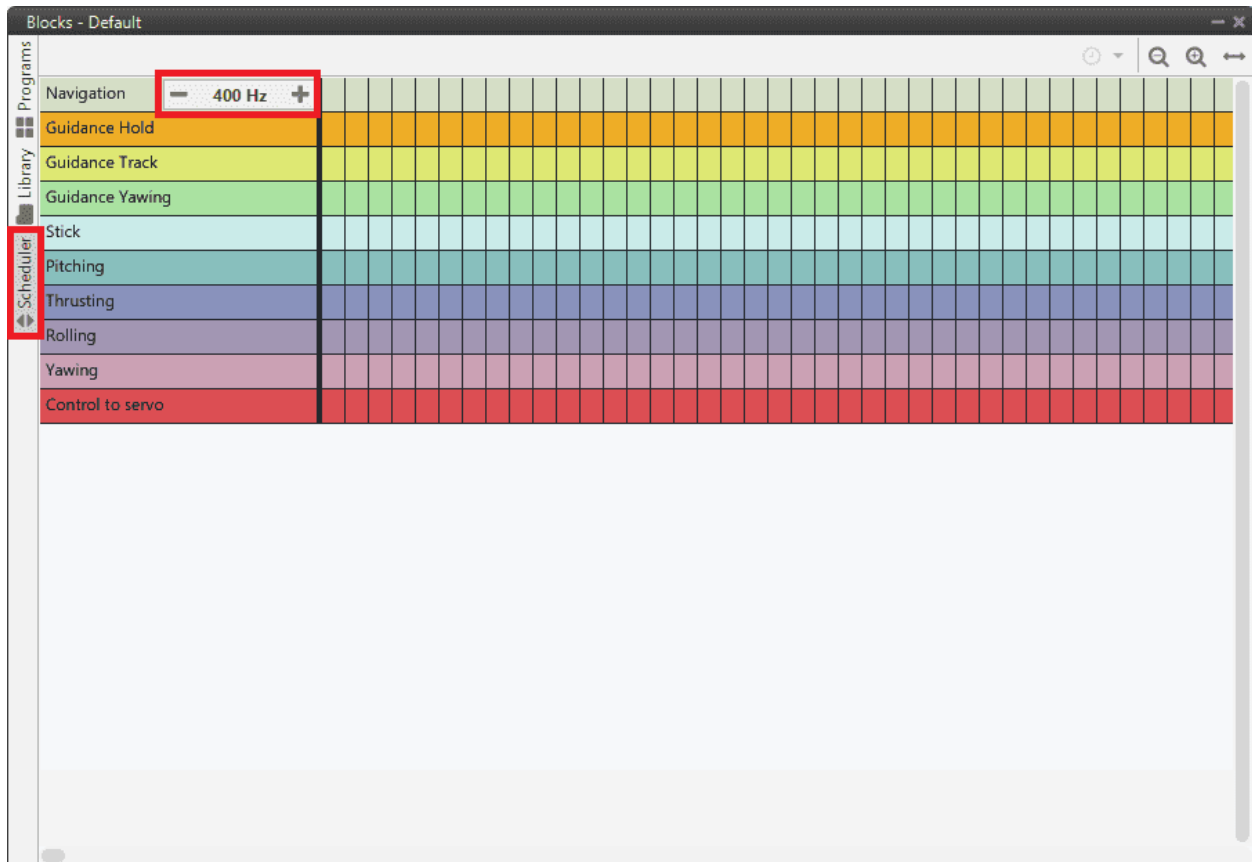
Fig. 198: Block Programs menu

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 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. 199: **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. 200: **Block Programs - 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:

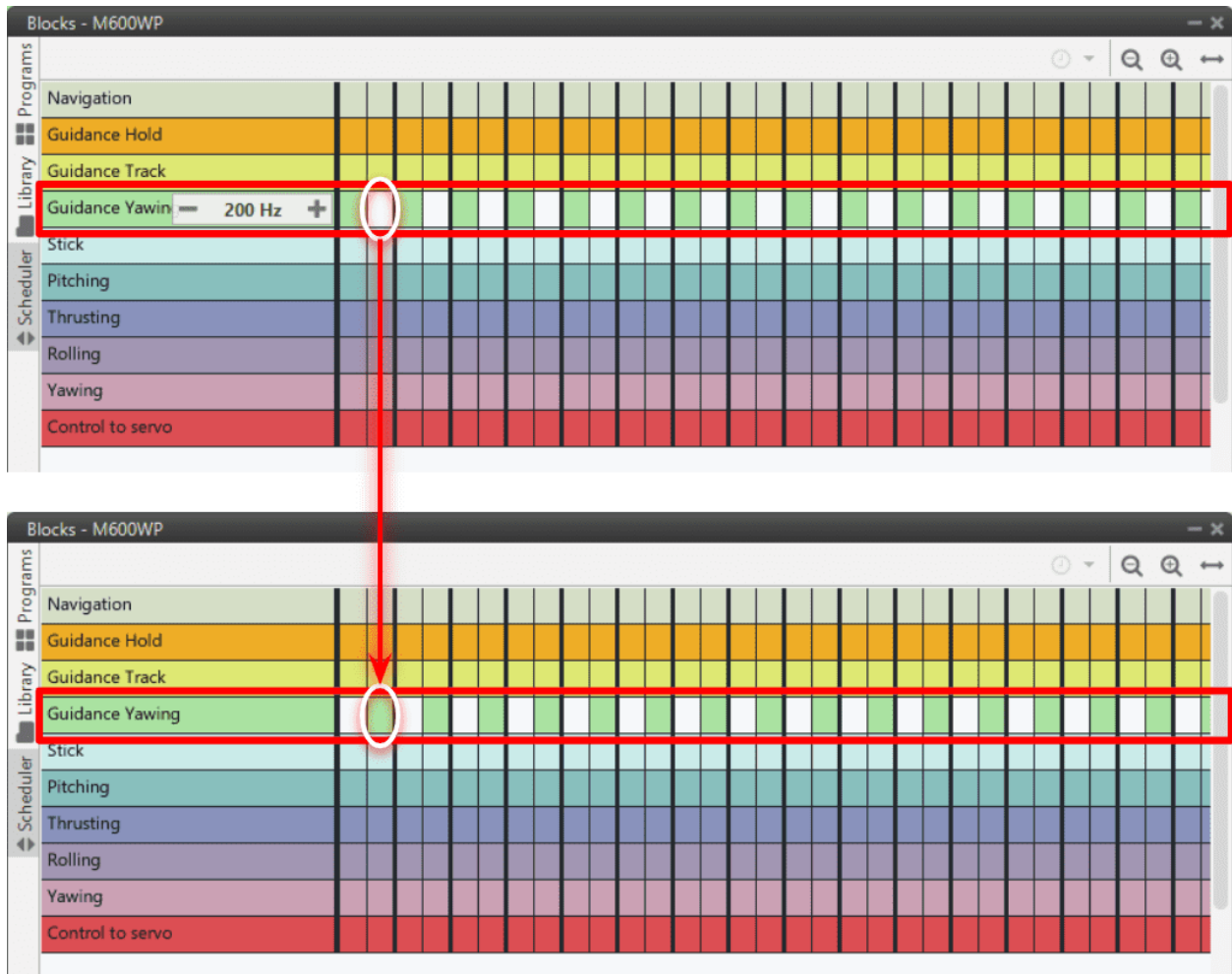


Fig. 201: Block Programs - Scheduler slots

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:

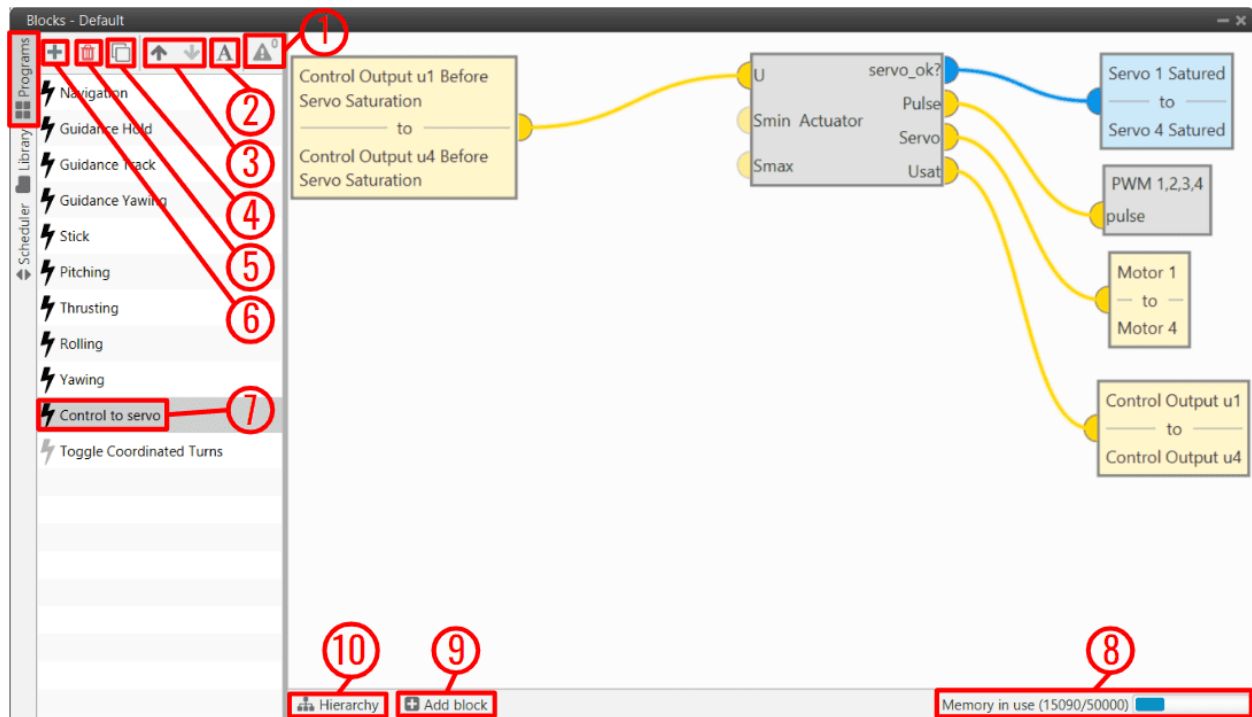


Fig. 202: Programs options









- 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. 203: Move to error

- B. **Change name** : 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 organisation and position of each block in the diagram that also represents part of this space.

Tip: To optimise memory, it is better to use more but smaller blocks than one large block.

- I. **Add block:** By clicking here, a new column menu will appear where the user can choose the block he wants to add.

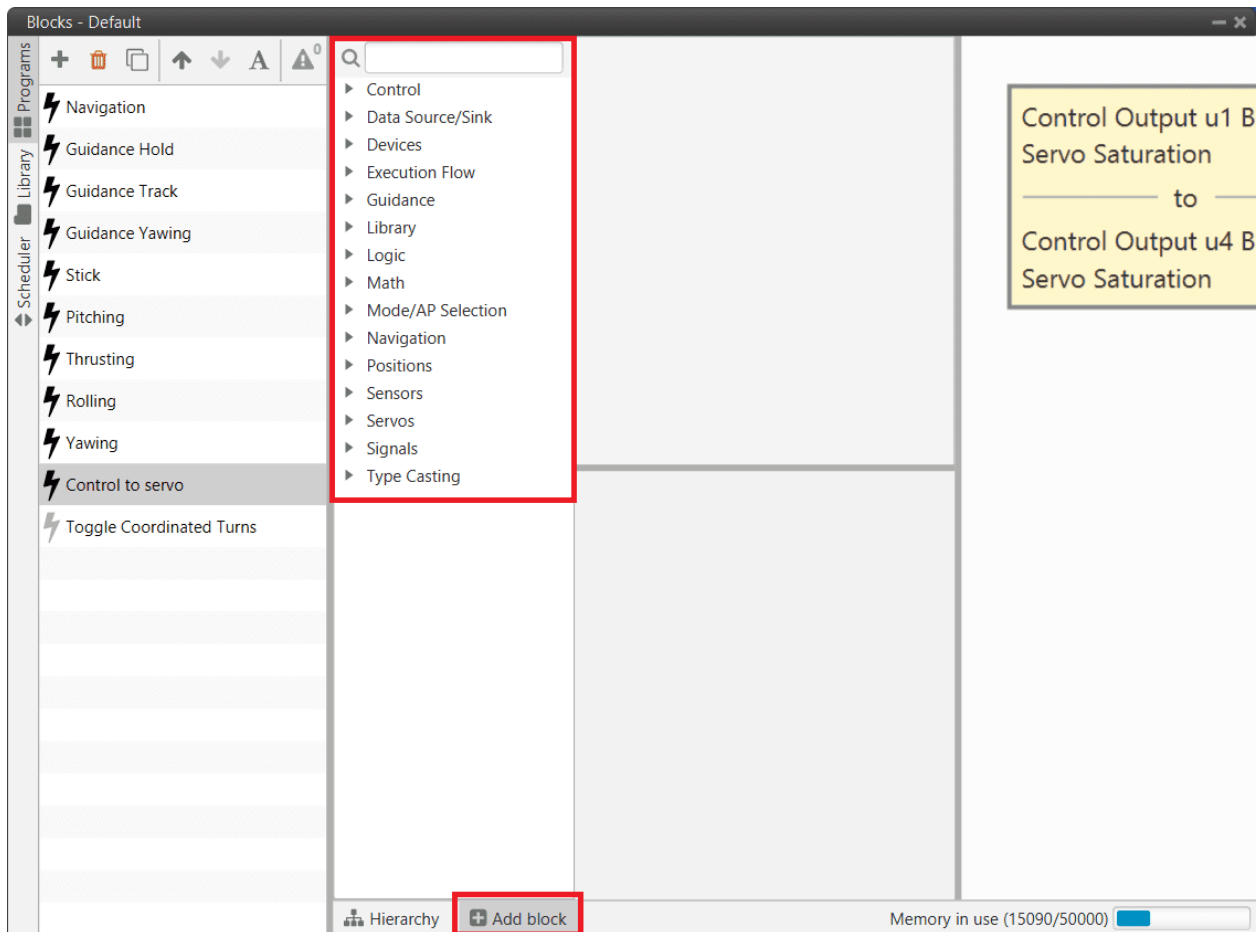


Fig. 204: Add block menu

- J. **Hierarchy:** By clicking here, a new column menu will appear where the user can see “information” about the existing blocks on the selected program.

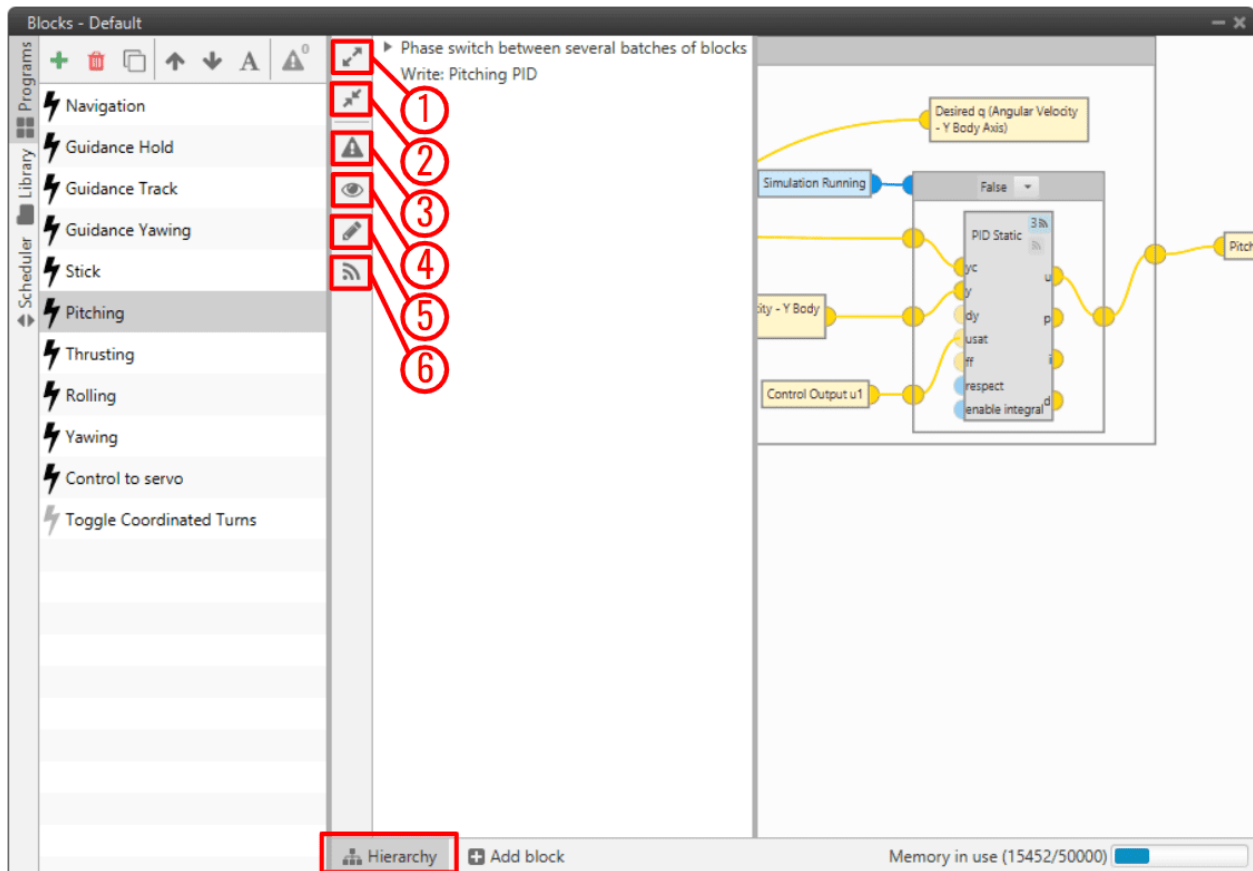
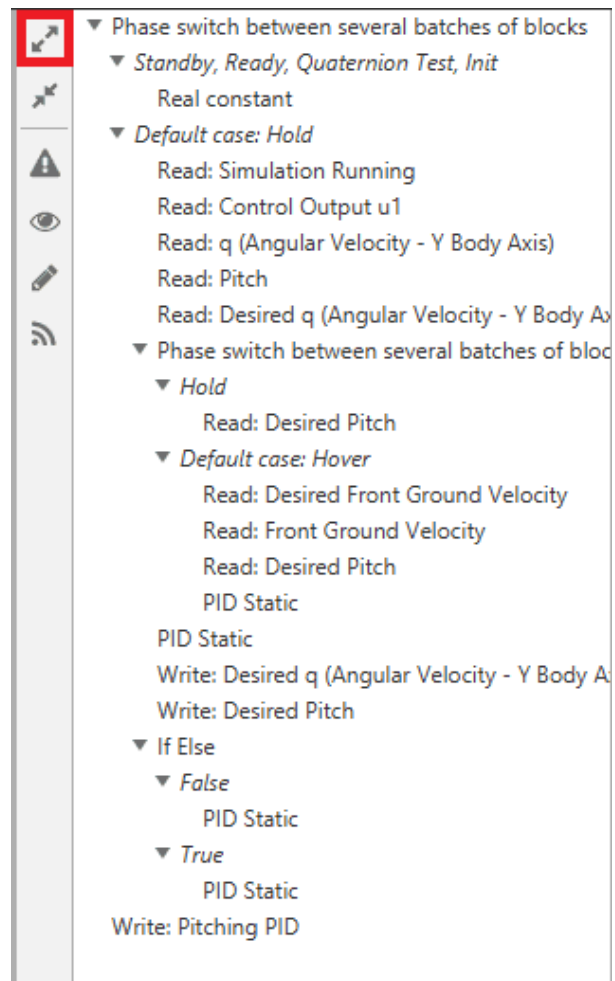
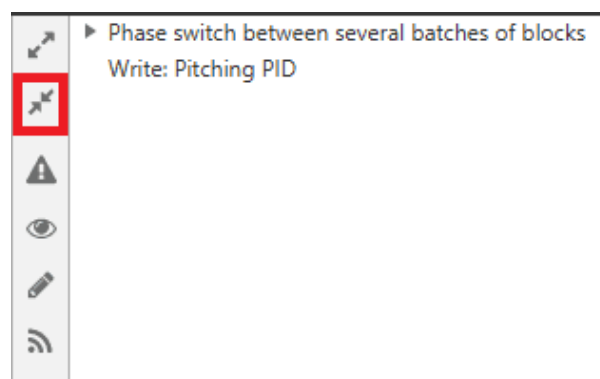


Fig. 205: Hierarchy menu

- a. **Expand All:** By clicking on it, all menus that are collapsed will be expanded, e.g. in programs that have **Switch** blocks.

Fig. 206: **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. 207: **Collapse All**

- c. **Show blocks with errors:**

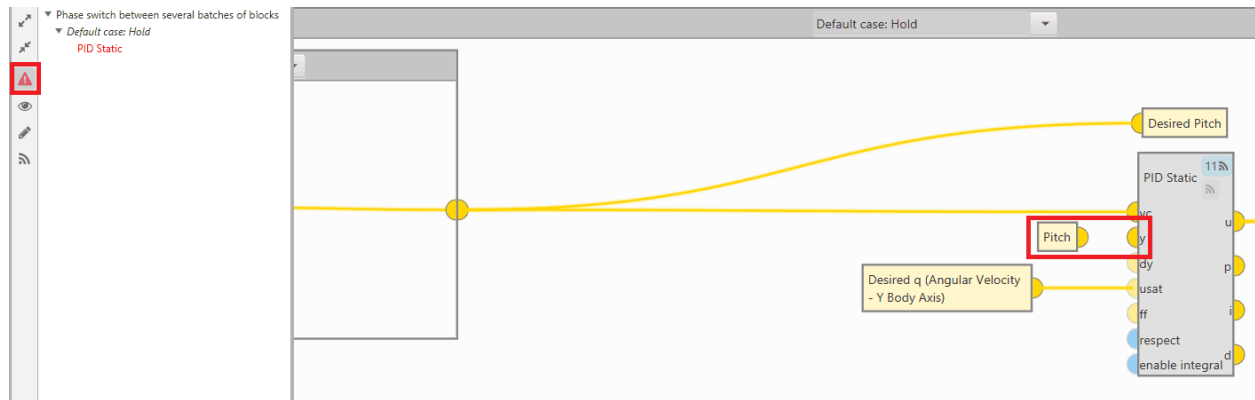


Fig. 208: Show blocks with errors

d. Show blocks that read from variables:

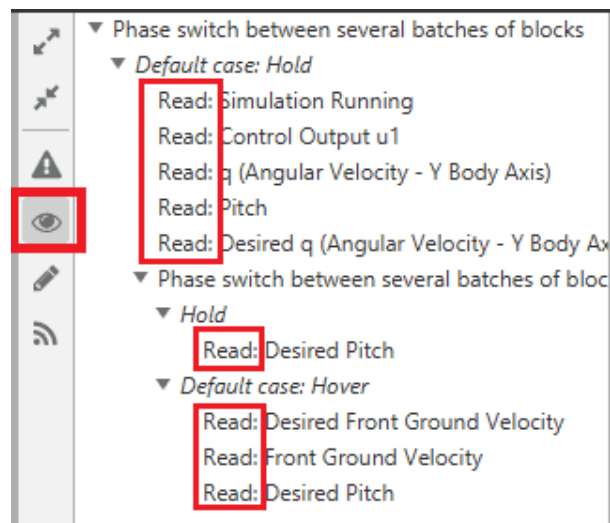


Fig. 209: Show blocks that read from variables

e. Show blocks that write in variables:

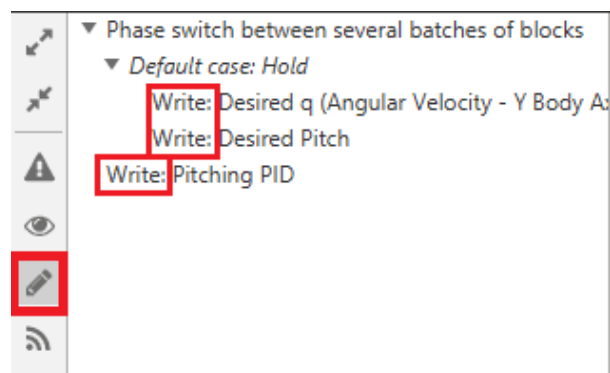


Fig. 210: Show blocks that write in variables

f. Show blocks that can be commanded:

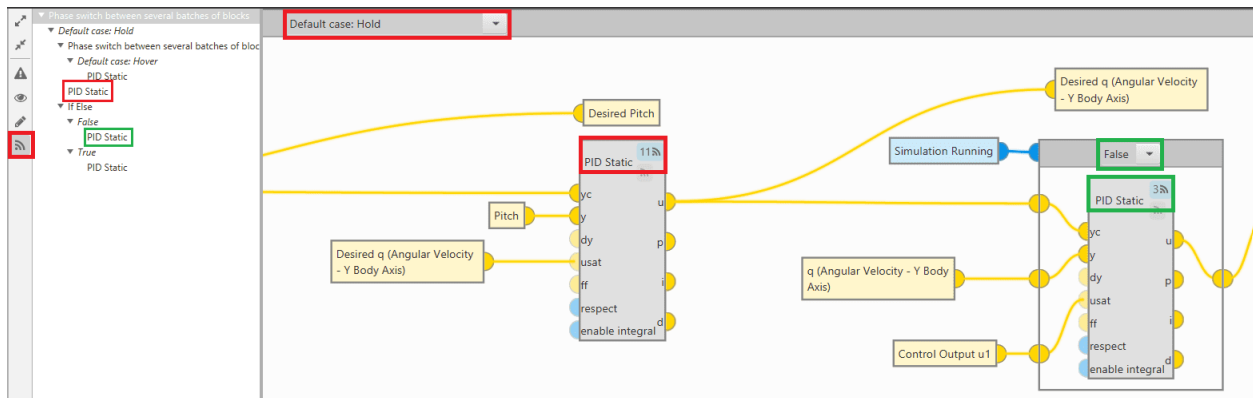







Fig. 211: Show blocks that can be commanded



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 variable**: User can change the selected variable, e.g. to be read or written.
- To **re-locate blocks**, just *click and drag* them.

Fig. 212: Adding blocks

Block inputs and outputs use a colour code in order to indicate variable types:

-  : 32-bit Real variables.
-  : BIT Boolean variables.
-  : 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. 213: **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](#).

Fig. 214: **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. 215: **Errors in blocks**



- An exception of the previous rule are translucent 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 1x autopilot 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 (4x autopilot).
- **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.

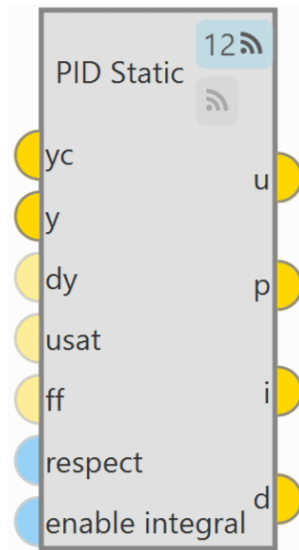


Fig. 216: **PID block**

As can be seen in the figure above, PID Static block has 2 buttons (which are like 'wifi icons') that enable or disable the block to be commanded from the [1x PDI Tuning](#) software.

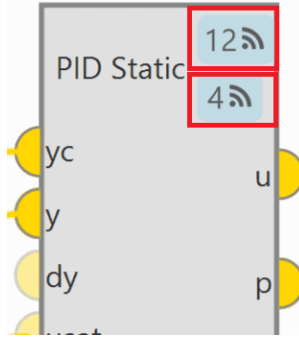


Fig. 217: PID block - Command buttons

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:

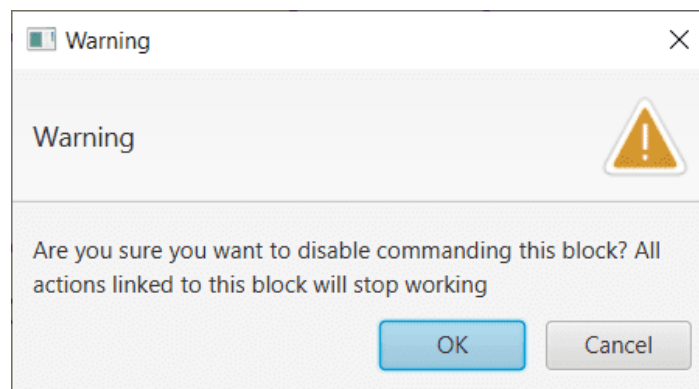


Fig. 218: Warning message when disabling the command

- 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.

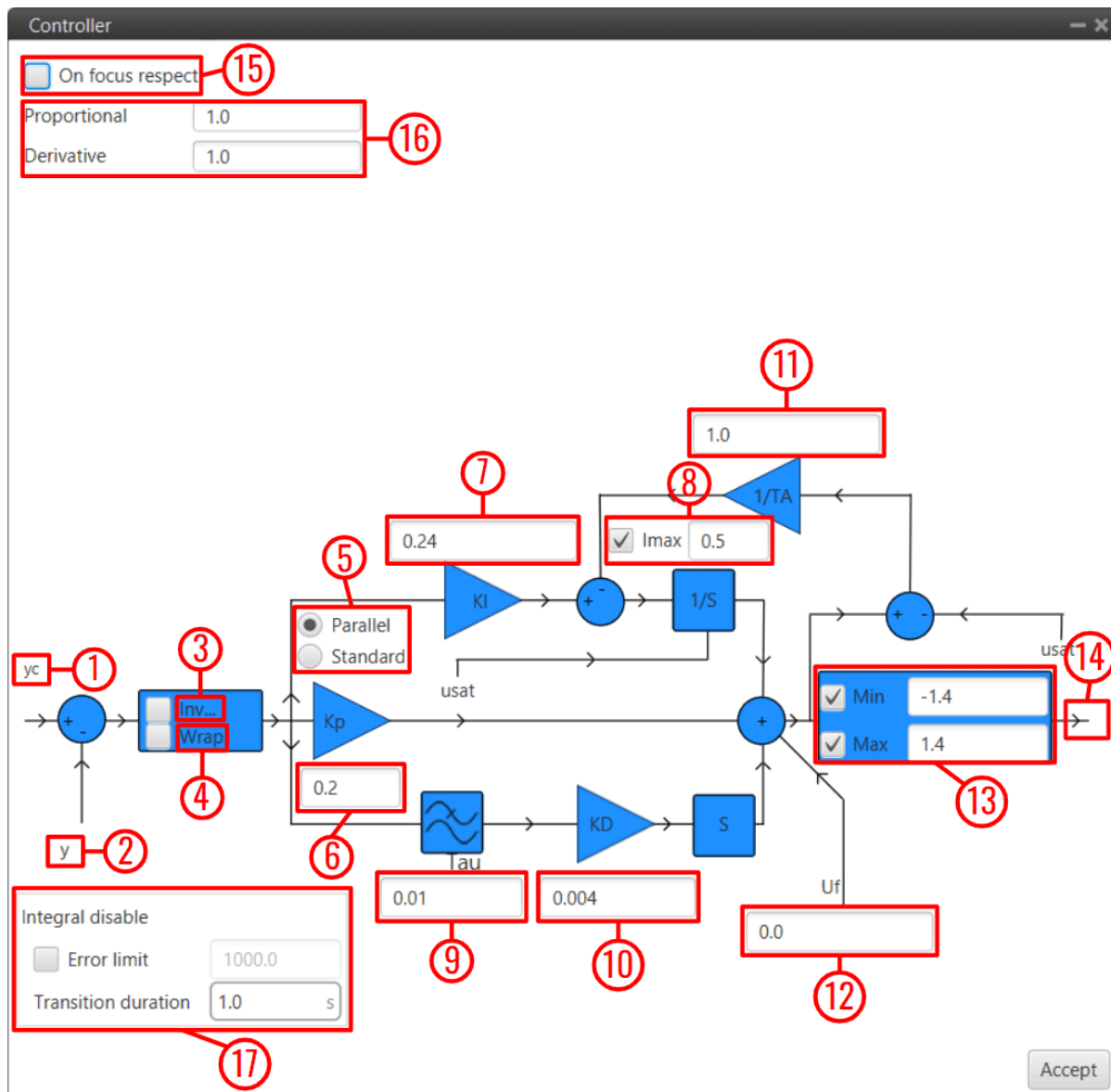


Fig. 219: PID block configuration

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 positive 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.
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 $(yc - y) > \text{Error limit}$.

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).

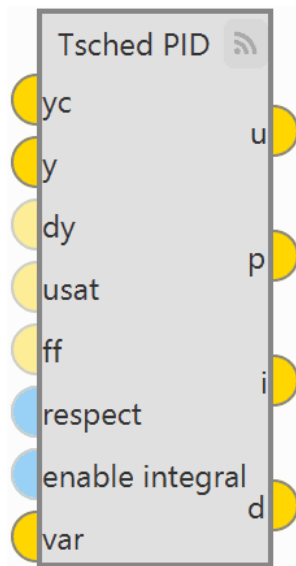


Fig. 220: **TSched PID** block

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.

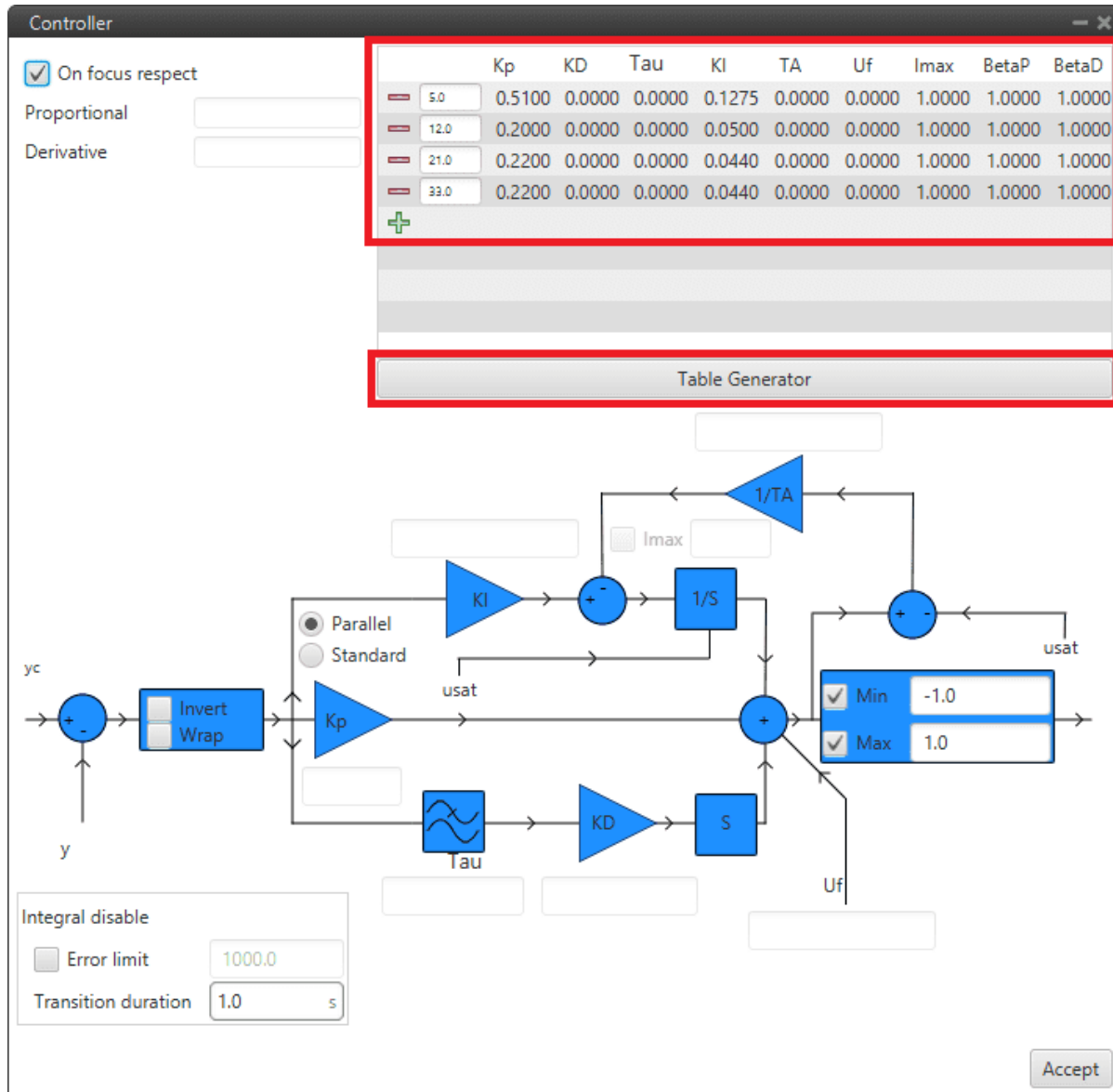


Fig. 221: TSched PID block configuration

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 the “+” icon and to remove them, click on the “-” icon.

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. 222: 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.

	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 row:** 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. 223: Example of Table Generator

2.9.1.3 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**.



Fig. 224: **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.

- **Configuration menu:**

Some parameters of the Energy algorithm can be modified by double clicking on the block:



Fig. 225: Energy Control block configuration

1. **Proportional gain for acceleration:** This is an indication of how aggressive 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. **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.
3. **K_STALL:** Stall correction coefficient. If 1, energy control is balanced for altitude and speed. If 0 only speed control is taken into account.
4. **IAS/V_STALL:** Speed/Stall ratio. Ratio between current speed and minimum speed.
5. **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 minimum speed selected in the *Envelope* section.

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. 226: **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**.

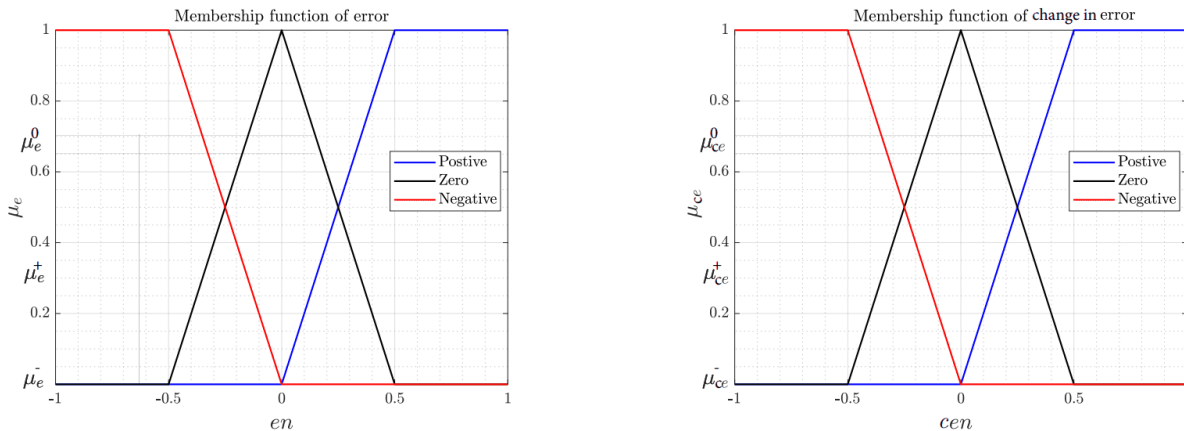


Fig. 227: **Example of membership functions of error and change in error**

- 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 arranged into a table, called the lookup table.

Δu		cen		
		+	0	-
en	+	+	+	0
	0	+	0	-
	-	0	-	-

Fig. 228: 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 .

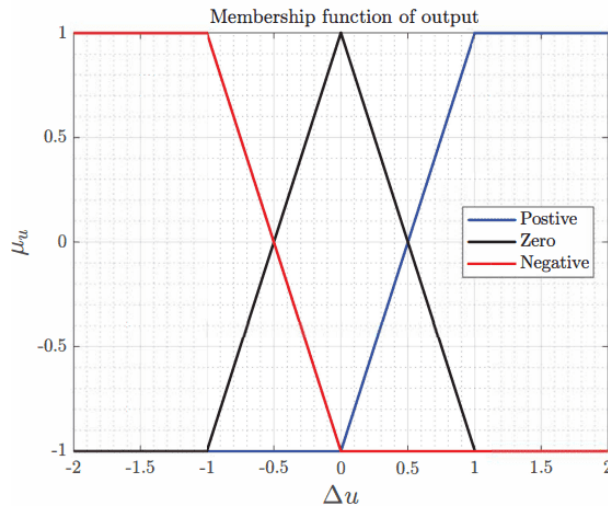


Fig. 229: Example of membership function of output

- 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:**

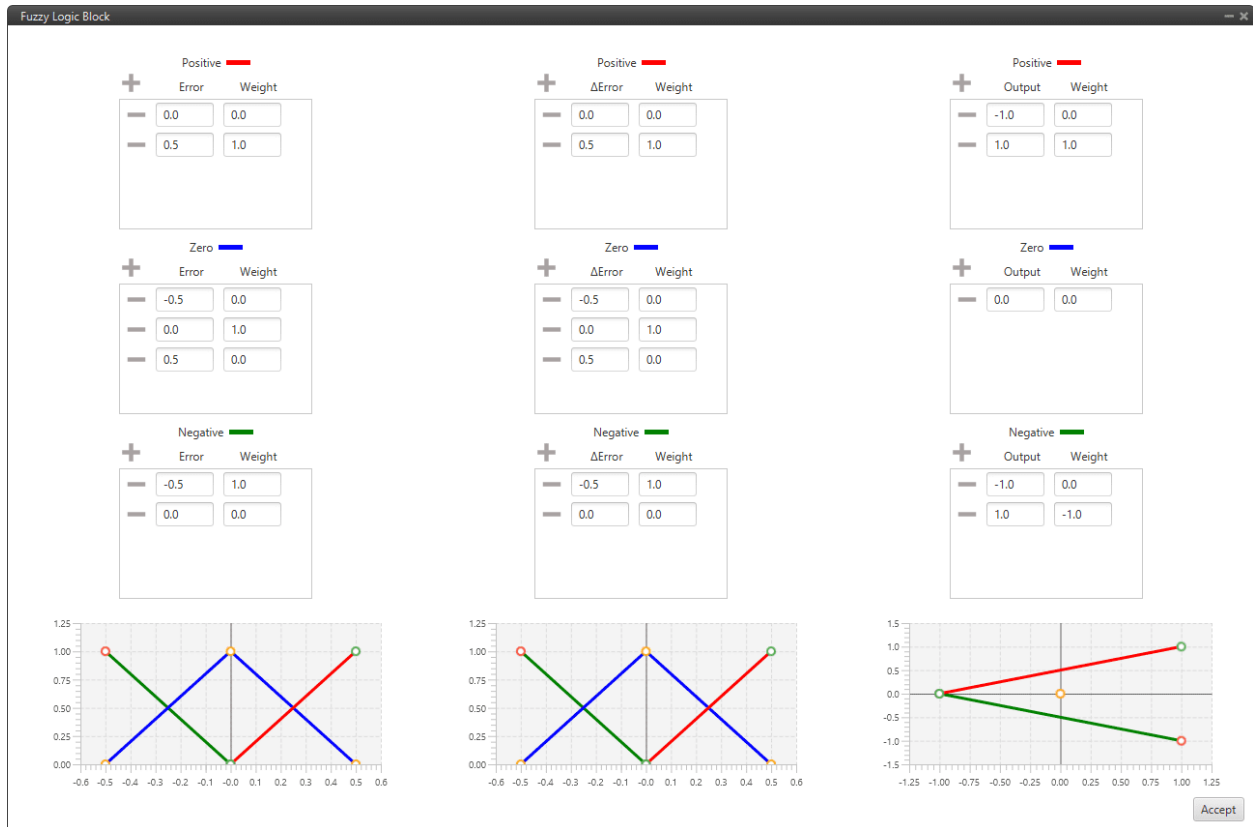


Fig. 230: Fuzzy Logic Controller block configuration

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 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.

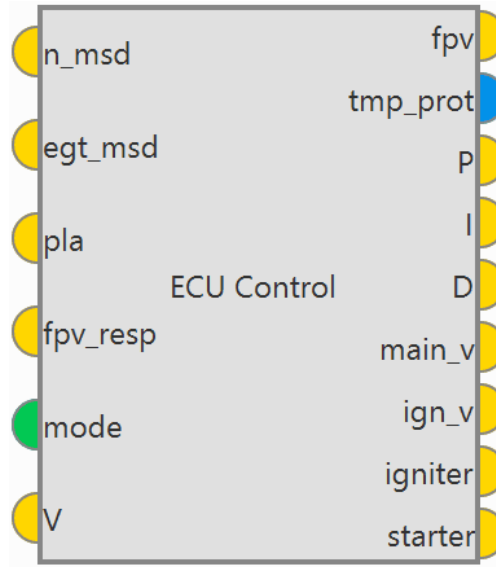


Fig. 231: ECU Control block

- 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.

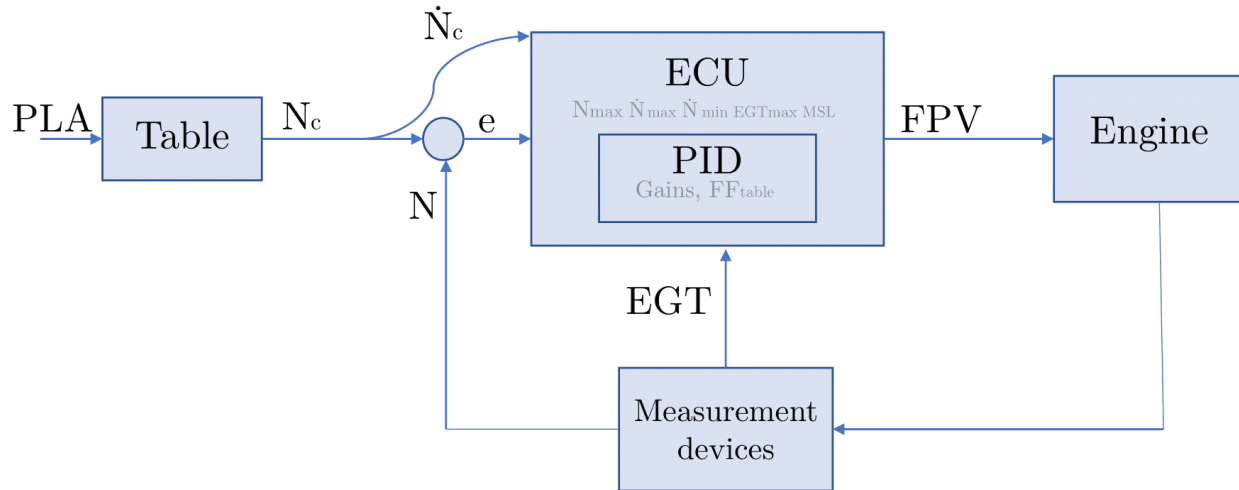


Fig. 232: ECU Control algorithm

• 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 \Rightarrow Off: all variables set to zero.
 - 1 \Rightarrow Checking: starter engine is commanded to max to check if engine is okay.
 - 2 \Rightarrow Starting: pilot has total control with PLA once engine is running.
- **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:

ECU Control

Maximum speed: 96000.0 rpm

Maximum acceleration: 50000.0 rpm/s

Maximum deceleration: -50000.0 rpm/s

Maximum Exit Gas Temperature: 1000.0000 °C

Start configuration

Maximum Voltage to valves: 15.600000 V

Maximum Voltage to igniter: 9.0 V

Maximum Voltage to starter: 12.0 V

Minimum Voltage to checking sequence: 12.0 V

Minimum Voltage to starting sequence: 14.800000 V

Fuel Pump Voltage to calibration: 1.0 V

Minimum EGT to ignition: 70.0 °C

Initial Voltage to starter engine: 5.0 V

Igniter stops at this speed: 20000.0 rpm

Starter engine stops at this speed: 29999.999 rpm

Minimum speed to check starting: 3000.0 rpm

Igniter pre-heat time: 5.0 s

Starter ramp-up to reach 100%: 2.0 s

Calibration time to reach calibration speed: 10.0 s

PID configuration

N	K_p	K_d	τ	K_I	τ_A	u_f	I_{max}	β_P	β_D
31000.0	0.0100	0.0010	0.4000	0.0200	0.5000	0.0000	-1.0000	1.0000	1.0000

Integral term options

☐ Error limit 1000.0

Transition duration 1.0 s

PLA	RPM
0.0	31000.0
0.1	46000.0
0.2	57000.0
0.3	68000.0
0.4	75000.0
0.5	80000.0

MSL	u_f
0.0	0.0

Apply

Fig. 233: 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 throttle 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.

2.9.1.6 Quaternion Control

Quaternion Control block for fixed multirotor aircraft.

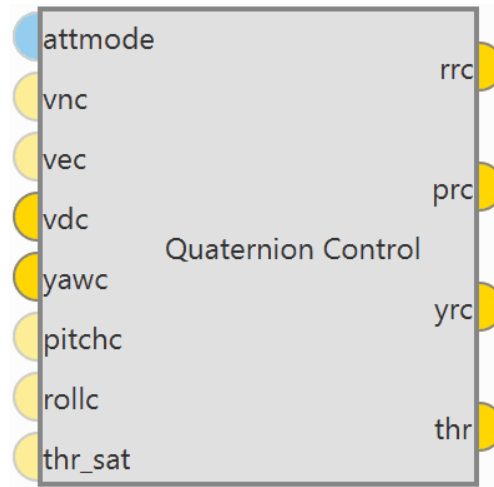


Fig. 234: **Quaternion Control block**

- 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.
- **yrp**: Desired yaw rate.
- **thr**: Desired thrust.

- **Configuration menu:**

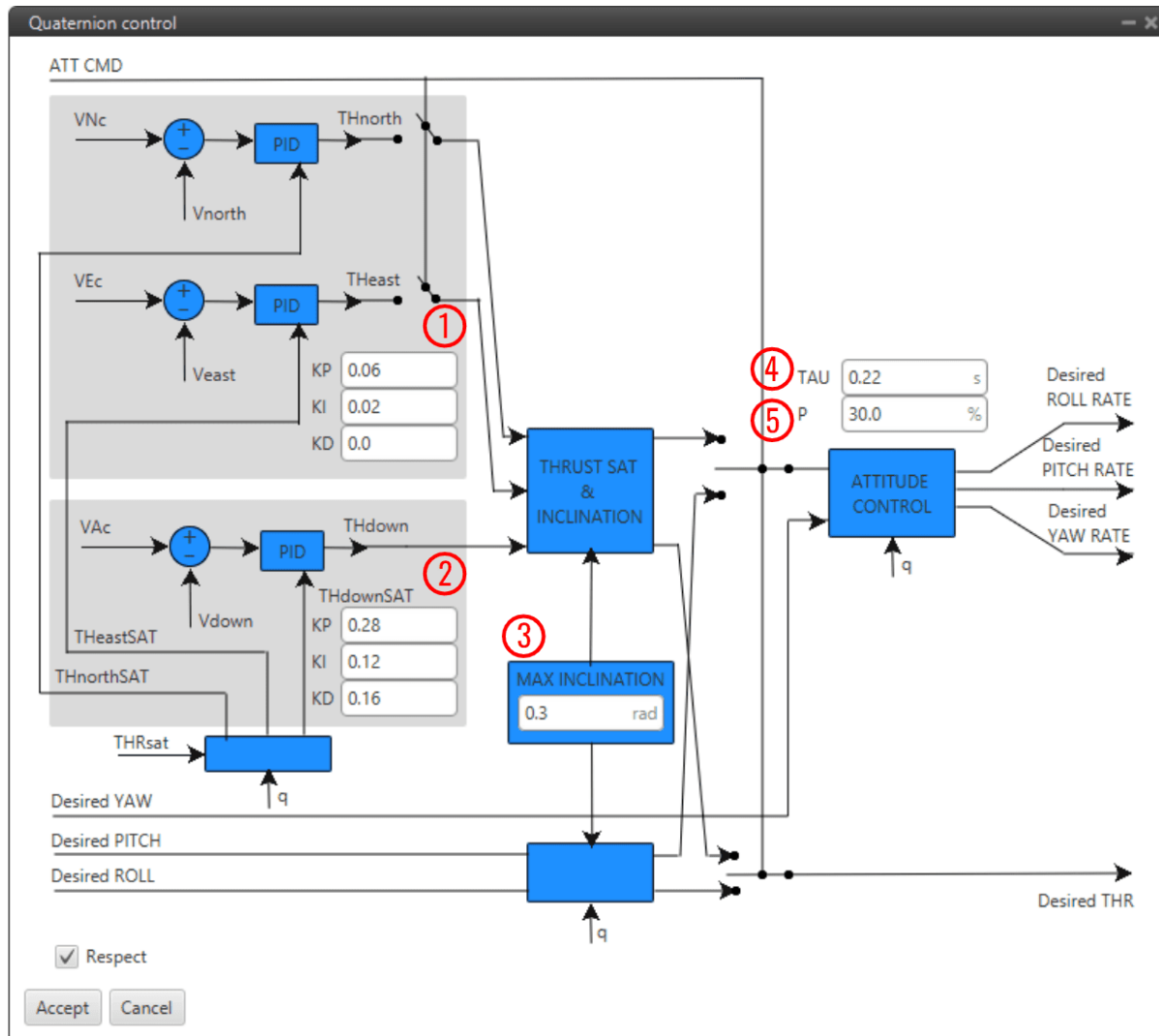


Fig. 235: Quaternion Control block configuration

As this block is mainly constituted by 3 PID controllers (for more information, 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%.

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.

2.9.1.7 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 **Adaptive-Predictive control algorithm**.

An example of use is shown below:

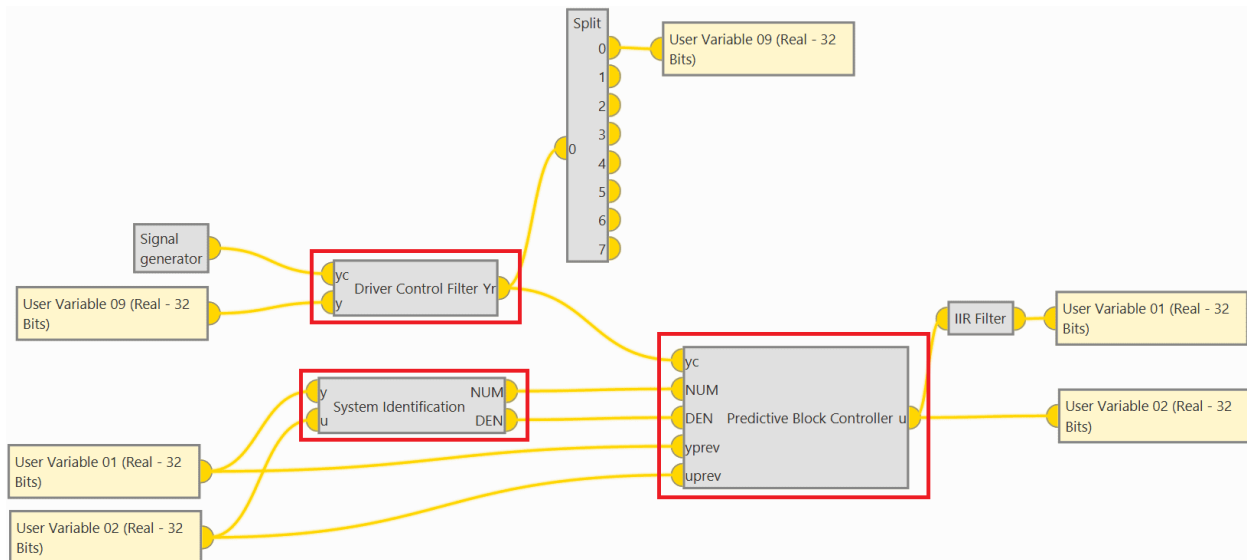


Fig. 236: Adaptive-Predictive control blocks example

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**:

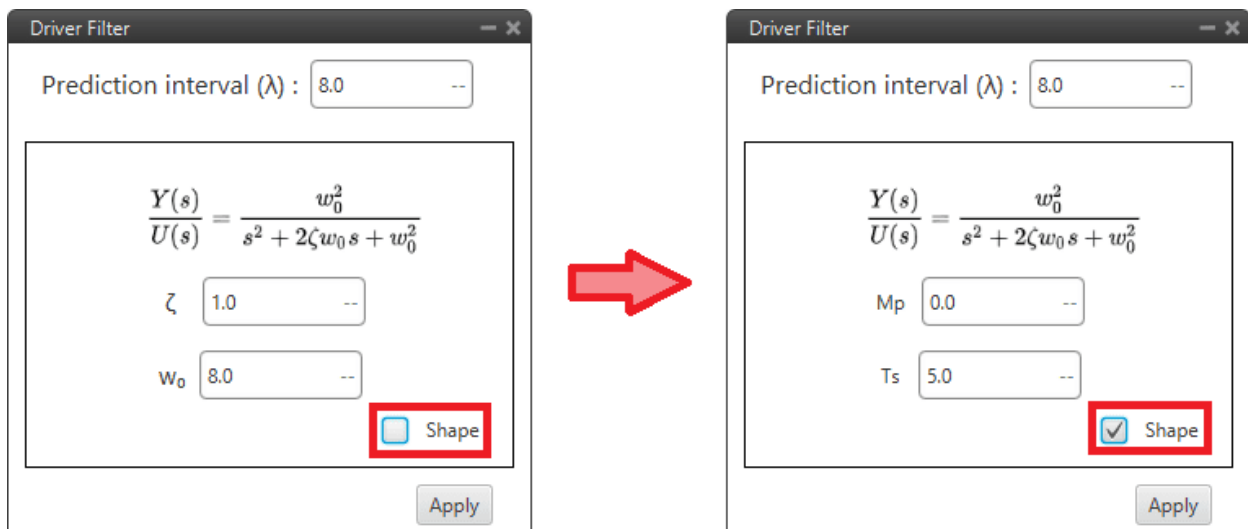
- **IIR Filter** simulates how the output responds to the input.
- **Signal generator** simply simulates a desired input function.

Driver Control Filter block gives a vector with variables set points (SP) using a second order filter.

Fig. 237: **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. 238: **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 parameters to be entered are different. Users can enable or disable it depending on the data available to them:
 - * ζ : Damping ratio.
 - * w_0 : Natural frequency.
 - * **Mp**: Maximum overshoot.
 - * **Ts**: Settling time (2% criteria).

Note: The relationships between the different parameters are:

$$Mp = \exp \left(-\zeta \cdot \frac{100 \cdot \pi}{\sqrt{1 - \zeta^2}} \right)$$

$$Ts = \frac{4}{\zeta \cdot w_0}$$

2.9.1.8 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 .

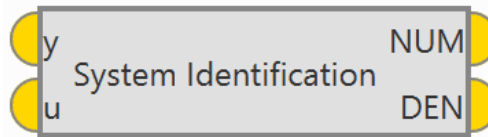
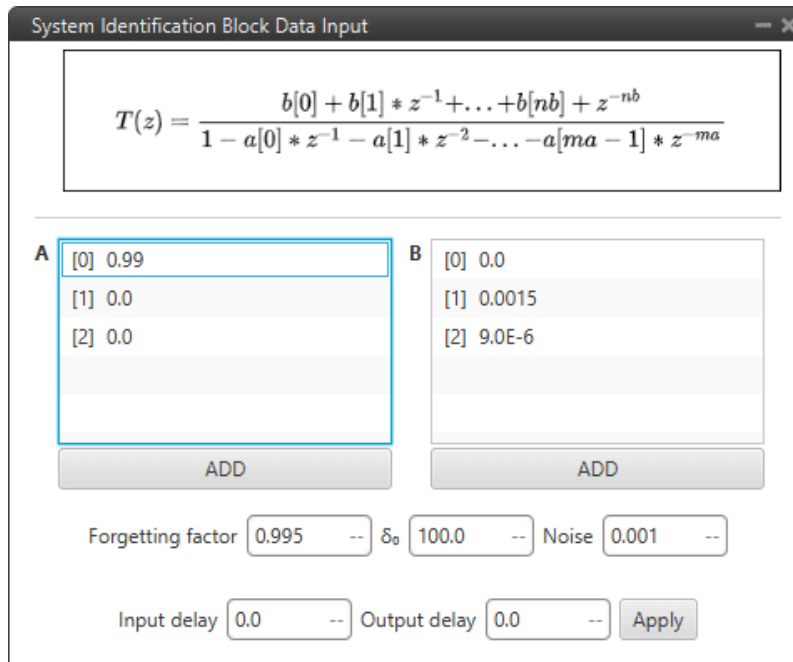


Fig. 239: **System Identification** block

This block works together with the *Driver Control Filter* and the *Predictive Control* blocks for the *Adaptive-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:**



System Identification Block Data Input

$$T(z) = \frac{b[0] + b[1] * z^{-1} + \dots + b[nb] * z^{-nb}}{1 - a[0] * z^{-1} - a[1] * z^{-2} - \dots - a[ma - 1] * z^{-ma}}$$

A

[0]	0.99
[1]	0.0
[2]	0.0

B

[0]	0.0
[1]	0.0015
[2]	9.0E-6

ADD

Forgetting factor -- δ_0 -- Noise --

Input delay -- Output delay --

Fig. 240: 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/Output delay**: Given the input/output size, a delay is applied to the sample vector size.

2.9.1.9 Predictive Control Block

Predictive Block Controller gives the optimal control output given a dynamic model as a result of the system identification block.

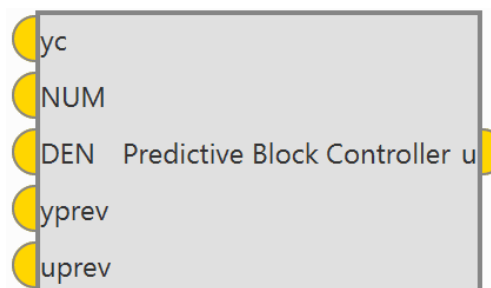


Fig. 241: Predictive Control block

This blocks works together with the *Driver Control Filter* and the *System Identification* blocks for the *Adaptive-Predictive control algorithm*, as explained above.

- **Inputs**

- **yc**: Desired system output (SP) or trajectory.
- **NUM**: Numerator coefficients of system model plant.
- **DEN**: Denominator coefficients of system model plant.
- **yprev**: Previous system output.
- **uprev**: Previous system input.

- **Output**

- **u**: Control output.

- **Configuration menu:**

Model Predictive Control

Prediction interval (λ) : 8.0 --

num size 3.0 -- den size 3.0 --

PREDICTION ☒ Optimal

$$Q(k, k) = Q(k-1, k-1) * f, \quad k = 2, \lambda$$

$$R(k, k) = R(k-1, k-1) * f,$$

$$Q(1, 1) = 1 \quad R(1, 1) = r_0$$

f 100.0 --

r_0 100.0 --

OUTPUT LIMITS

☐ Min 0.0 -- ☐ Max 1.0 --

Apply

Fig. 242: 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.
 - * **r_0** : If this factor is greater than 1, there is a more aggressive 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).

2.9.2 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 to **overwrite any variable** in the system. **Variables that can be written** using Sink blocks are:
 - User Variables
 - Desired variables (variables whose name starts with ‘Desired’)

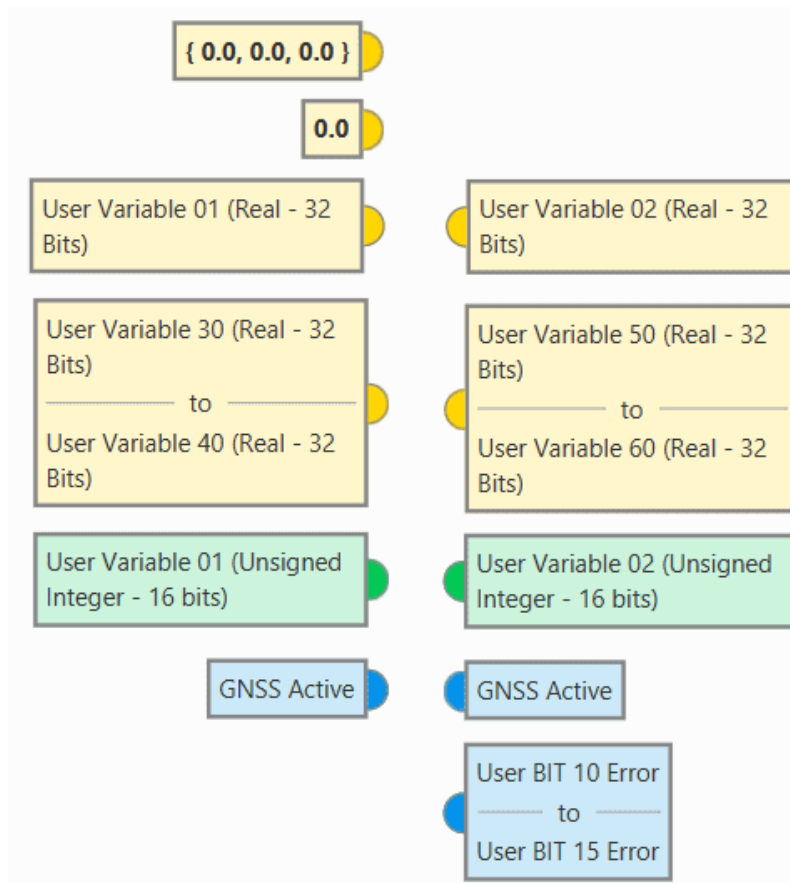


Fig. 243: Source/Sink blocks

Warning:

- **Desired variables** are naturally written by **Guidances** algorithms. If an active **Guidance** is **writing** a certain Desired variable, writing 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 1x autopilot 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 1x autopilot 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. 244: **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:**

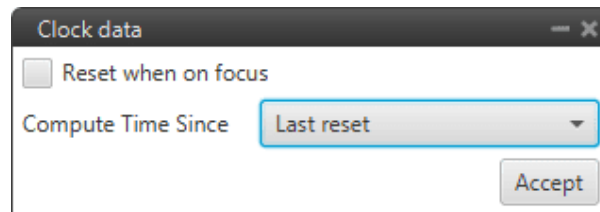


Fig. 245: **Clock block configuration**

- **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.

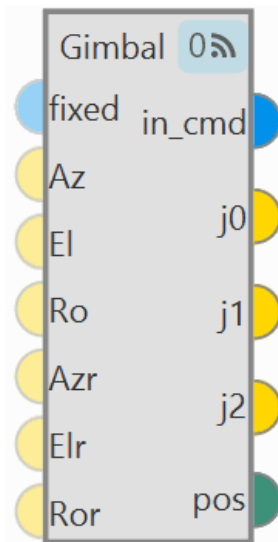


Fig. 246: Gimbal block

• 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**:

The screenshot shows a configuration window for a Gimbal block. The 'Type' dropdown is set to 'Pan Tilt'. The 'Logic' dropdown is set to 'Conventional gim...'. There is an 'Edit Rotation Matrix' button. The 'Base to gimbal X', 'Base to gimbal Y', and 'Base to gimbal Z' input fields are all set to 0.0. The 'Roll rate Gain' and 'Pitch rate Gain' input fields are both set to 0.0. The 'Center Limit Yaw [0, 2pi]', 'Center Limit Pitch', and 'Center Limit Roll' input fields are all set to 0.0. The 'Delta Limit Yaw (0, pi]', 'Delta Limit Pitch', and 'Delta Limit Roll' input fields are all set to 6.2831855.

Fig. 247: 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.

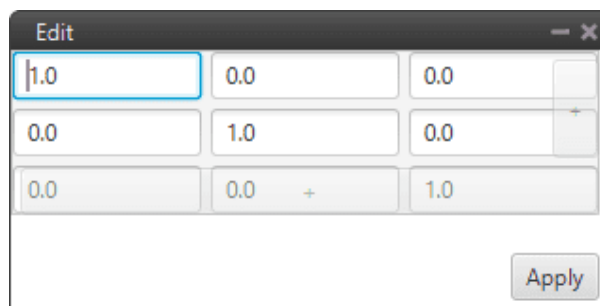


Fig. 248: Gimbal block configuration - Rotation Matrix

- **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:

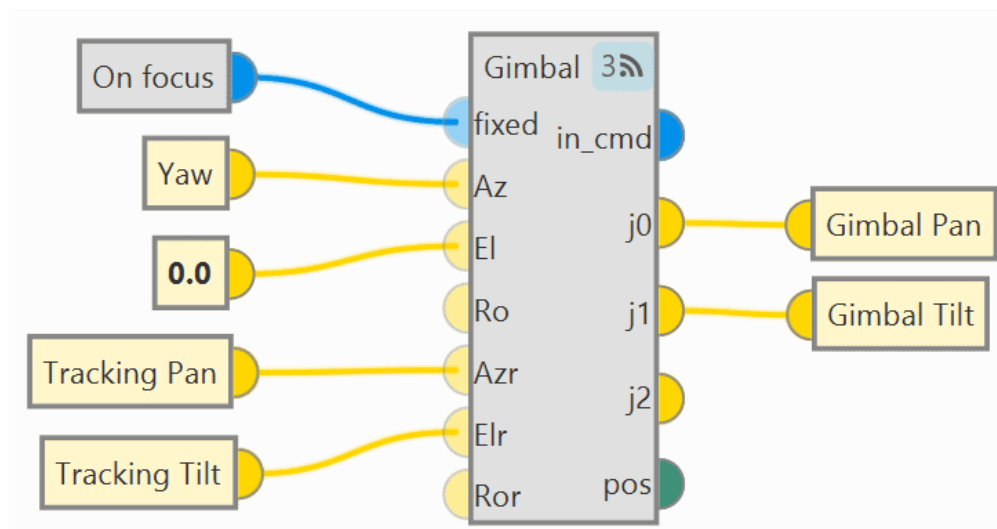


Fig. 249: Gimbal block - Example of use

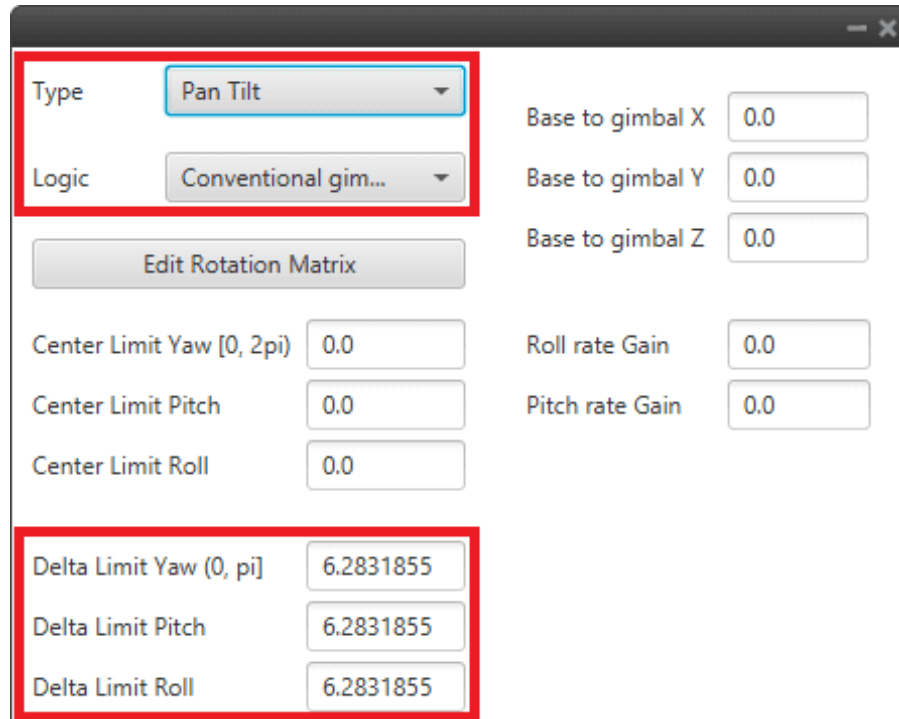


Fig. 250: Gimbal block - Configuration example

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.

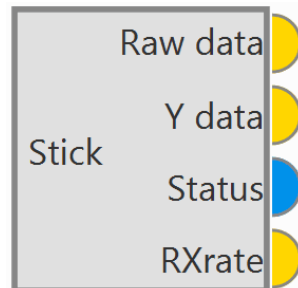


Fig. 251: Stick block

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**.

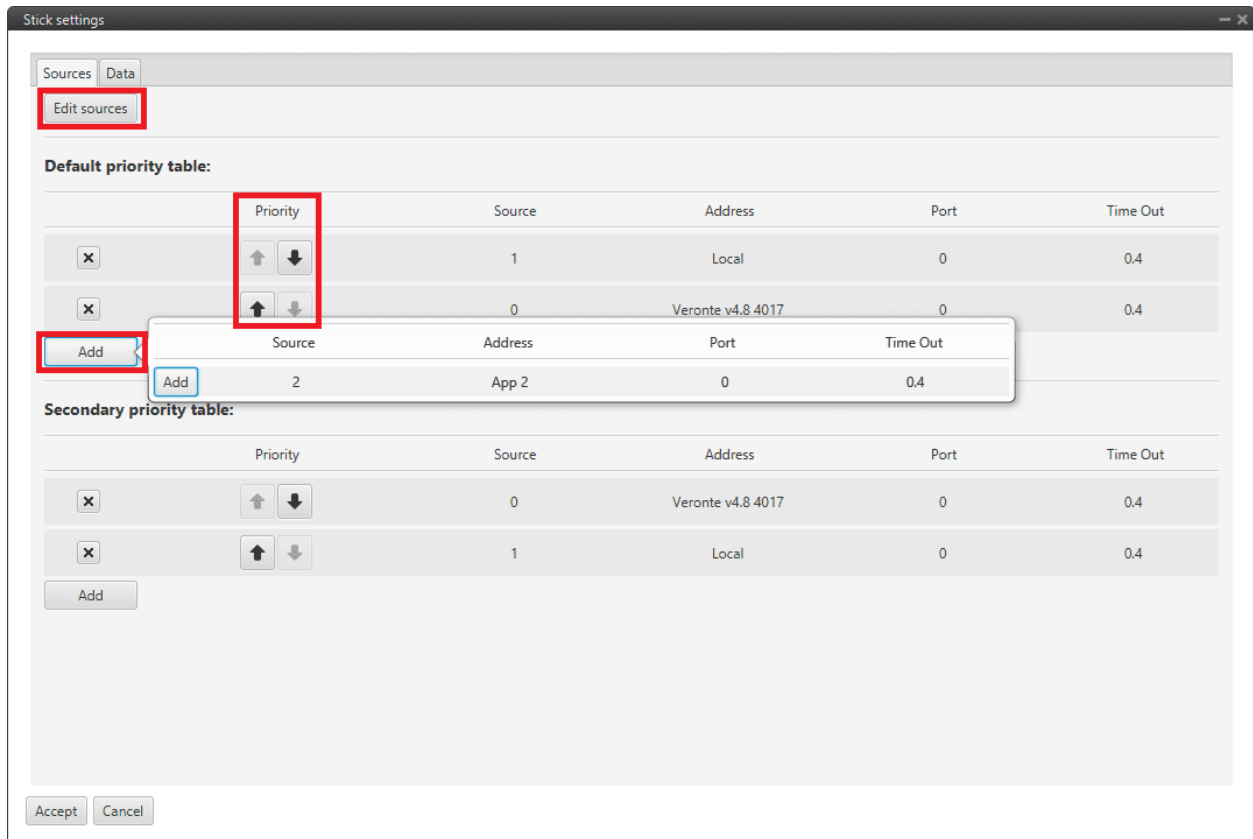


Fig. 252: Stick block configuration - Sources

- * **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.

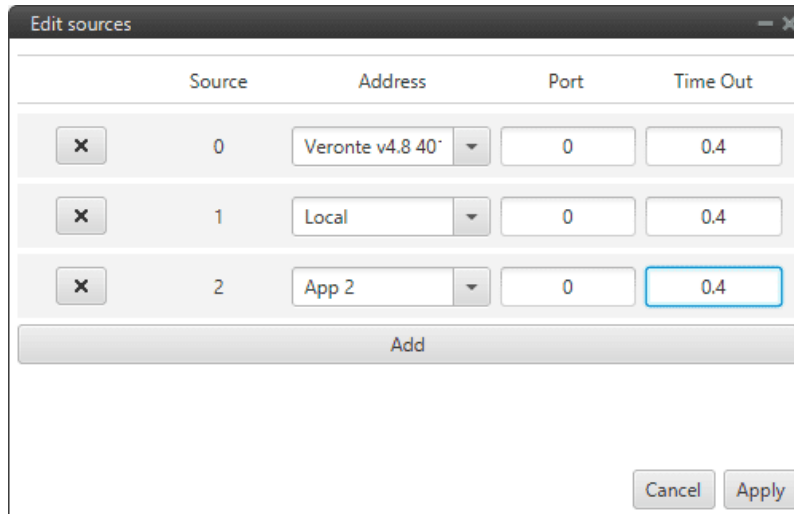


Fig. 253: Edit sources menu

- **Source:** It is the order in which sources are created in this menu. This **does not set the priority**.
 - **Address:** This defines the source of where the stick information is taken from. The following options are available:
 - App 2: Means that the information is coming from the **virtual stick** 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.
 - Veronte v4.X XXXX: Means the information is coming from a **particular autopilot**, which needs to be visible in **Veronte Link**.
 - **Port:** From each source it is possible 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 $Output = (Mix\ Matrix)(Raw\ channels) + Offset$, 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.

Stick settings

Sources Data

OUTPUT = MIX MATRIX * RAW CHANNELS + OFFSET

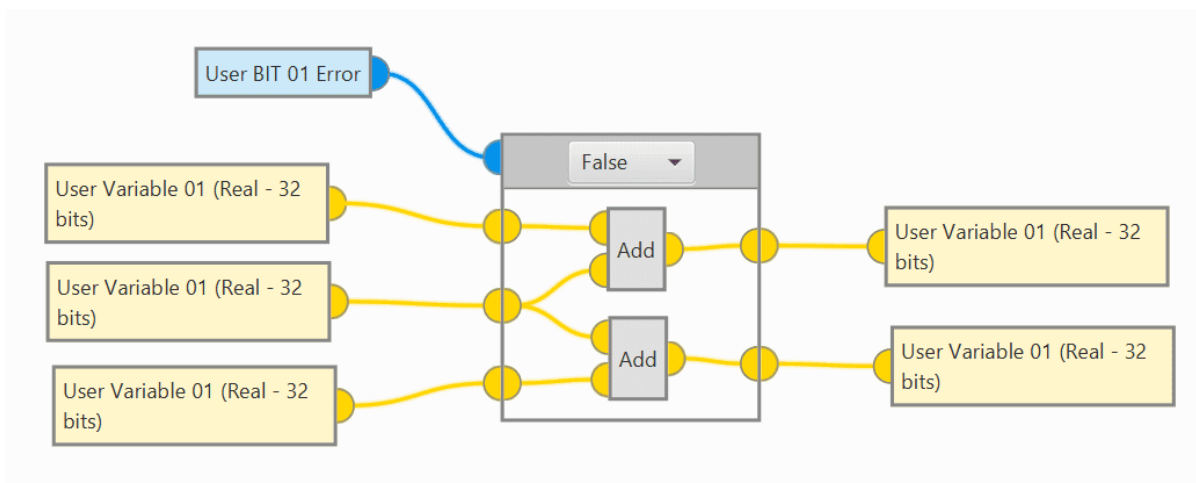
OUTPUT		MIX MATRIX					OFFSET	
		Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
+	1	-0.354	-0.354	1	0.25	0	0	0
-	2	-0.354	0.354	1	-0.25	0	0	0
-	3	0.354	0.354	1	0.25	0	0	0
-	4	0.354	-0.354	1	-0.25	0	0	0

Accept Cancel

Fig. 254: 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.



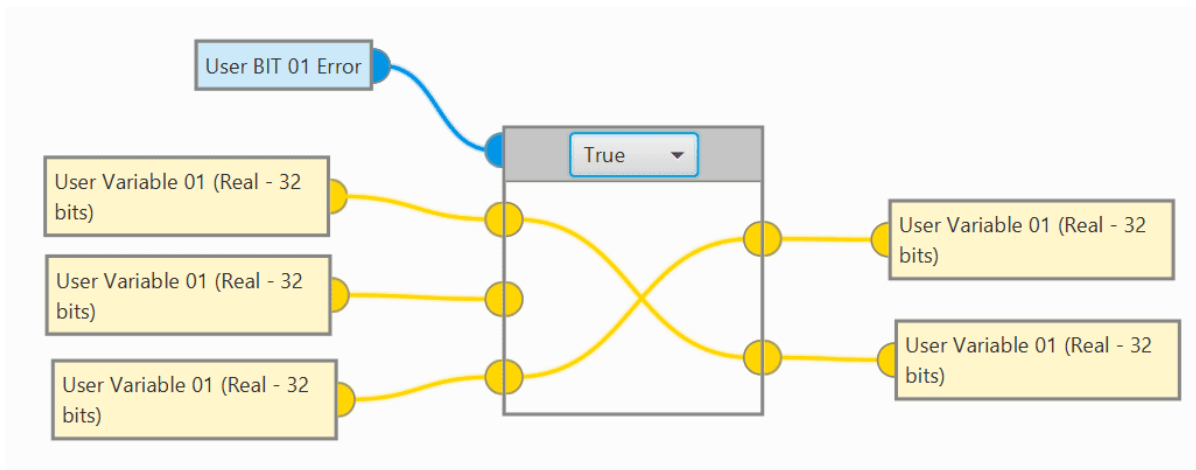


Fig. 255: Execution flow blocks

Execution flow blocks are divided into 2 different types of blocks:

- **On focus block:** The On Focus block outputs a boolean value, which is **only True the first time** the block is executed.

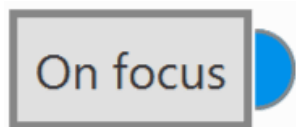


Fig. 256: On focus block

- 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. 257: On focus block example

• Switch blocks

- **If-Else Switch block:** Choose between two cases based on the state of a boolean variable.

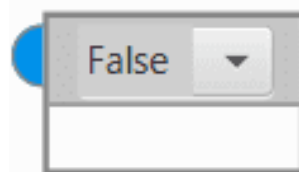
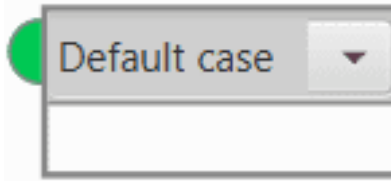


Fig. 258: If/Else switch block

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**'.

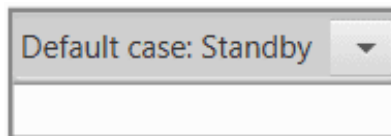
Fig. 259: **Invert created blocks**

- **Integer Switch block:** Choose a case based on the value of an integer variable.

Fig. 260: **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. 261: **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:** **Right-click** inside a Switch block to insert a new block into it.

Fig. 262: **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. 263: 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
<i>Climb</i>	Makes the aircraft climb from the start of the phase to another altitude.
<i>Cruise</i>	Makes the aircraft follow a determined route created by the user.
<i>Landing</i>	Creates the route that the airplane will follow to land.
<i>Rendezvous</i>	Used to create a meeting point where the Air unit will approach a second unit (either Air or Ground) within a determined offset.
<i>Taxi</i>	Creates a linear path along the runway that is followed by the aircraft.
<i>VTOL</i>	Vertical take-off and landing.
<i>Yaw current/heading/north</i>	Indicates the behaviour 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**.

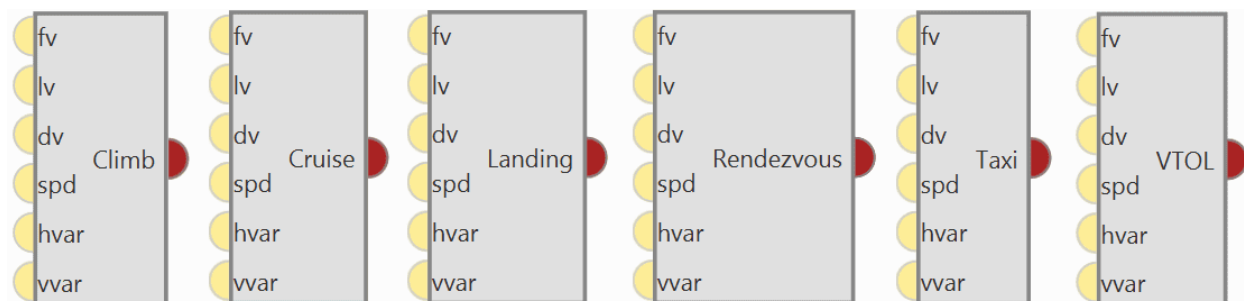


Fig. 264: Common guidance blocks

- **Inputs**

- (Optional) **fv**: **F**irst 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 *Guidance Computation* block via this output.

- **Configuration menu:**

Fig. 265: 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 be 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 is selected, the platform will follow the predefined 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**. **1x autopilot 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 be 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 1x autopilot 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**.

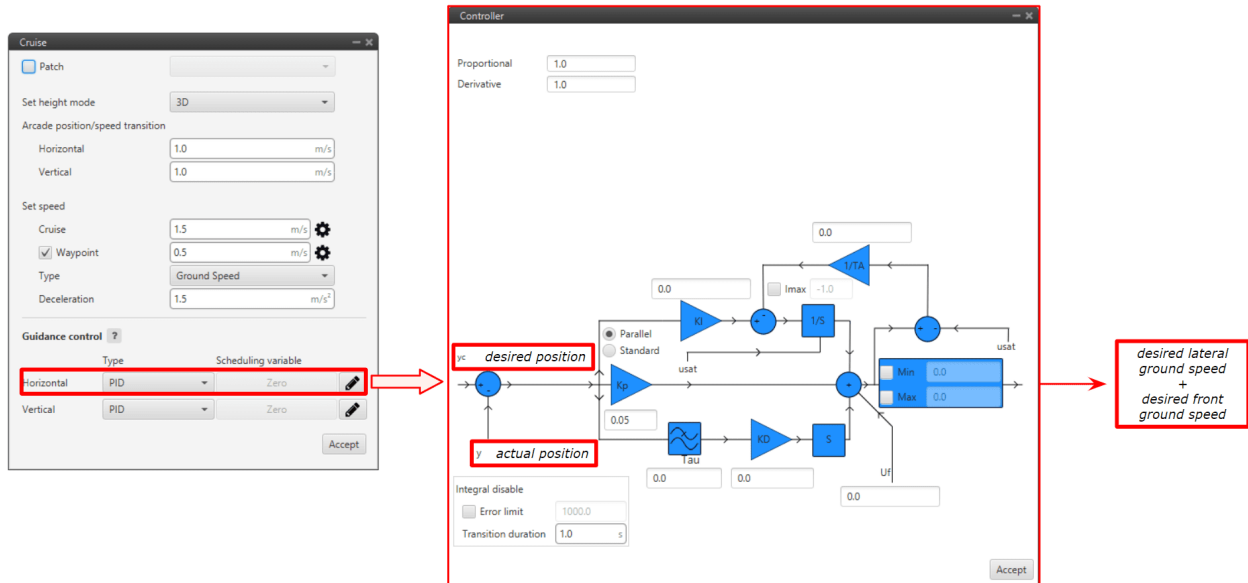
4. **Set speed:** This option sets the speed that the vehicle will have during the manoeuvre.
 - **Cruise:** Lets the user set the velocity modulus of the guidance. This velocity can be slightly modified by the autopilot control algorithms.
 - **Waypoint:** If enable, it indicates the speed at which 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.

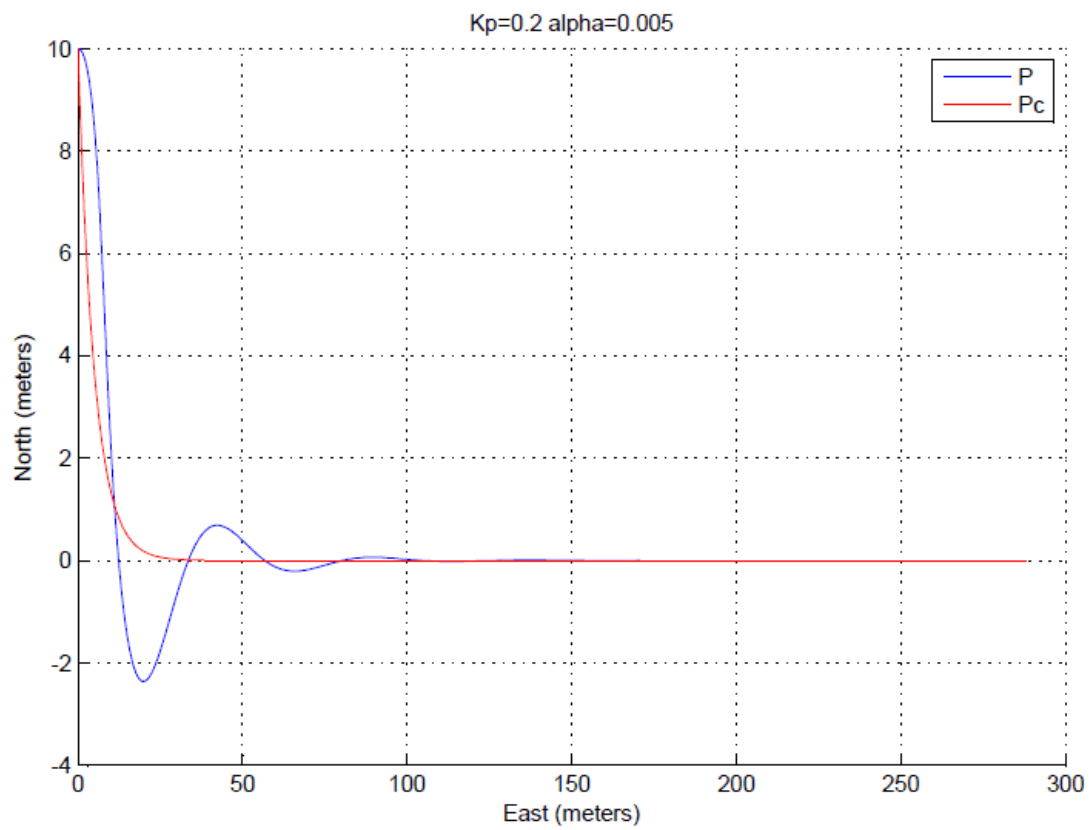
Fig. 266: **Horizontal Position PID**

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 behaviour with $K_p = 0.2$ and $K_p = 0.02$.



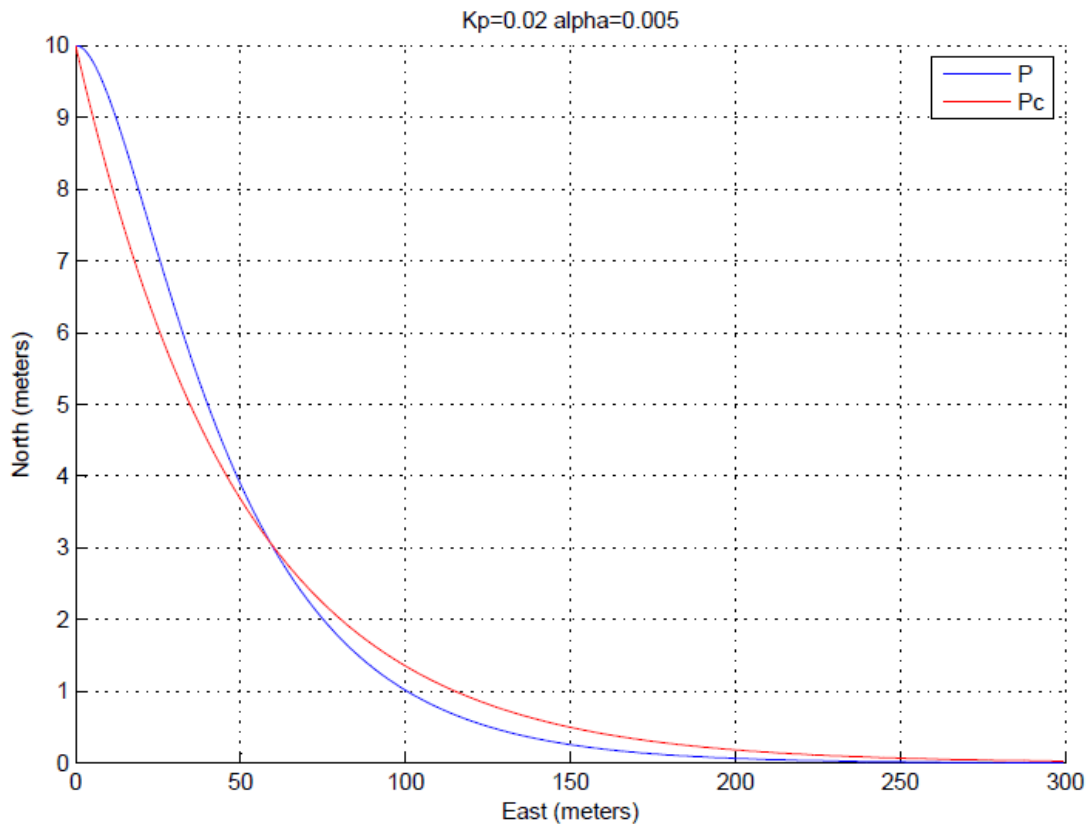


Fig. 267: PID Proportional gains

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.

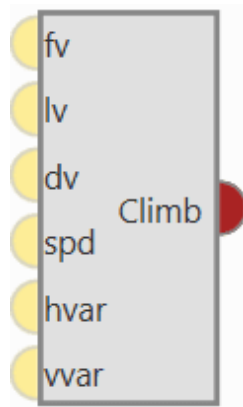


Fig. 268: **Climb block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidance 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

Climb

☐ Patch

Set height mode: 3D

Arcade position/speed transition

Horizontal: 1.0 m/s

Vertical: 1.0 m/s

Set speed

Cruise: 5.0 m/s

☐ Waypoint: 5.0 m/s

Type: Ground Speed

Deceleration: 5.0 m/s²

Guidance control ?

Type

Horizontal: PID

Vertical: PID

Runway: TKO runway

☐ Advanced

Loiter point: Runway Loiter

Direction: Ref: Runway Direction

☒ Loiter pos is center

Taxi extension: 1000.0 m

Horizontal distance (dxy): 1000.0 m

Radius head turn (R3): 1000.0 m

Radius loiter (R1): 1000.0 m

Flight path angle: 0.2 rad [-π,π]



Accept

Fig. 269: 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:

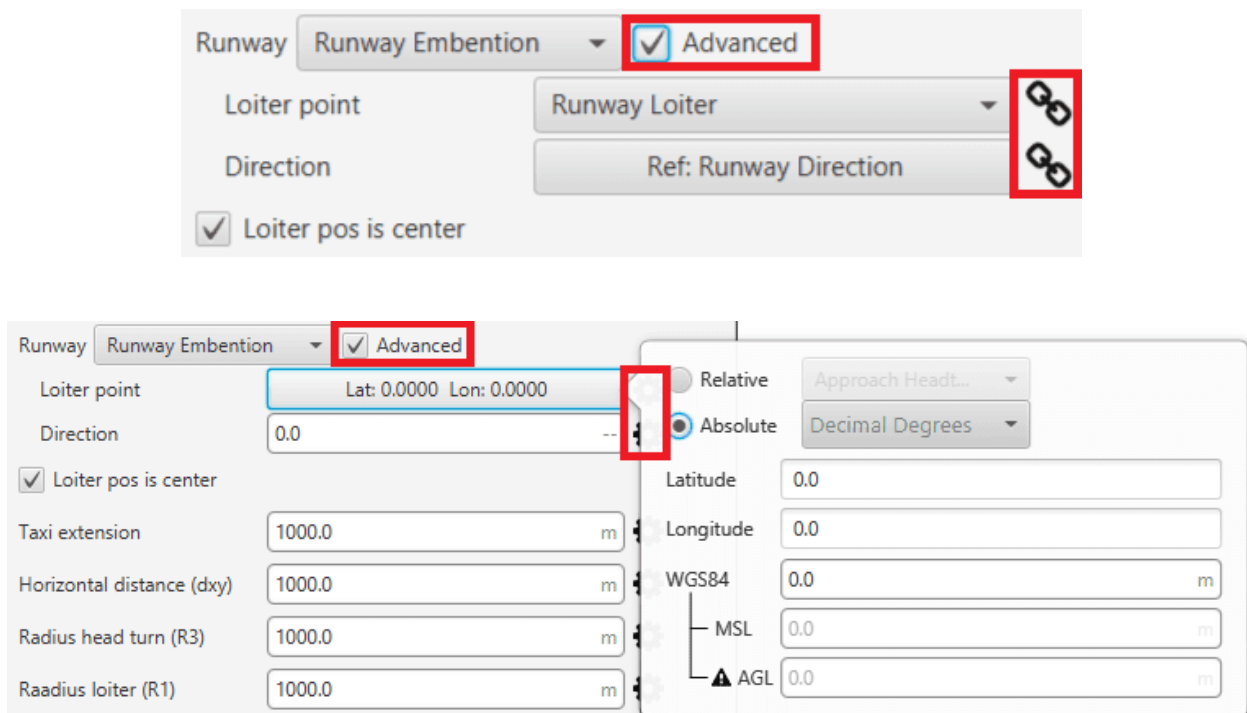






Fig. 270: Runway and Loiter Position Options

– **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 a 1x autopilot.
 - **Absolute:** The coordinates can be set in UTM, MGRS, Decimal Degrees or Degree ° ' ". 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.

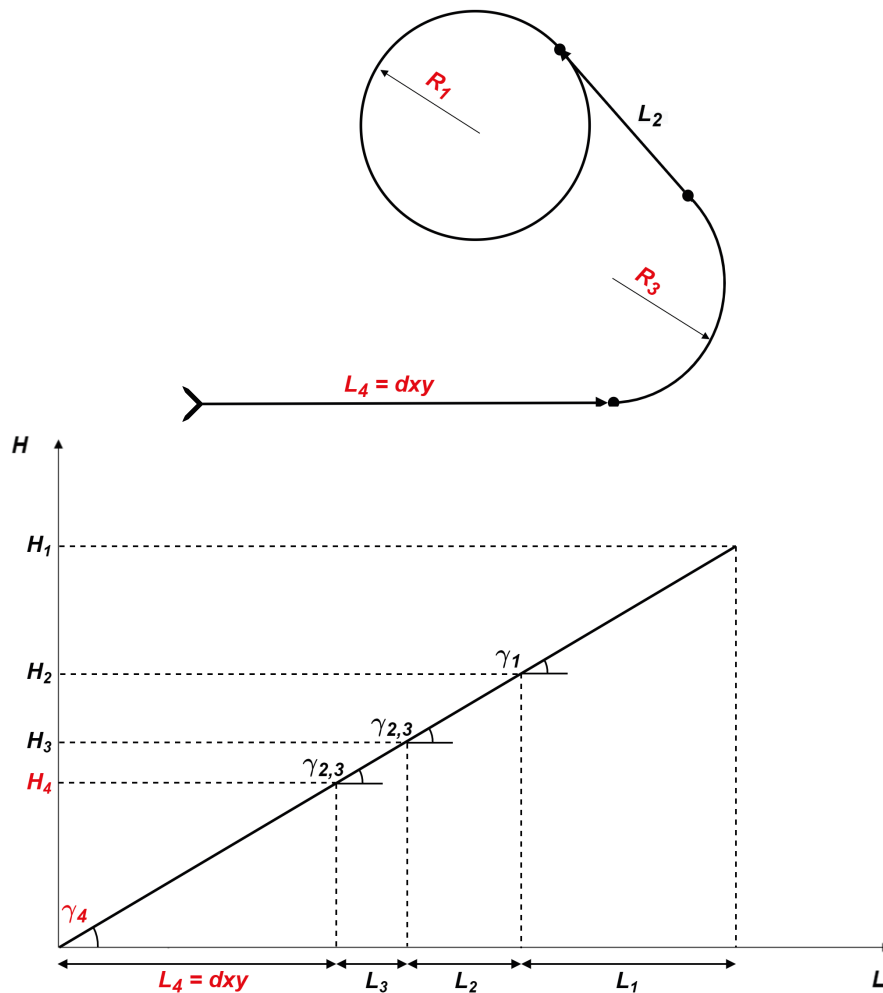


Fig. 271: **Climb route top and front views with parameter identification**

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 behaviour 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.**

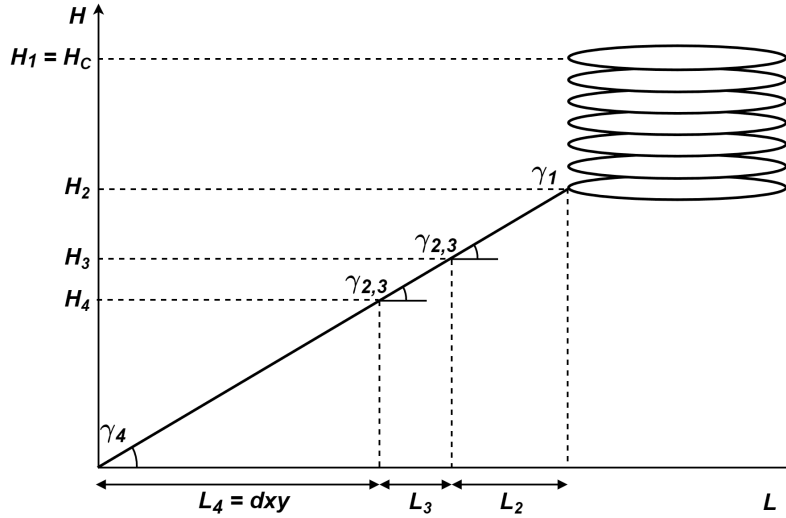
$$\gamma_4 = FPA$$

$$H_4 = dxy \tan(\gamma_4)$$

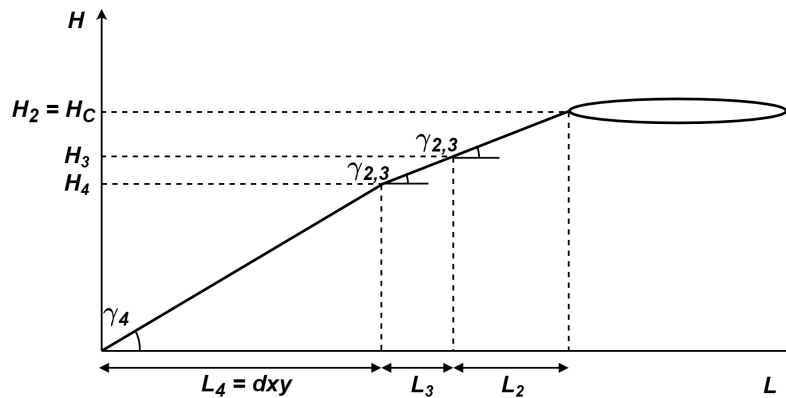
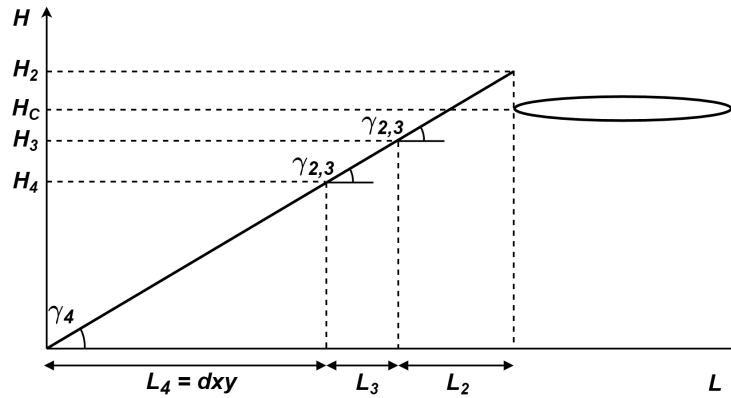
- L_3 : Circular turn to head the platform towards the loiter. The user can set the radius, R_3 . It is **possible to set this path to zero** clicking on   . The FPA, $\gamma_{2,3}$, can be modified by the algorithm.
- L_2 : Straight path that reaches the climbing loiter point. This path is **completely 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:

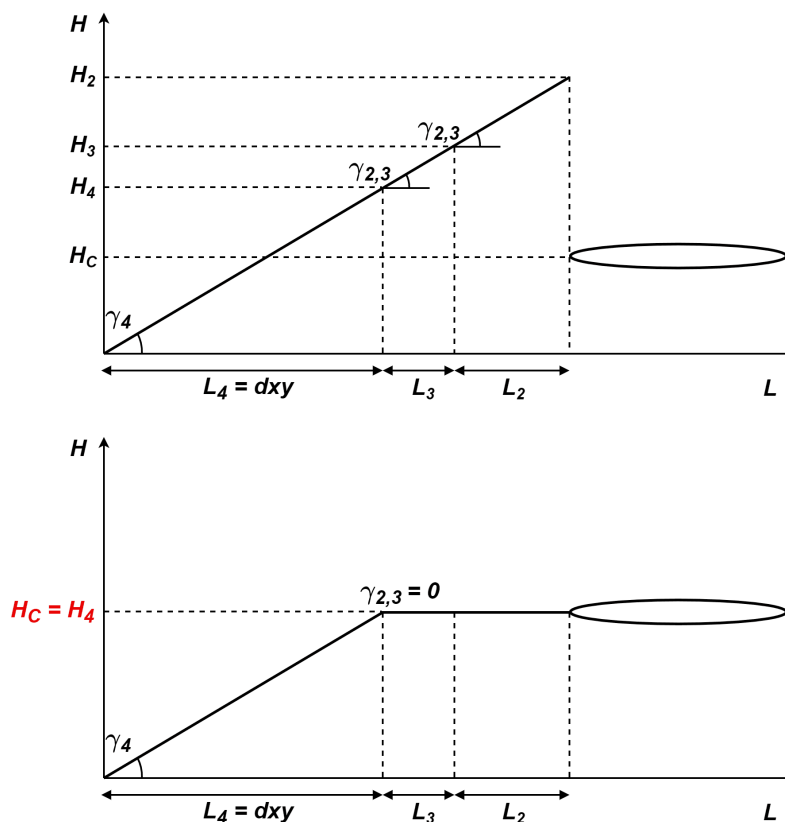
1. $H_2 < H_c$: This is the general case, in this situation no corrections will be applied as shown below and $\gamma_{2,3} = \gamma_1 = FPA$.

Fig. 272: 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. 273: 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. 274: 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.

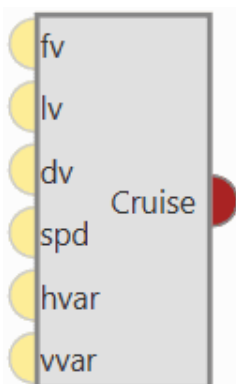


Fig. 275: Cruise block

Warning: In order to produce a guidance computation this block has to be connected to the *Guidance Computation* block.

- **Configuration menu:**

The screenshot shows the 'Cruise' configuration window with the following parameters:

- 1** Patch: ☐ Patch
- 2** Set height mode: 3D
- 3** Arcade position/speed transition:
 - Horizontal: 1.0 m/s
 - Vertical: 1.0 m/s
- 4** Set speed:
 - Cruise: 5.0 m/s
 - Waypoint: ☐ Waypoint 5.0 m/s
 - Type: Ground Speed
 - Deceleration: 5.0 m/s²
- 5** Guidance control:
 - Horizontal: PID
 - Vertical: PID

Accept

Fig. 276: 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 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*.

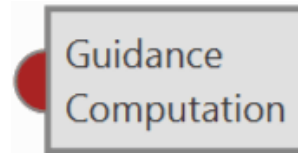


Fig. 277: Cruise block

- **Input**

- **Pin 0:** Guidance configuration.

2.9.5.5 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.

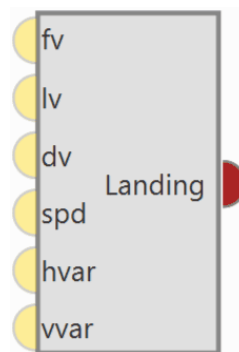


Fig. 278: Landing block

Warning: In order to produce a guidance computation this block has to be connected to the *Guidance 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 detailed 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.

The figure shows two views of the 'Landing' configuration window. The left view highlights several sections with red boxes and numbers 1 through 7. The right view highlights the 'Flight path angle' and 'Velocities' sections with red boxes and numbers 8 and 9 respectively.

Left Screenshot (Numbered sections):



- 1. Patch (checkbox)
- 2. Set height mode (3D)
- 3. Arcade position/speed transition (Horizontal: 1.0 m/s, Vertical: 1.0 m/s)
- 4. Set speed (Cruise: 5.0 m/s, Waypoint: 5.0 m/s, Type: Ground Speed, Deceleration: 5.0 m/s)
- 5. Guidance control (Type: PID, Horizontal: PID, Vertical: PID)
- 6. Runway (TKO runway), Advanced (checkbox), Touch point (Runway Touch Point), Loiter point (Runway Loiter), Direction (Ref: Runway Direction)
- 7. Loiter pos is center (checkbox), Taxi extension (1000.0 m), Horizontal distance (dxy) (1000.0 m), Radius head turn (R3) (1000.0 m), Radius loiter (R1) (1000.0 m)

Right Screenshot (Numbered sections):

- 8. Flight path angle (Initial maximum (absolute): 0.0 rad [- π , π], Loiter: 0.0 rad [- π , π], Aim: 0.0 rad [- π , π], DXY: 0.0 rad [- π , π])
- 9. Velocities (Initial: 5.0 m/s, Loiter: 5.0 m/s, Aim: 5.0 m/s, DXY: 5.0 m/s)

Fig. 279: 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:

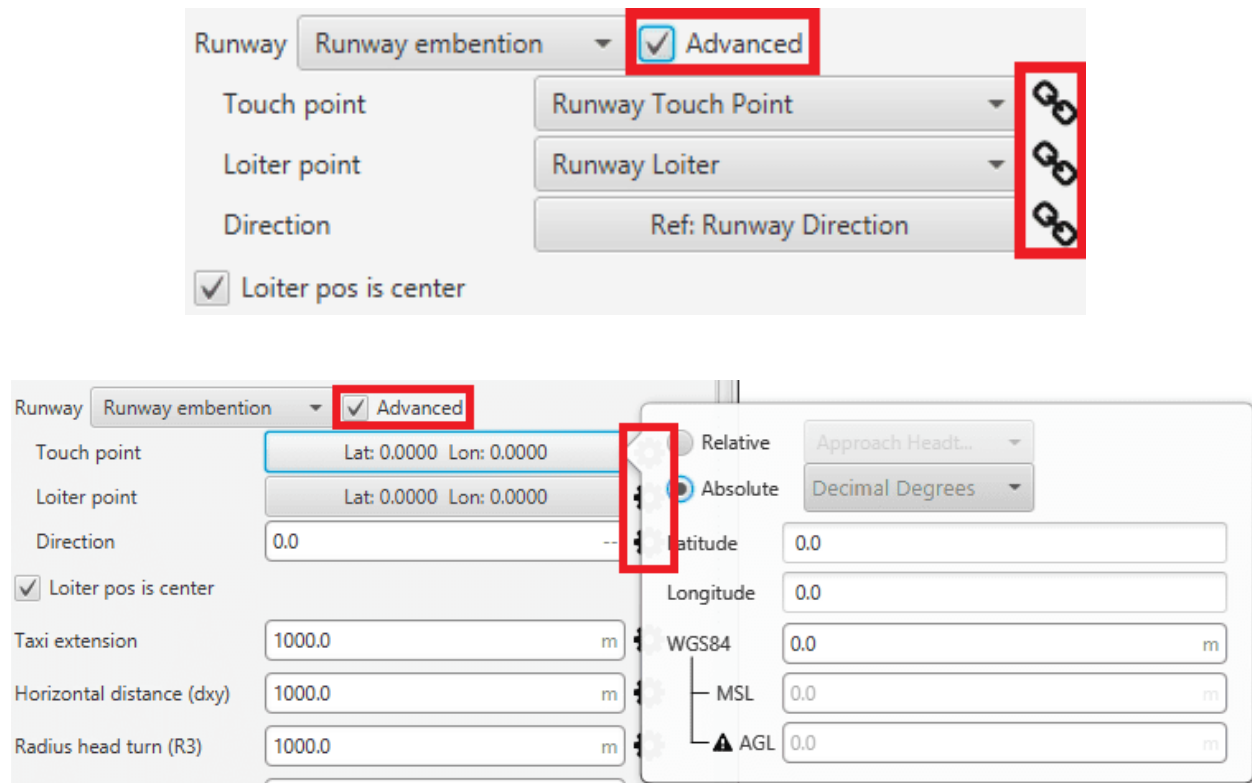










Fig. 280: Runway and Loiter Position Options



- **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 a 1x autopilot.
 - **Absolute:** The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress ° ' ' '. 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 a 1x autopilot.
 - **Absolute:** The coordinates can be set in UTM, MGRS, Decimal Degrees or Degress ° ’ ’’. 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 brought 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 than 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 connect 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.

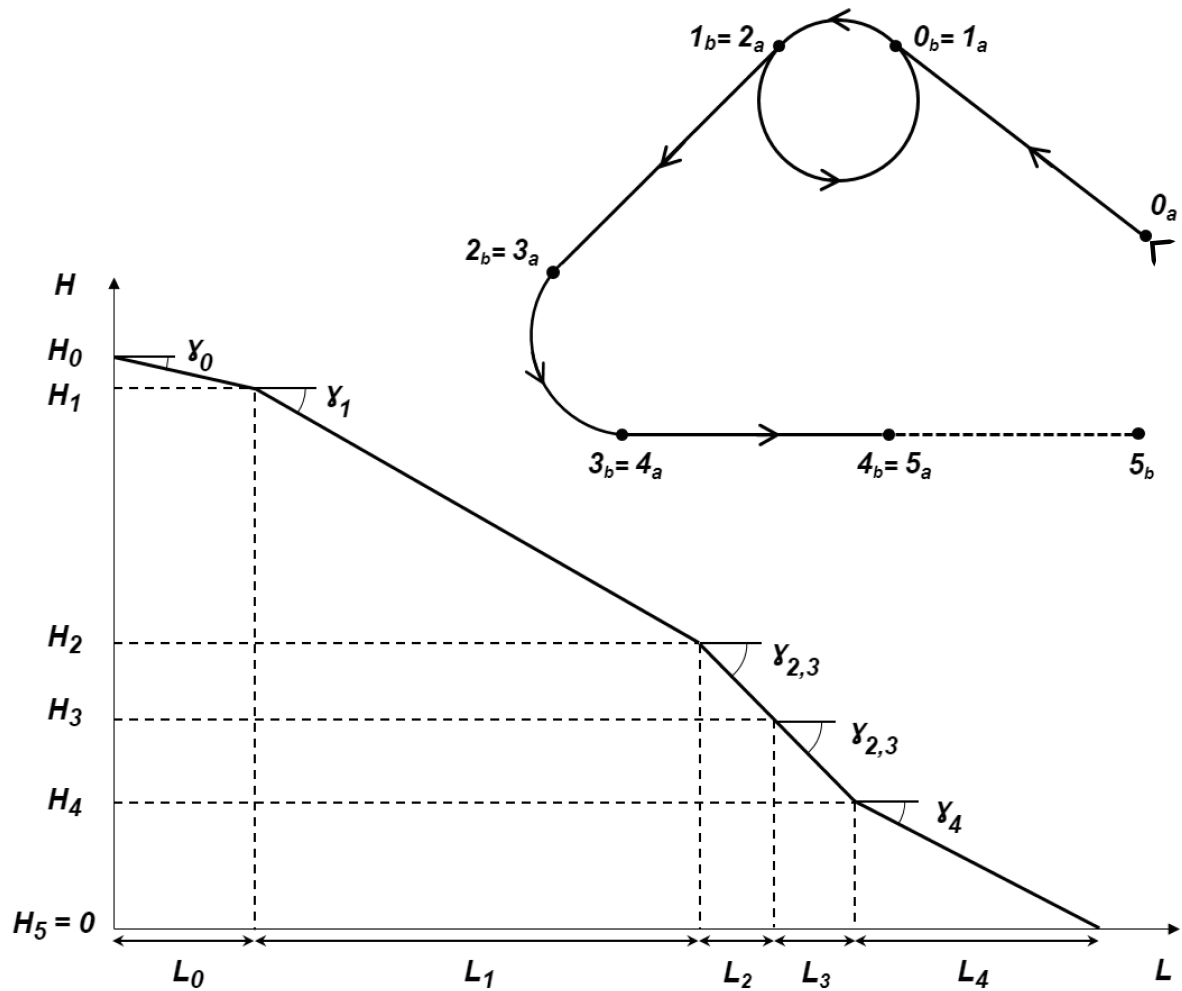


Fig. 281: Landing route top and front views with parameter identification

2.9.5.6 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 **Interest** input to improve the precision from its guidance, being the most suitable kind for Interest integration.

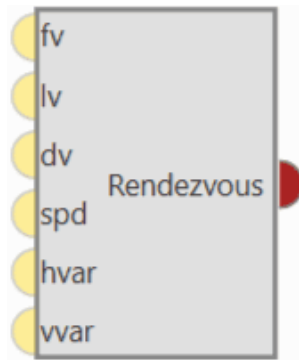


Fig. 282: **Rendezvous block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidance Computation* block.

- **Configuration menu:**

All the parameters that define the rendezvous guidance are detailed.

The screenshot shows the 'Rendezvous' configuration window. It contains several sections:

- 1 Patch:** A checkbox and a dropdown menu.
- 2 Set height mode:** A dropdown menu set to '3D'.
- 3 Arcade position/speed transition:** Two input fields for 'Horizontal' and 'Vertical', both set to '1.0' with units of 'm/s'.
- 4 Set speed:** A section with 'Cruise' (5.0 m/s), 'Waypoint' (5.0 m/s), 'Type' (Ground Speed), and 'Deceleration' (5.0 m/s²).
- 5 Guidance control ?:** A section with 'Type' and 'Horizontal'/'Vertical' controls, both set to 'PID'.
- 6 Relative positions:** A table with columns X, Y, and Z. It contains 'Rendezvous relative position' (0.0, 0.0, -8.0) and 'Docking relative position' (0.0, 0.0, 1.0), all in meters.
- 7 Base orientation:** Three controls for 'Base yaw', 'Base pitch', and 'Base roll', all set to 'Constant value: 0.0'.
- 8 Docking base:** A dropdown menu set to 'Moving Object 01'.
- 9 Use Internet:** A checkbox and a 'Timeout' field set to '0.2'.

 An 'Accept' button is at the bottom right.

Fig. 283: 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 1x autopilot orientation (for more information about 1x orientation visit [Hardware installation - Orientation](#) section of **1x user 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 Internet:** As Rendezvous navigation is prepared to take Internet 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 1x autopilot axes convention):

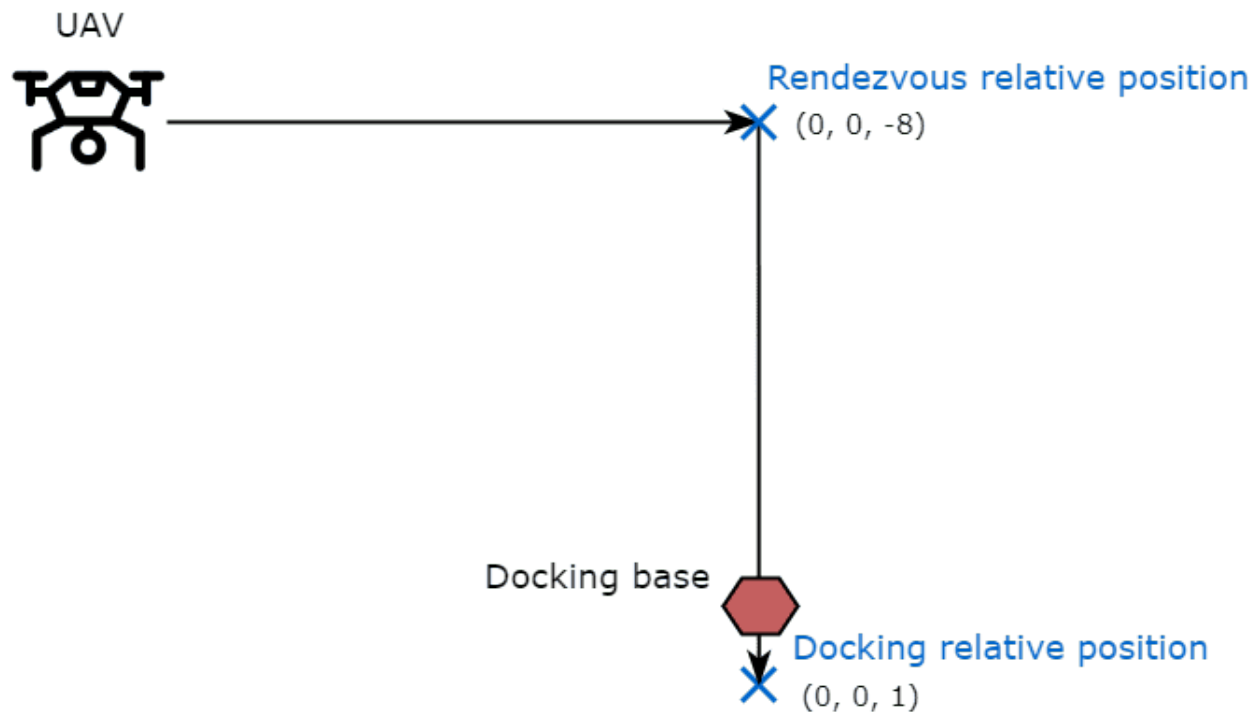


Fig. 284: Rendezvous guidance parameters

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 1x Autopilots -> 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** section → **Data link to VApp**.
2. Add **Moving Object** to the list of variables.

2.9.5.7 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.

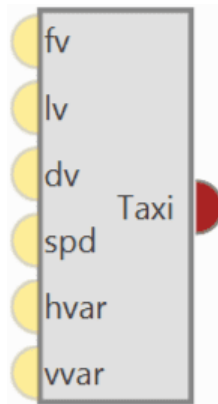


Fig. 285: **Taxi block**

Warning: In order to produce a guidance computation this block has to be connected to the *Guidance Computation* block.

- **Configuration menu:**

The screenshot shows the 'Taxi' configuration window with the following elements highlighted by red boxes and numbered callouts:



- 1:** A checkbox labeled 'Patch'.
- 2:** A dropdown menu labeled 'Set height mode' with '3D' selected.
- 3:** The 'Arcade position/speed transition' section, containing two input fields: 'Horizontal' (1.0 m/s) and 'Vertical' (1.0 m/s).
- 4:** The 'Set speed' section, containing:
 - 'Cruise' input (5.0 m/s) with a gear icon.
 - A checkbox for 'Waypoint'.
 - 'Waypoint' input (5.0 m/s) with a gear icon.
 - 'Type' dropdown (Ground Speed).
 - 'Deceleration' input (5.0 m/s²).
- 5:** The 'Guidance control' section, containing:
 - 'Type' dropdown.
 - 'Horizontal' dropdown (PID) with a pencil icon.
 - 'Vertical' dropdown (PID) with a pencil icon.
- 6:** The 'Runway' section, containing:
 - 'Runway' dropdown (TKO runway) and an 'Advanced' checkbox.
 - 'End point' dropdown (Runway End Position).
 - 'Direction' dropdown (Ref: Runway Direction) with a chain-link icon.

An 'Accept' button is located at the bottom right of the window.

Fig. 286: 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:

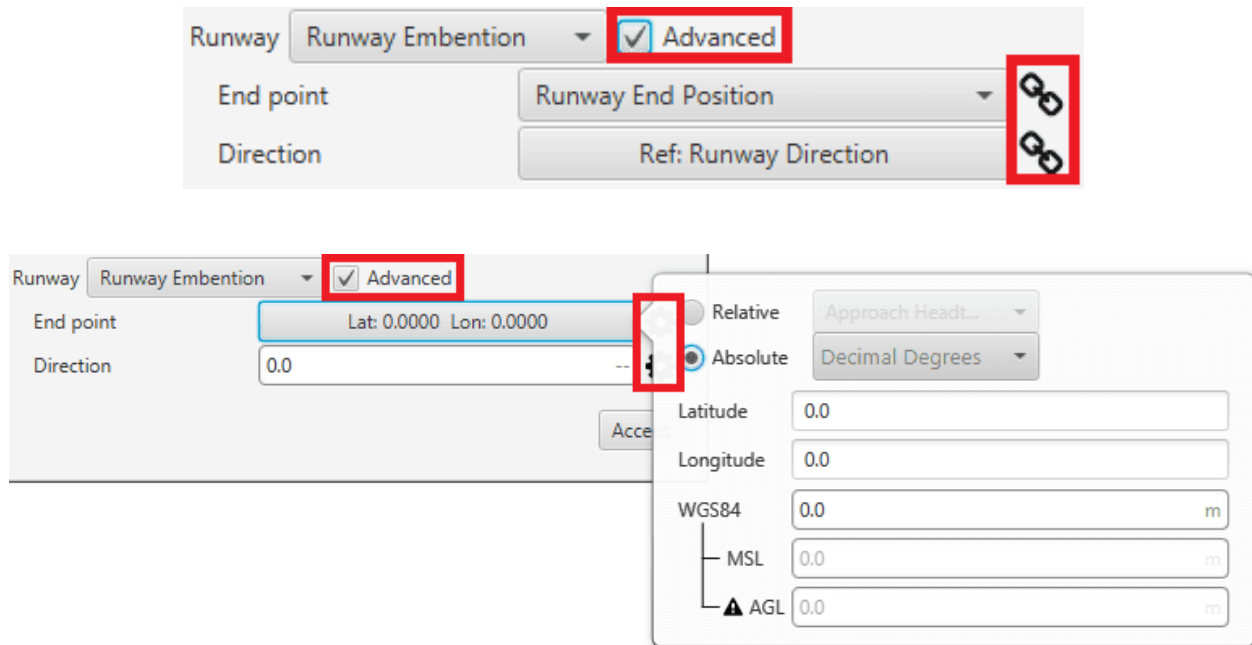






Fig. 287: Runway parameters

– **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 a 1x autopilot.
 - **Absolute:** The coordinates can be set in UTM, MGRS, Decimal Degrees or Degree ° ’ ’’. 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.

2.9.5.8 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.

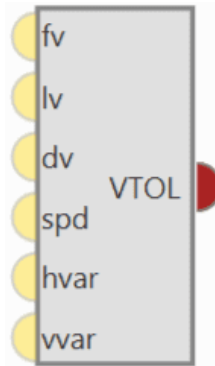


Fig. 288: VTOL block

Warning: In order to produce a guidance computation this block has to be connected to the *Guidance Computation* block.

- Configuration menu:

The screenshot shows the 'VTOL' configuration window. It contains several sections, each highlighted with a red box and a numbered callout (1-6) on the right side:

- 1**: A checkbox labeled 'Patch'.
- 2**: A dropdown menu labeled 'Set height mode' with '3D' selected.
- 3**: A section titled 'Arcade position/speed transition' containing two input fields: 'Horizontal' (1.0 m/s) and 'Vertical' (1.0 m/s).
- 4**: A section titled 'Set speed' containing:
 - 'Cruise' speed: 5.0 m/s (with a gear icon).
 - 'Waypoint' speed: 5.0 m/s (with a gear icon).
 - 'Type': 'Ground Speed' (dropdown).
 - 'Deceleration': 5.0 m/s².
- 5**: A section titled 'Guidance control' with a question mark icon. It contains:
 - 'Type' dropdown.
 - 'Horizontal' control: 'PID' (dropdown) with an edit icon.
 - 'Vertical' control: 'PID' (dropdown) with an edit icon.
- 6**: A section containing:
 - 'Type': 'Straight' (dropdown).
 - 'Extend': 'None' (dropdown).
 - 'Safe': 0.0 m (with a gear icon) and a 'Relative' dropdown.
 - 'Touch': 'Runway Touch Point' (dropdown) with a link icon.

An 'Accept' button is located at the bottom right of the window.

Fig. 289: 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 drop-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 a 1x autopilot.

- **Absolute:** The coordinates can be set in UTM, MGRS, Decimal Degrees or Degree ° ' ' '. They are indicated through the latitude and longitude.

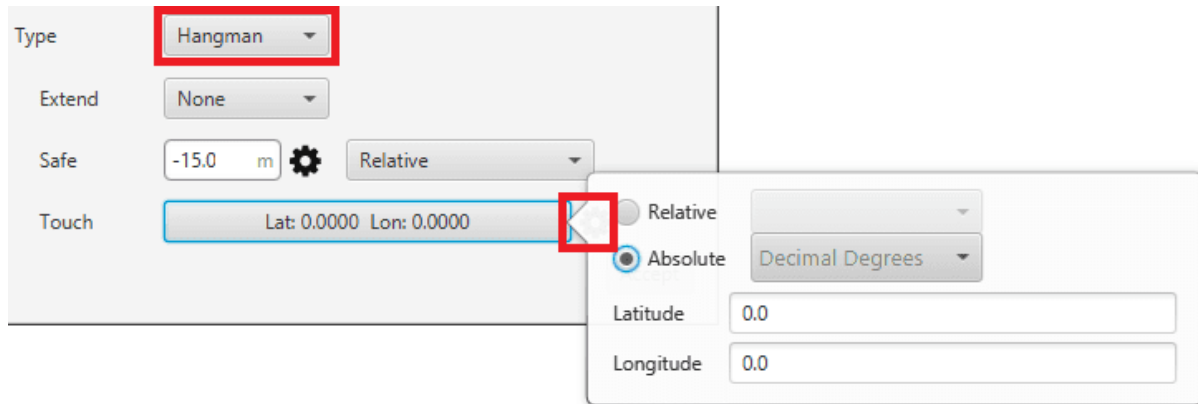


Fig. 290: Touch options

The following image gives an overview of some parameters introduced:

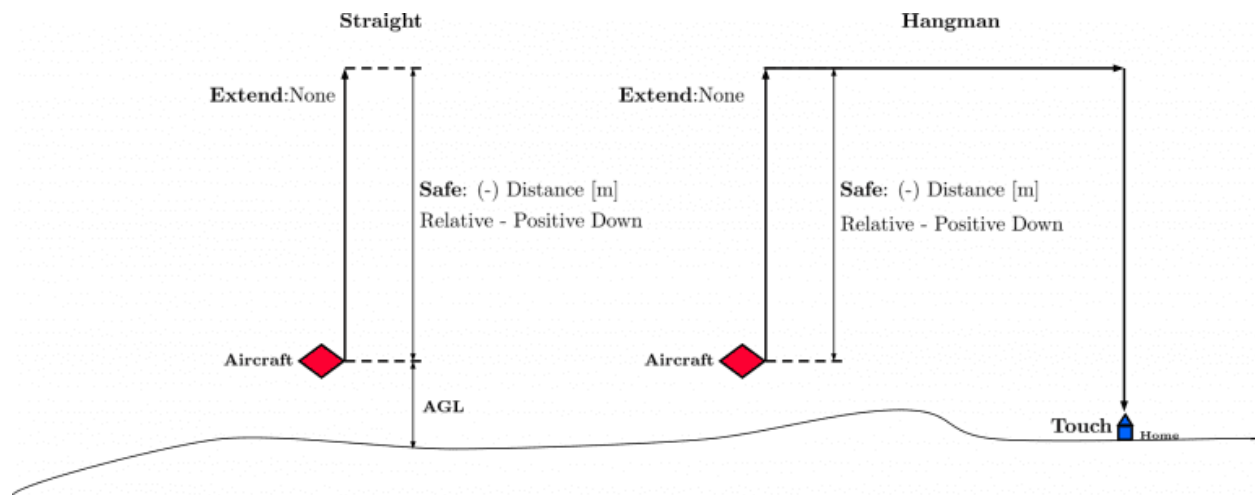


Fig. 291: Parameters Overview

2.9.5.9 Yawing current

Yaw guidance is used in multicopters to indicate the behaviour 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 parallel 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**.

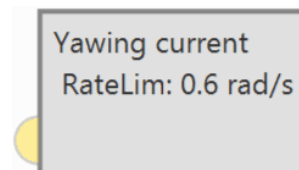


Fig. 292: Yaw current block

- **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. 293: 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**.

2.9.5.10 Yawing heading

Yaw guidance is used in multicopters to indicate the behaviour 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 parallel 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**.

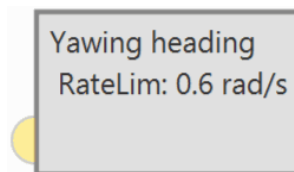


Fig. 294: Yaw heading block

- **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:**

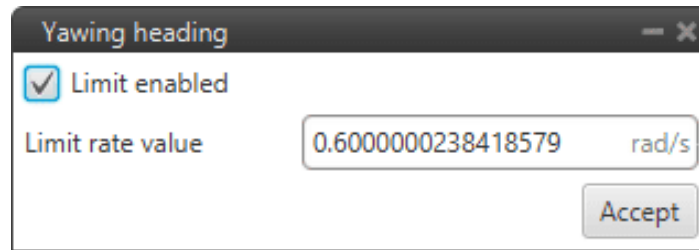


Fig. 295: Yaw heading block configuration

- **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**.

2.9.5.11 Yawing north

Yaw guidance is used in multicopters to indicate the behaviour 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 parallel 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**.

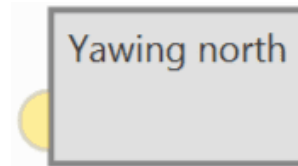


Fig. 296: Yaw north block

- **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:**

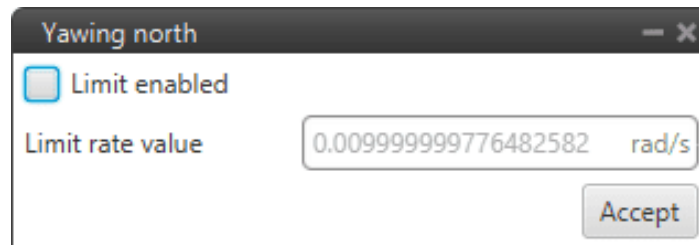


Fig. 297: Yaw north block configuration

- **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.

2.9.5.12 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.

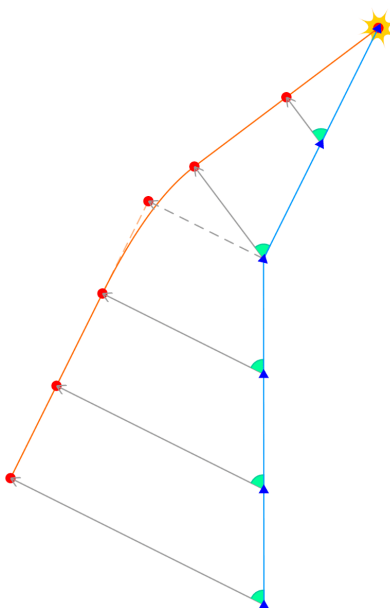


Fig. 298: **PNav algorithm**: A missile (blue) intercepts a target (red) by maintaining constant bearing to it (green)

- **Modified PNav**: Modified proportional navigation guidance (Old Miura).
- **GENEX**: Generalized Vector Explicit Guidance (GENEX) block.

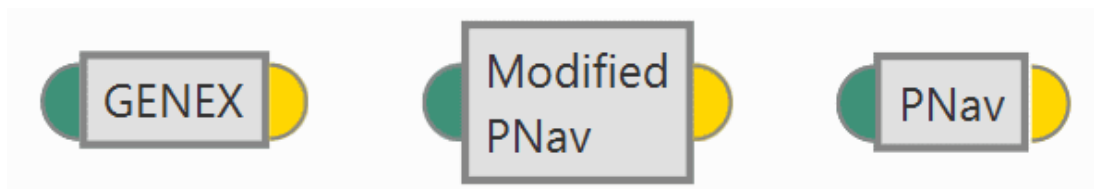


Fig. 299: **Navigation guidance blocks**

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:**

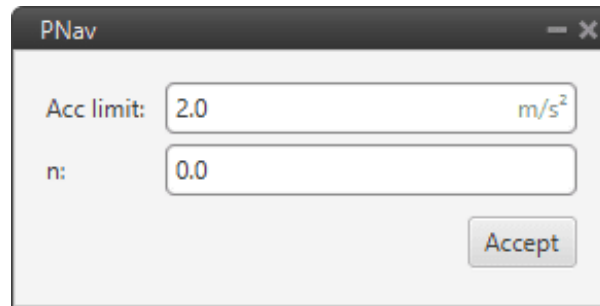


Fig. 300: PNav block configuration

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.

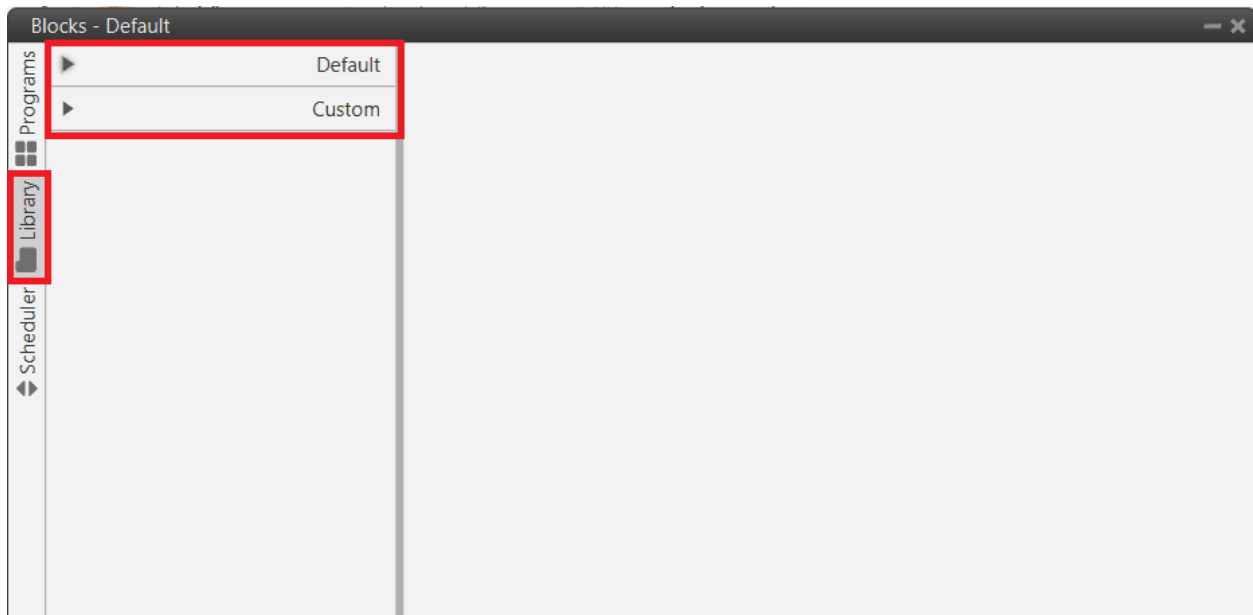


Fig. 301: Library blocks menu

- **Custom**

These are custom blocks created by the user. They can be added by simply clicking on the “+” icon. An example of how to create one of these blocks is shown below:

Fig. 302: Example of custom library block

Examples of such blocks are those related to the Stick, which have been named “Center Stick” and “Trim Stick”:

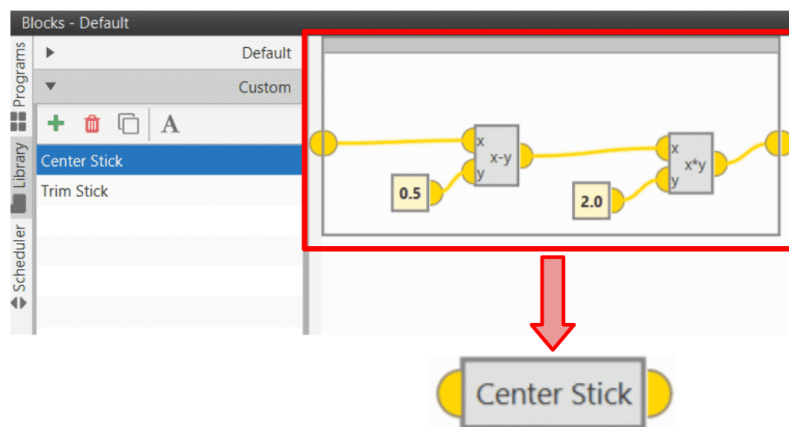


Fig. 303: Example of custom library block - Center Stick

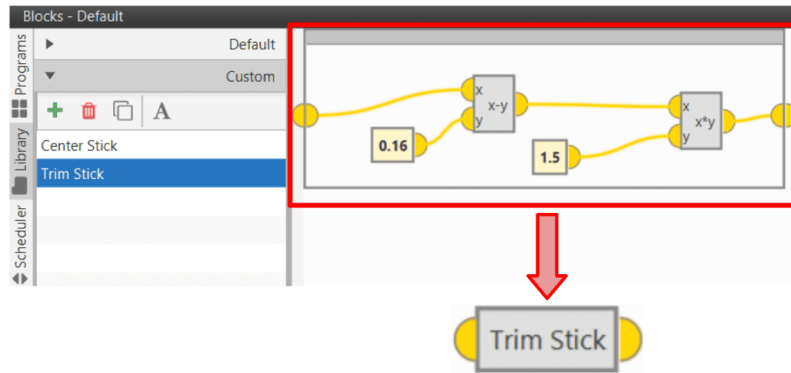


Fig. 304: Example of custom library block - Trim Stick

- **Default**

There is only one 'Default' library block, it is called: **Sagotech ADSB**. This is a block created for the Transponder/ADS-B "**Sagotech MXS**".

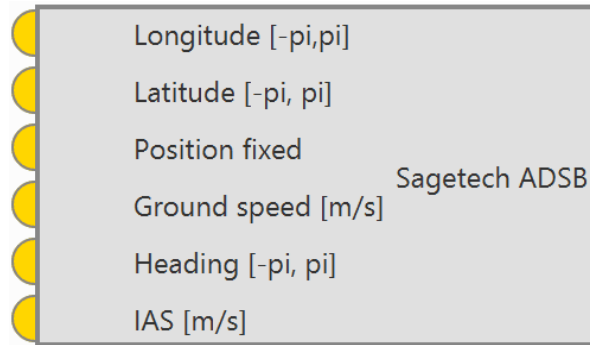


Fig. 305: Default library block - Sagotech ADSB block

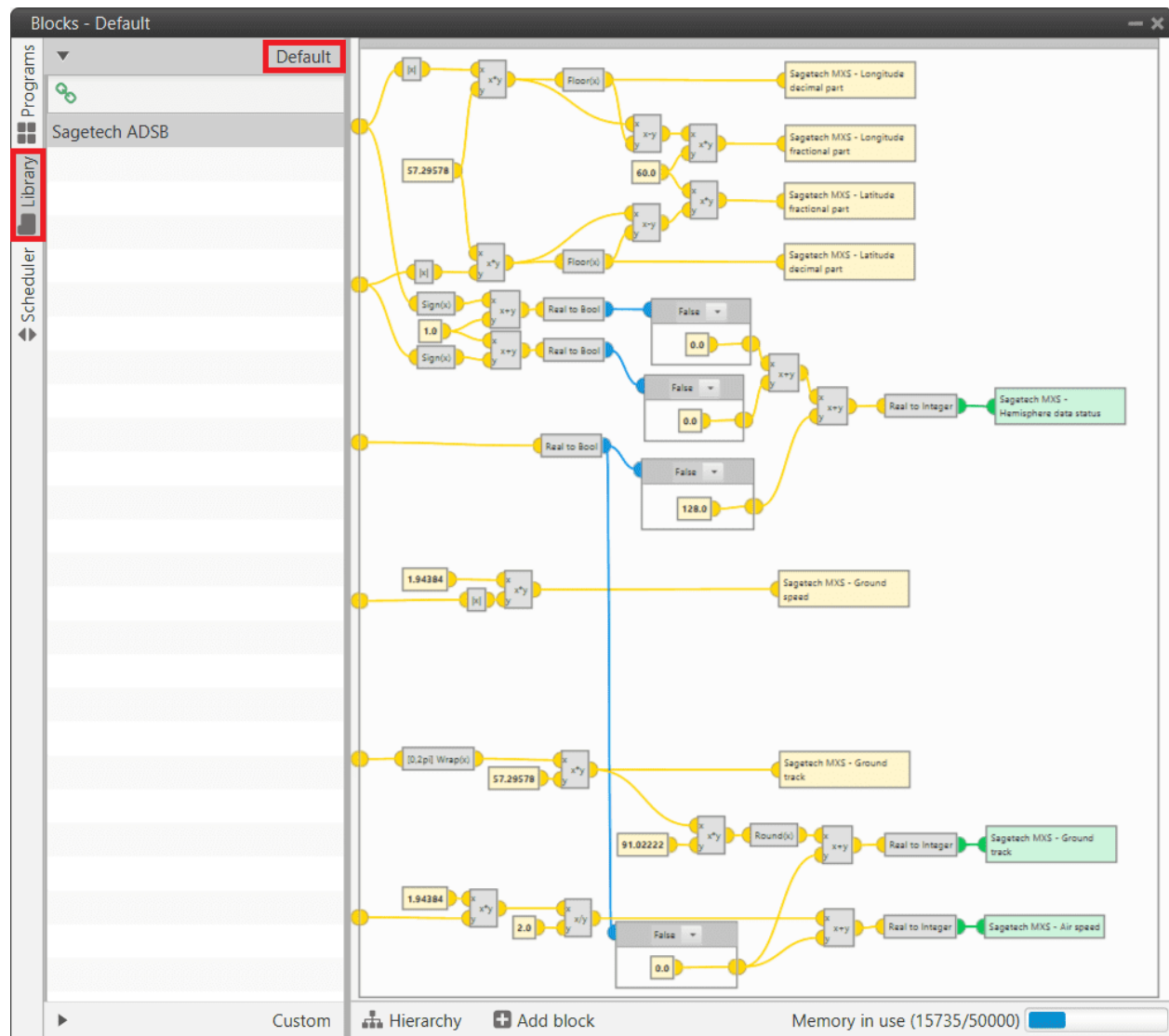


Fig. 306: Default library block - Sagatech ADSB block configuration


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. 307: Default library block - Sagatech 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.

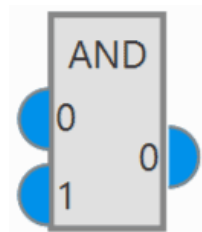


Fig. 308: AND block

- **Inputs**
 - 0: Input bit.
 - 1: Input bit.
- **Output**
 - 0: Output bit.
- **Configuration menu**



Fig. 309: 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 OR

Returns true if ANY of inputs are true, else return false.

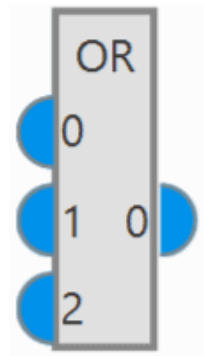


Fig. 310: OR block

- **Inputs**

- **0:** Input bit.

- **1:** Input bit.

- **2:** Input bit.

- **Output**

- **0:** Output bit.

- **Configuration menu**



Fig. 311: 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**.

2.9.7.3 OR

Logical complement (negation) computation.



Fig. 312: NOT block

- **Input**

- **0:** Input bit.

- **Output**

- 0: Output negated bit.

2.9.8 Math blocks

Math blocks allow to perform a wide variety of mathematical operations.

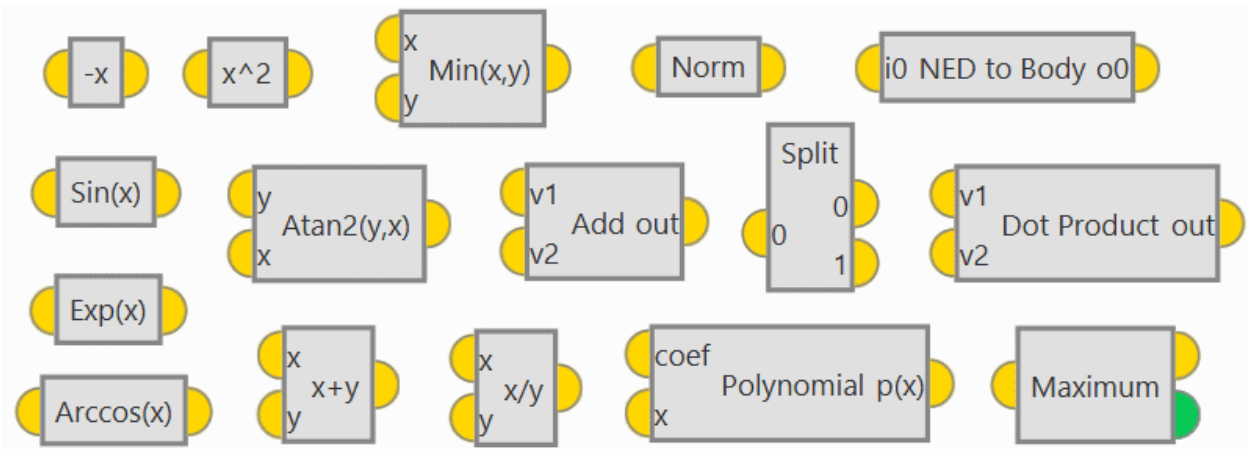


Fig. 313: **Math blocks**

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 integer 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**: Multiplier block.
- **x+y**: Adder block.
- **x-y**: Subtract block computes the subtraction of the first input minus 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 .

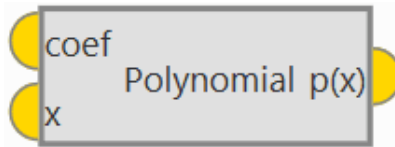


Fig. 314: Polynomial block

- **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**

- : 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).

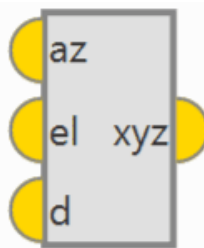


Fig. 315: azeld -> xyz block

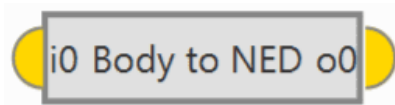
- **Inputs**

- **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. 316: **Body to NED block**

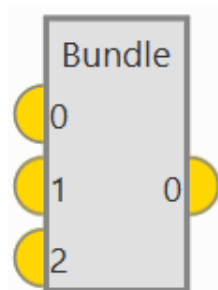
– **Input**

- **i0:** Vector in Body frame.

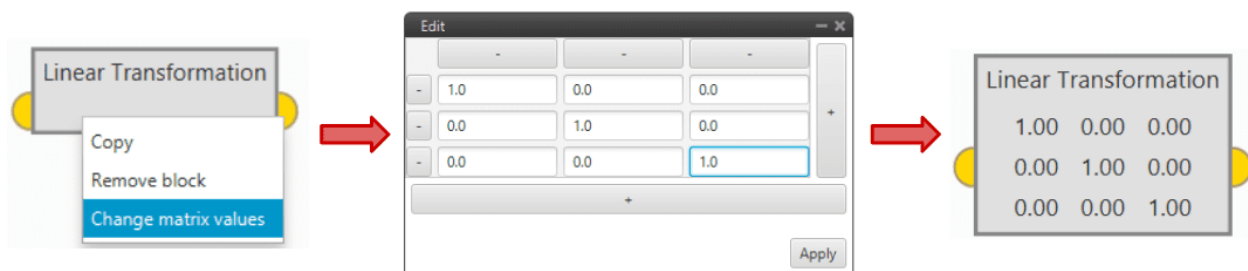
– **Output**

- **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. 317: **Bundle block**

- **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. 318: **Linear Transformation block**

- **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. 319: **Body to NED block**

– **Input**

- **i0**: Vector in Body frame.

– **Output**

- **o0**: Vector in NED frame.

- **Norm**: Computes the norm of the input vector.
- **Scale**: Multiply the input vector (**vIN**) by a scalar value (**k**).
- **Split Bool**: Splits a vector of booleans into singular boolean variables. In its configuration, the user can set the number of outputs.
- **Split Real**: Splits a vector of real variables into singular real variables. In its configuration, the user can set the number of outputs.
- **xyz -> azeld**: Conversion from NED (North, East, Down) to azimuth, elevation and distance.

Fig. 320: **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 (4x autopilot).

2.9.9.1 AP Selection

AP selection block is a **4x Veronte Autopilot** selection helper. This block helps providing the control outputs from other autopilots according to the '4x selection' table.

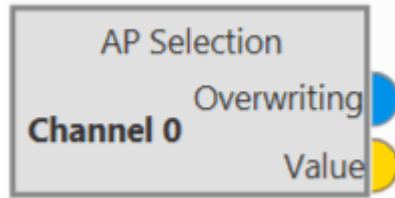


Fig. 321: AP Selection block

- **Outputs**

- **Overwriting:** **TRUE** if this channel is being overwritten by other autopilot, that means that the current autopilot is not selected.

FALSE if the current autopilot is selected, and therefore it has the control of this channel.

- **Value:** If the channel is being overwritten by other autopilot (Overwriting is **TRUE**), it returns the value that the other autopilot is applying. Otherwise (Overwriting is **FALSE**), value is 0.

- **Configuration menu:** The user must select the channel to be controlled.

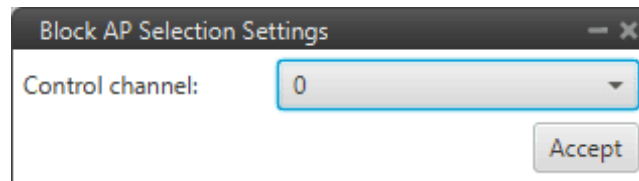


Fig. 322: AP Selection block configuration

Attention: If the **4x Veronte tab** of the *Control menu* is not configured, the following warning message will appear when trying to open the configuration menu of this block.

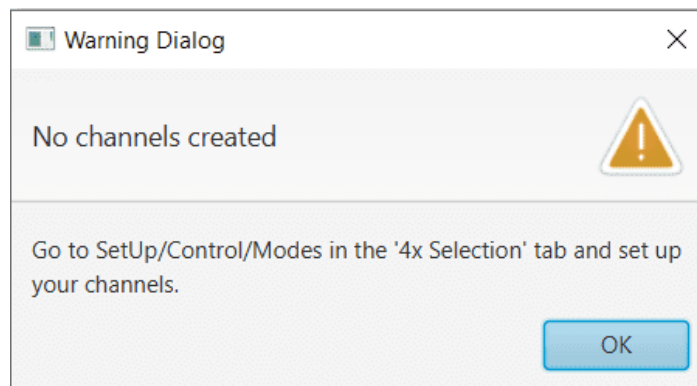


Fig. 323: AP Selection block warning message

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 4x Autopilot 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:

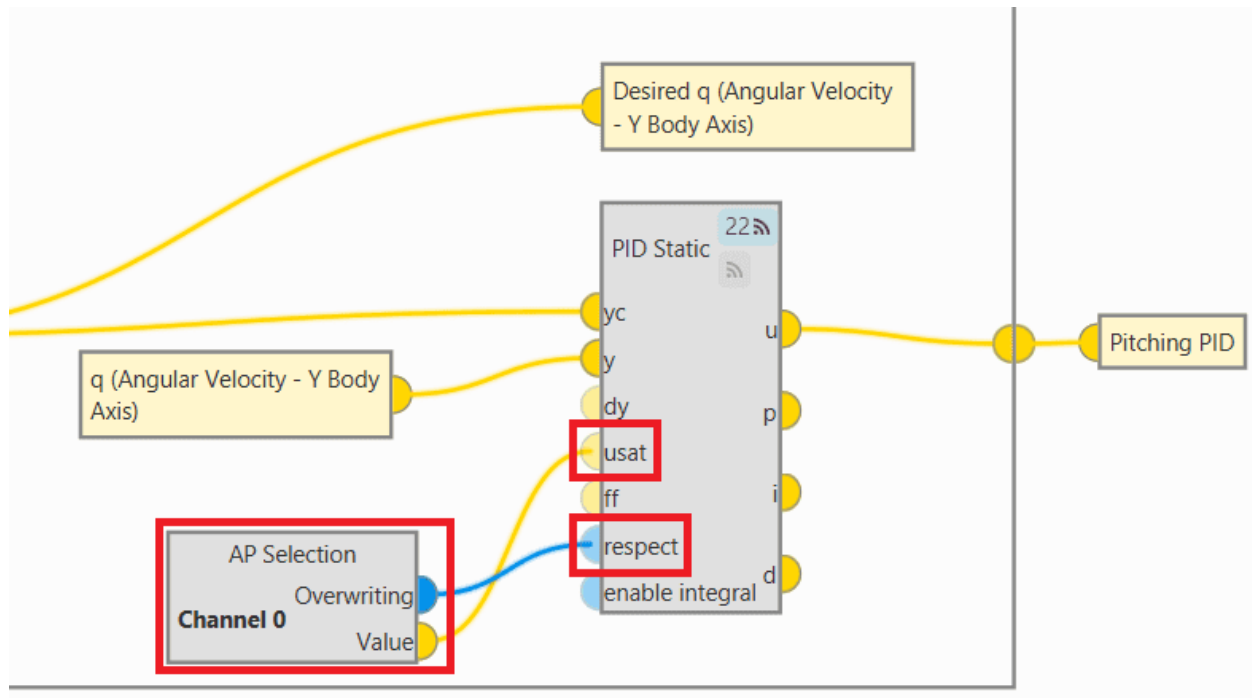


Fig. 324: AP Selection block - Example of use

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 *mode configuration table* to check the configuration, see [Modes section](#).

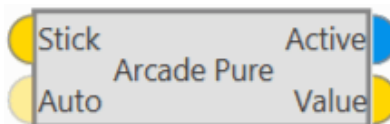


Fig. 325: Arcade block

- **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**.

- **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:

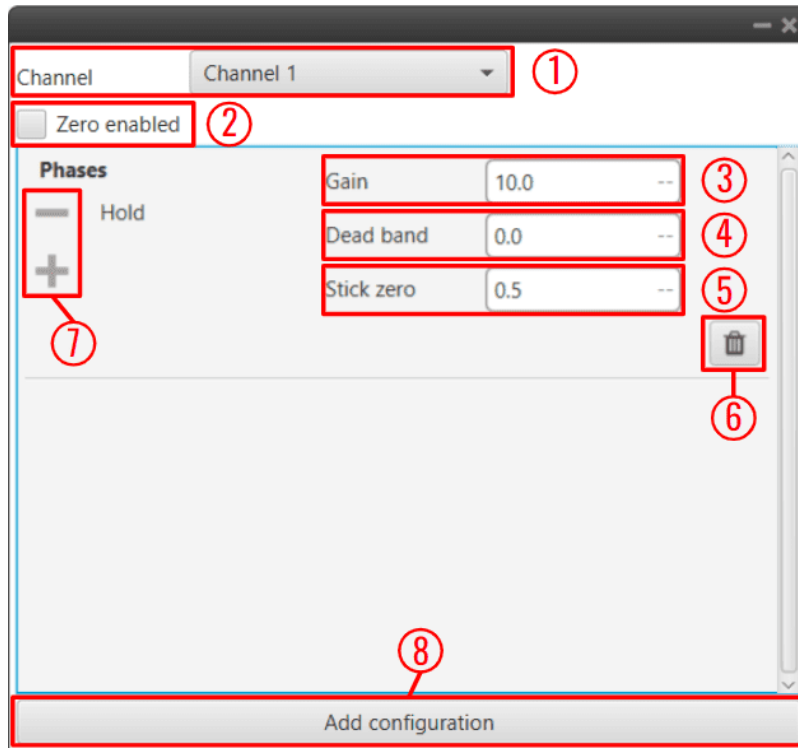


Fig. 326: Arcade block configuration

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 section*.
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.
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, see [Modes section](#).

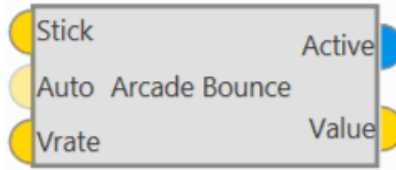


Fig. 327: **Arcade Bounce** block

In the pictures below there is an example controlling the yaw.

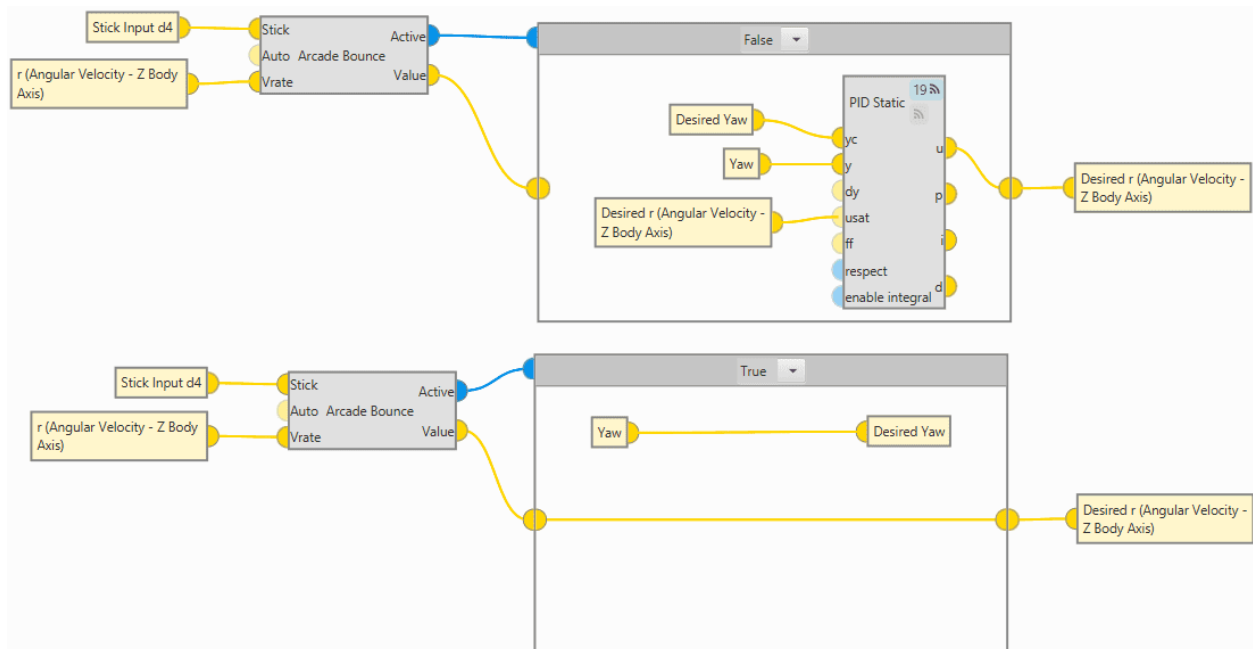


Fig. 328: **Arcade Bounce** block example

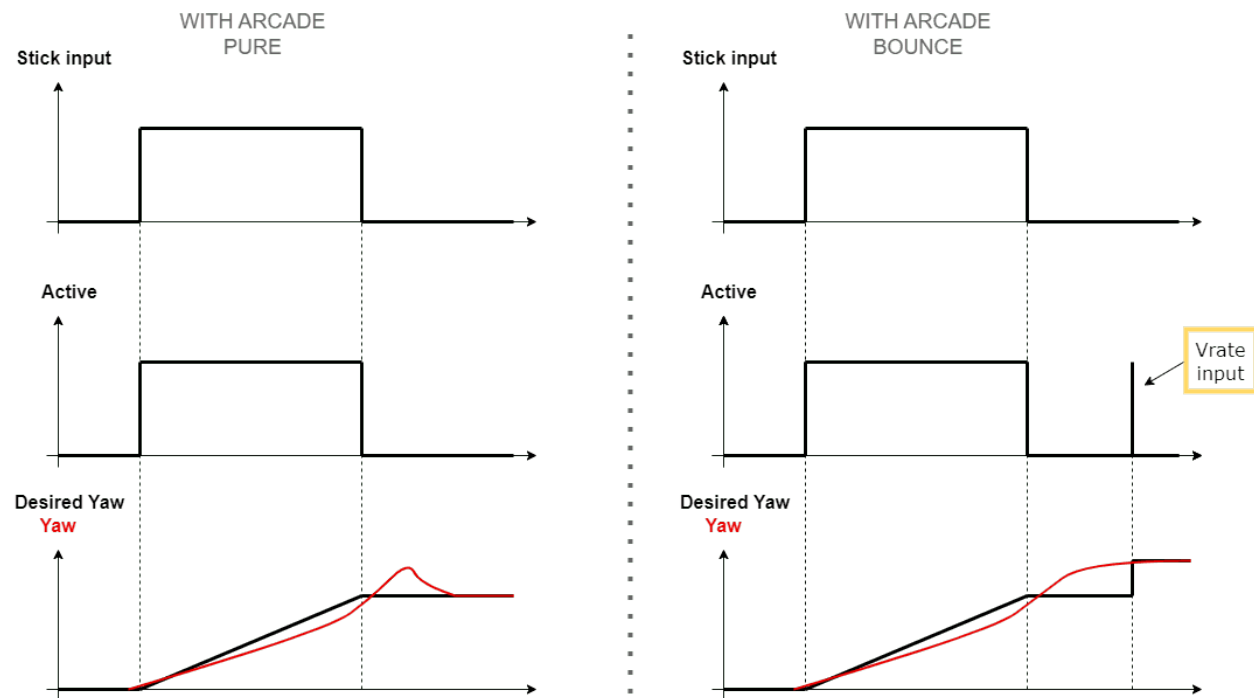


Fig. 329: Arcade Pure VS Arcade Bounce

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:**

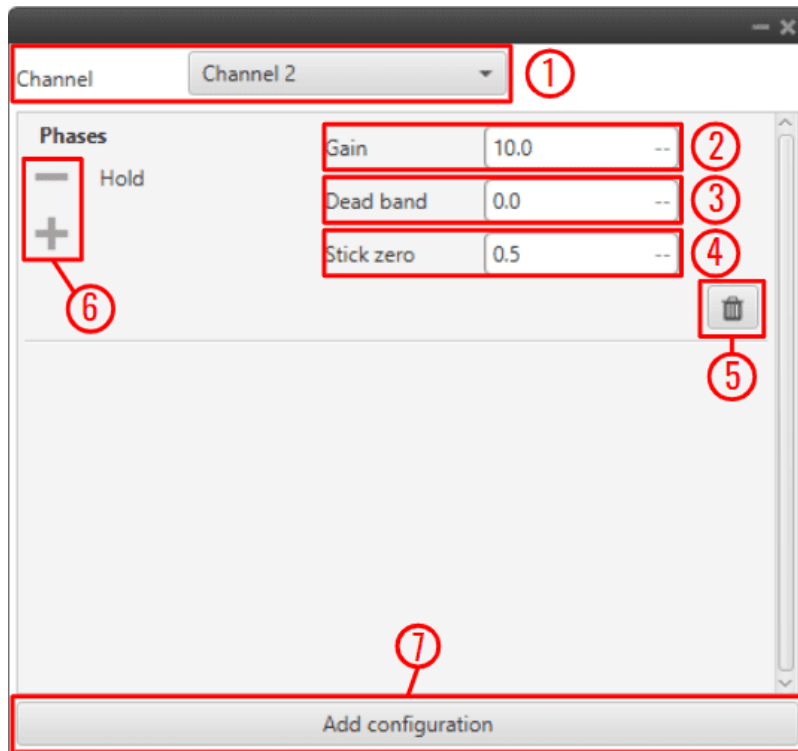


Fig. 330: Arcade Bounce block configuration

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 section*.
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, see *Modes section*.

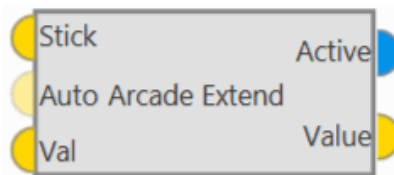


Fig. 331: Arcade Extend block

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:**

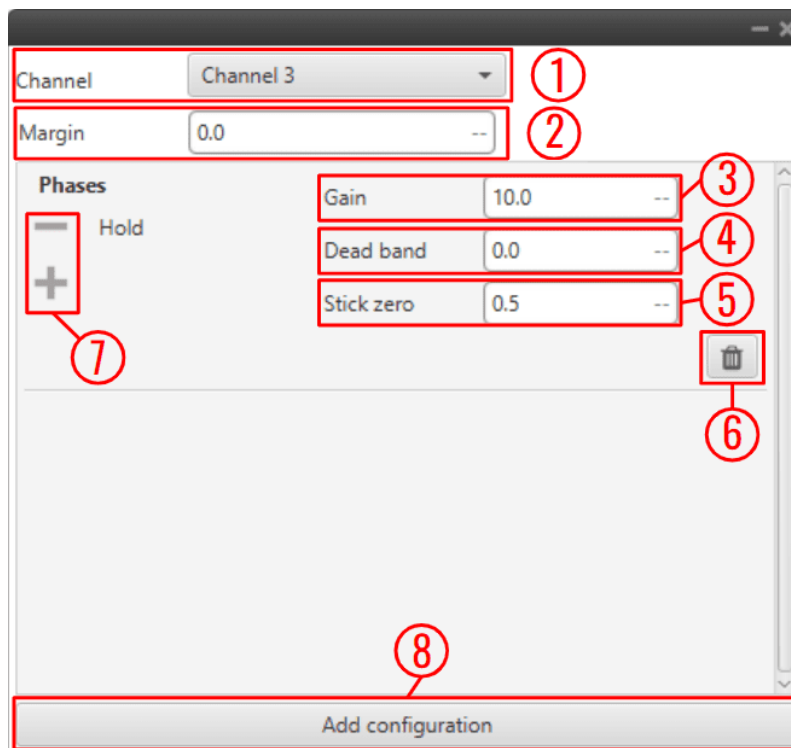


Fig. 332: Arcade Extend block configuration

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 section*.
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, see *Modes section*.

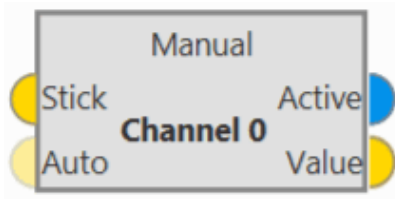


Fig. 333: Manual block

- **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.

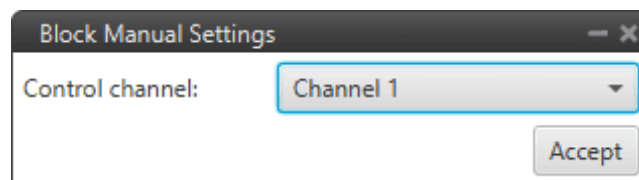


Fig. 334: Manual block configuration

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.

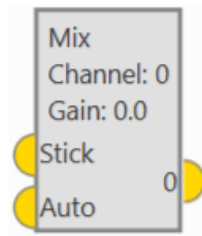


Fig. 335: Mix block

- **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:**

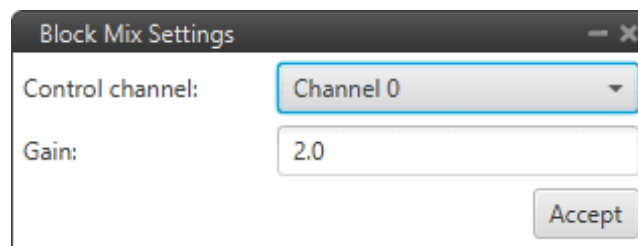


Fig. 336: Mix block configuration

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.

Altitude

Altitude block adapts an external altimeter sensor, like LIDAR, Sonar, etc., to the EKF input.

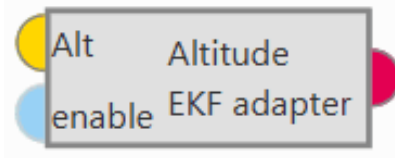


Fig. 337: Altitude block

- **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:**

The dialog box titled "Range sensor EKF adapter settings" contains the following configuration options:

- Decimation:** A text box with the value "40" and a unit dropdown set to "dec".
- Position of altimeter (Body frame):** Three text boxes for X, Y, and Z coordinates, all with the value "0.0" and a unit dropdown set to "m".
- Enable tilt limit:** A checked checkbox next to a text box with the value "-2.5663707" and a unit dropdown set to "rad [-π,π]".
- Enable tilt correction:** A checked checkbox.
- Enable sensor limit:** A checked checkbox next to two text boxes with values "0.05" and "40.0", both with unit dropdowns set to "m".

At the bottom of the dialog are "Accept" and "Cancel" buttons.

Fig. 338: Altitude block configuration

The following accelerometer 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).
- **Position of altimeter (Body frame):** Parameter to indicate the distance from the altimeter to the centre 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 measurement, with internal pitch and roll measurements.

$$correction = (AGL\ measurement) \cos(pitch) \cos(roll)$$

- **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.

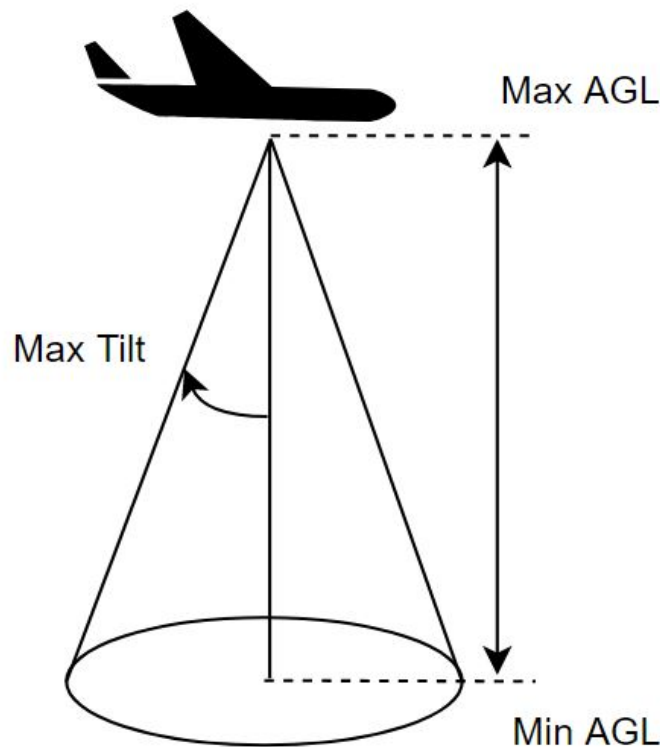


Fig. 339: Altitude block - Limits

GNSS compass

GNSS compass block takes two relative position measurements and converts them to **misalignment vectors** to use in the EKF for attitude correction.

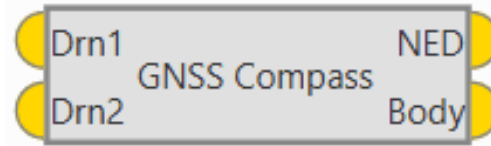


Fig. 340: GNSS compass block

- **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:

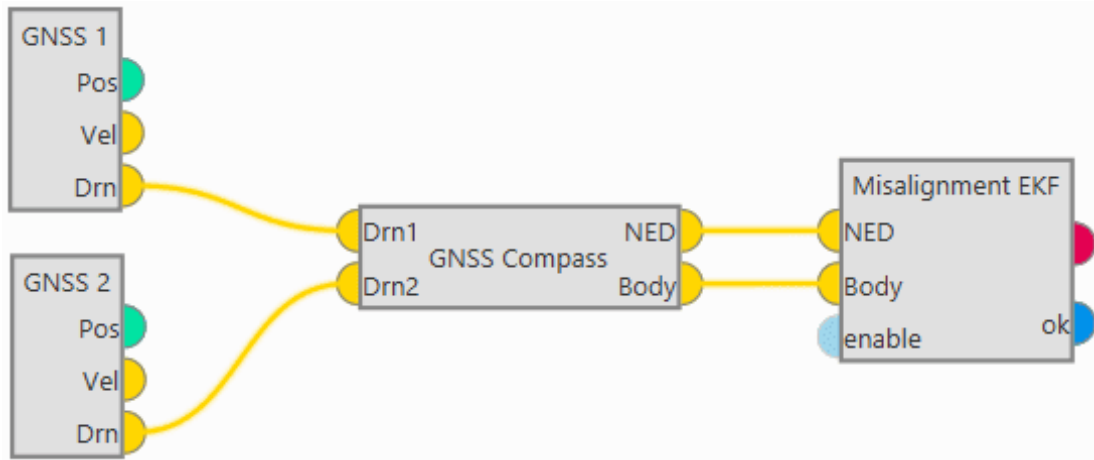


Fig. 341: GNSS compass block example

Misalignment

Misalignment block transforms from **two vectors** expressed in NED and body frames to **EKF misalignment data for attitude correction**.

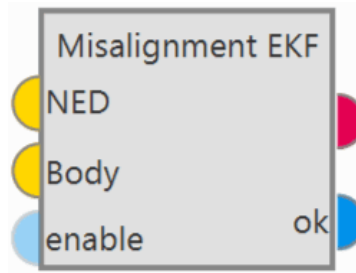


Fig. 342: Misalignment block

• 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. 343: 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).
- **Use 3D:** If enabled, the attitude correction will be in 3D, as by default, only the attitude correction in the horizontal plane is activated.

Position

Position block adapts **absolute position** data to **EKF data for position update**.

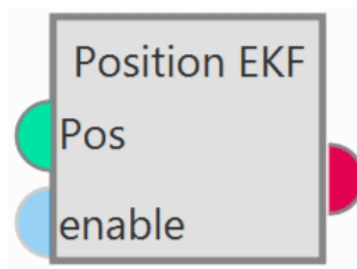


Fig. 344: **Position block**

• 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.**

• Outputs

- **Pin 0:** EKF input data (H, R, y).

Note:

- **H:** Observation matrix.
- **R:** Measurement covariance matrix.
- **y:** Measurement.

• Configuration menu:

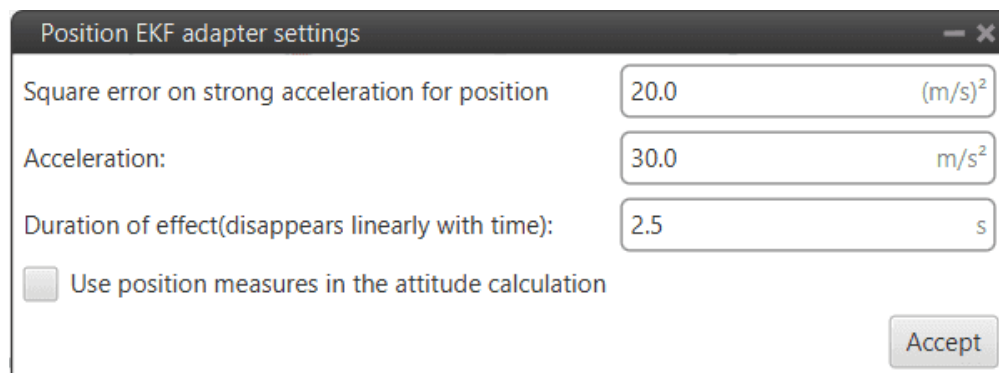


Fig. 345: **Position block configuration**

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.

Static Pressure

Static Pressure block converts **static pressure** measurement into **EKF data for altitude update**.

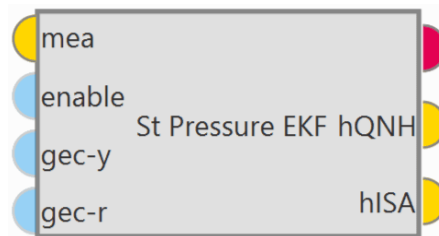


Fig. 346: Static Pressure block

• 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:**

Navigation EKF Adapter for Static Pressure

Navigation

Decimation

Variance rate limit

☒ Max falling rate Pa²/s

☐ Max rising rate Pa²/s

Ground effect

Square error Pa²

Altitude correction threshold (Positive down) m

Fig. 347: Static Pressure block configuration

The following parameters must be configured:

Navigation

- **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).

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.

Ground effect: Veronte Autopilot 1x is configured to apply a correction to the Ground Effect for landing purposes, which needs to be configured here.

- **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 decrease 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 Autopilot 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 table. 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.

Correction compensation: Users can edit the correction compensation by clicking here:

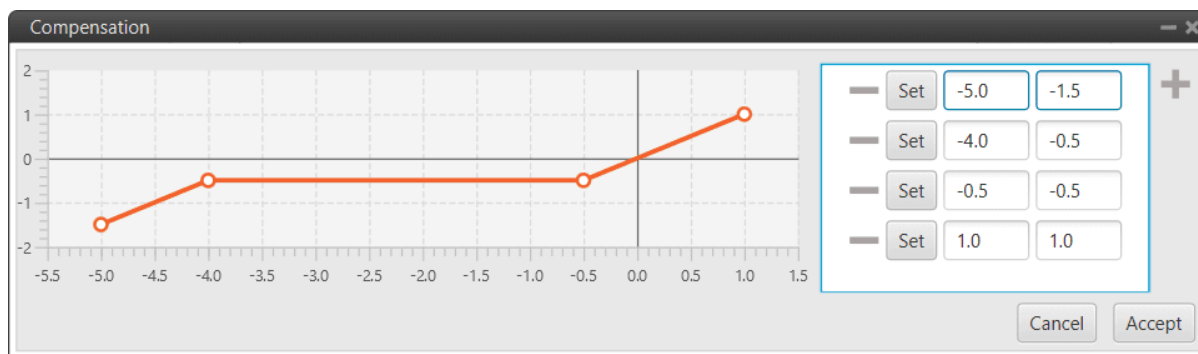


Fig. 348: Static Pressure block configuration - Correction compensation table

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 unit of measurement is meters.

Terrain height

Terrain height block transforms from **terrain height** measurement to **EKF data for terrain height update**.

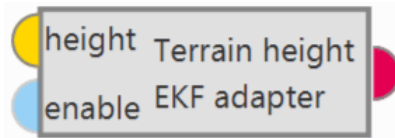


Fig. 349: **Terrain height** block

- **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.

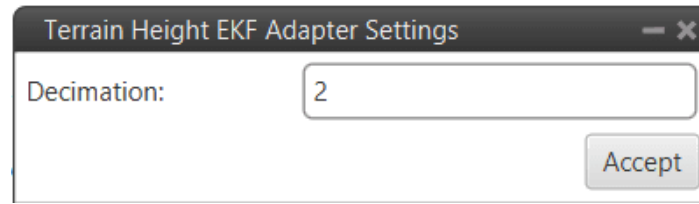


Fig. 350: Terrain height block configuration

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).

Velocity

Velocity block converts **velocity** measurement into **EKF data for velocity update**.

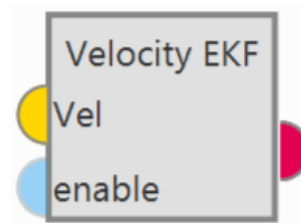


Fig. 351: Velocity block

• 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:**

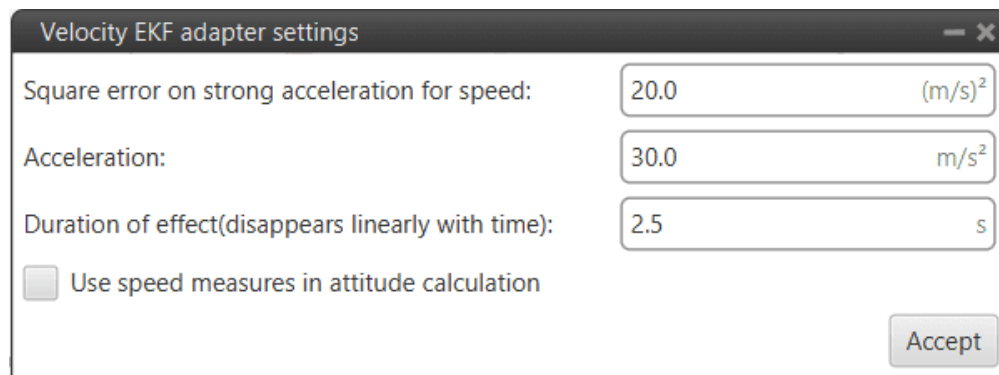


Fig. 352: **Velocity block configuration**

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.

Velocity down

Velocity down block adapts the **velocity down** measurement to **EKF data for velocity down update**.

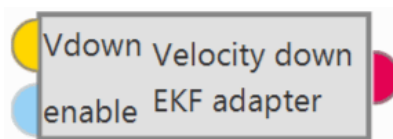


Fig. 353: **Velocity down block**

- **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:**

The screenshot shows a dialog box titled "Sensor velocity down settings". It contains the following controls:

- Decimation**: A text input field with the value "10".
- Sensor variance ((m/s)²)**: A text input field with the value "5.0" and a "--" unit indicator.
- Enable tilt limit**: A checked checkbox.
- Max tilt**: A text input field with the value "0.0" and a "rad ..." unit indicator.
- Enable speed limit**: An unchecked checkbox.
- Min speed down**: A text input field with the value "0.0" and a "m/s" unit indicator.
- Max speed down**: A text input field with the value "0.0" and a "m/s" unit indicator.
- Buttons**: "Accept" and "Cancel" buttons at the bottom left.

Fig. 354: **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).
- **Sensor variance ((m/s)²)**: Sensor error variance in metres per second squared.
- **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.

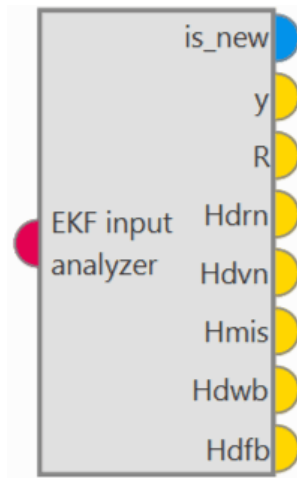


Fig. 355: **EKF Split** block

- **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 va must be connected to it in order to display the data.

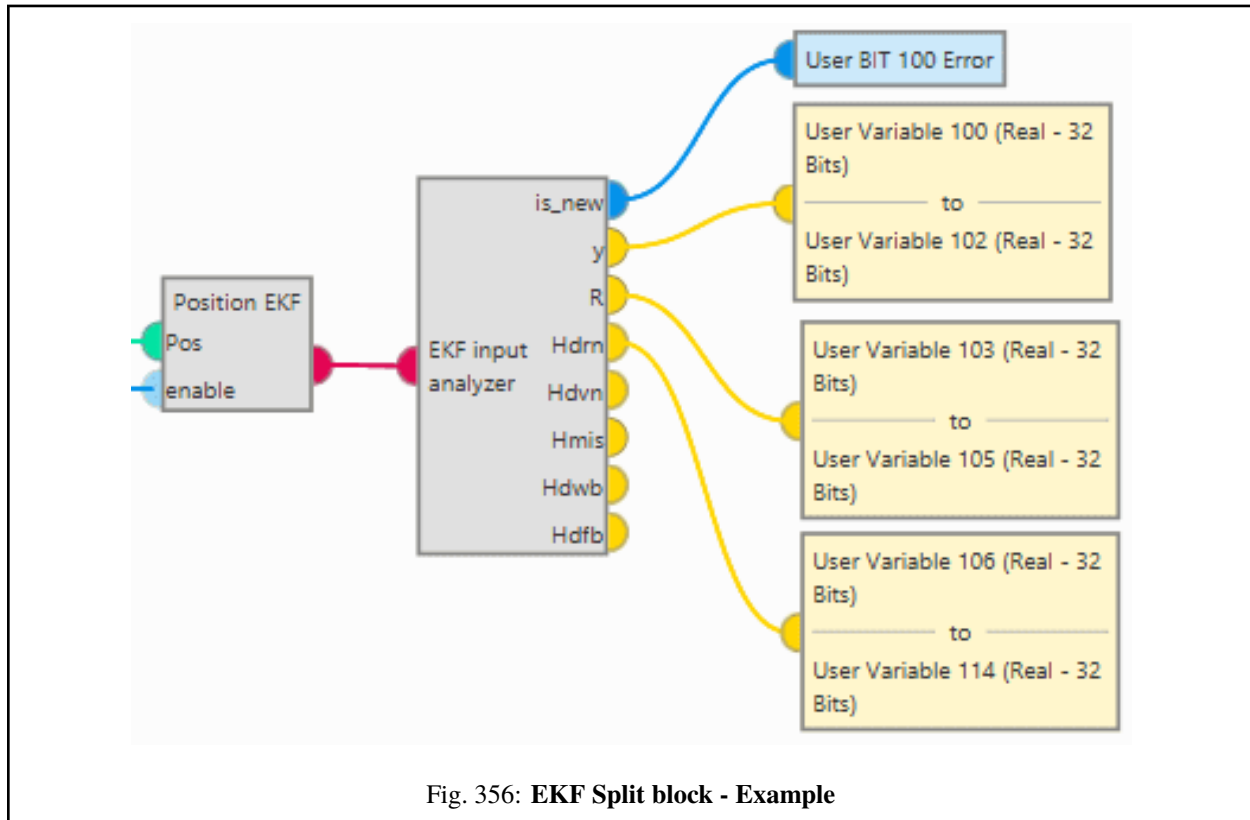


Fig. 356: EKF Split block - Example

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.

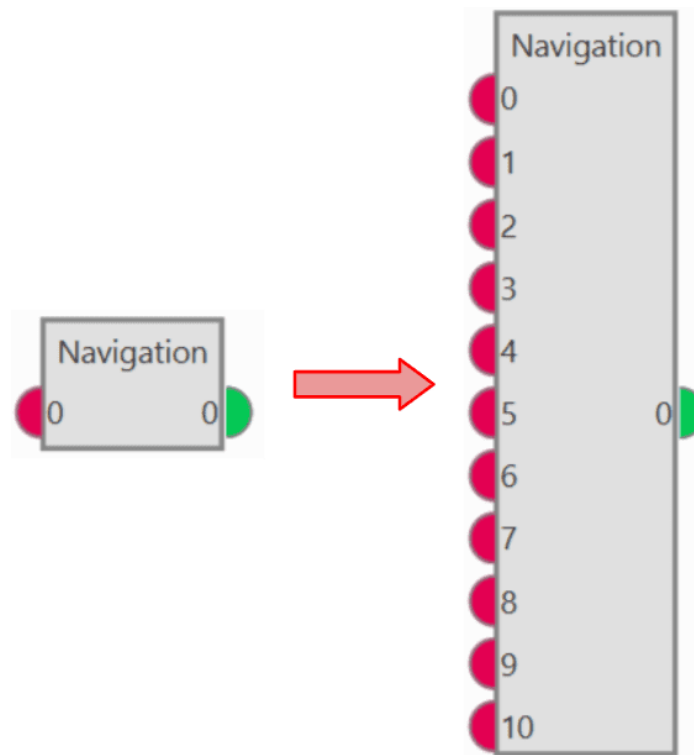


Fig. 357: Navigation block

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. 358: 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**:

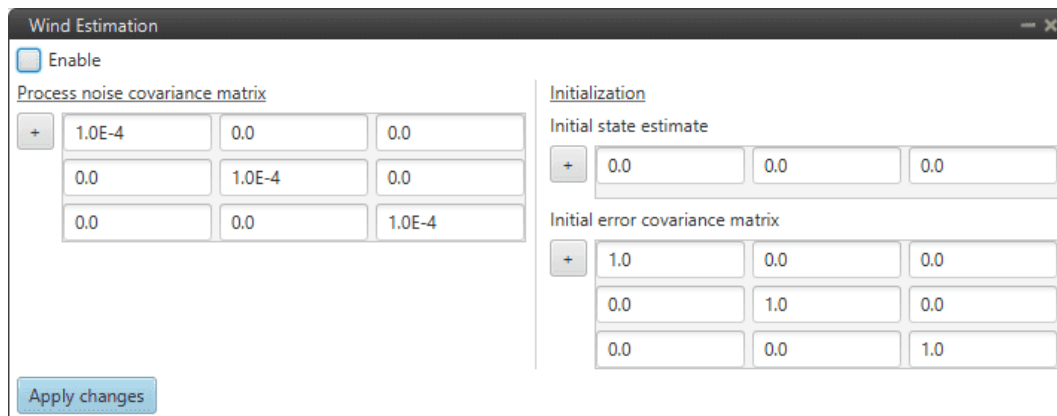
- **Inputs size:** Users must set the number of inputs.
- **Navigation:** The type of navigation must be selected, the available options are:
 - * **Internal:** Uses internal data for navigation. Data (position, attitude, etc.) is processed into 1x unit from sensor measures.
 - * **External VCP:** Uses 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:



The 'Wind Estimation' configuration window contains the following elements:

- Enable:** A checkbox that is currently unchecked.
- Process noise covariance matrix:** A 3x3 matrix with a '+' button on the left. The values are:

1.0E-4	0.0	0.0
0.0	1.0E-4	0.0
0.0	0.0	1.0E-4
- Initialization:**
 - Initial state estimate:** A 1x3 vector with a '+' button on the left. The values are 0.0, 0.0, and 0.0.
 - Initial error covariance matrix:** A 3x3 matrix with a '+' button on the left. The values are:

1.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	1.0
- Apply changes:** A button at the bottom left.

Fig. 359: 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 referred to as **Features**.

2.9.11.1 Constant Position

Constant Position block defines a position on Earth using Latitude, Longitude and WGS84 Height.

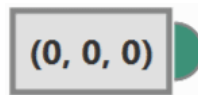


Fig. 360: **Constant Position** block

- **Output**
 - **Pin 0:** Output of the configured position.
- **Configuration menu:**

Fig. 361: **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 **Degrees ° ‘ ”**.
- 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 respect 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.

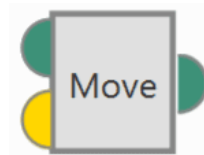


Fig. 362: **Move block**

- **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.

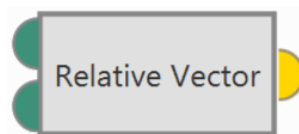


Fig. 363: **Relative Vector block**

- **Inputs**

- **Pin 0:** First position.
- **Pin 1:** Second position.

- **Output**

- **Pin 0:** 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.

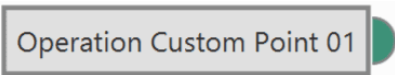


Fig. 364: **Read Feature** block

- **Output**
 - **Pin 0:** Position to read.
- **Configuration menu:**

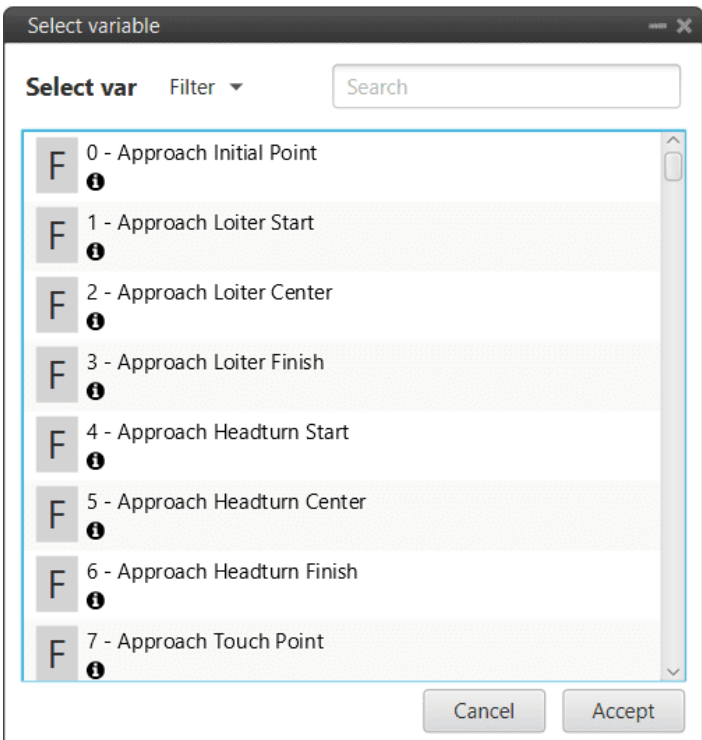


Fig. 365: **Read Feature** block configuration

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.

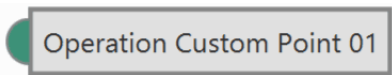
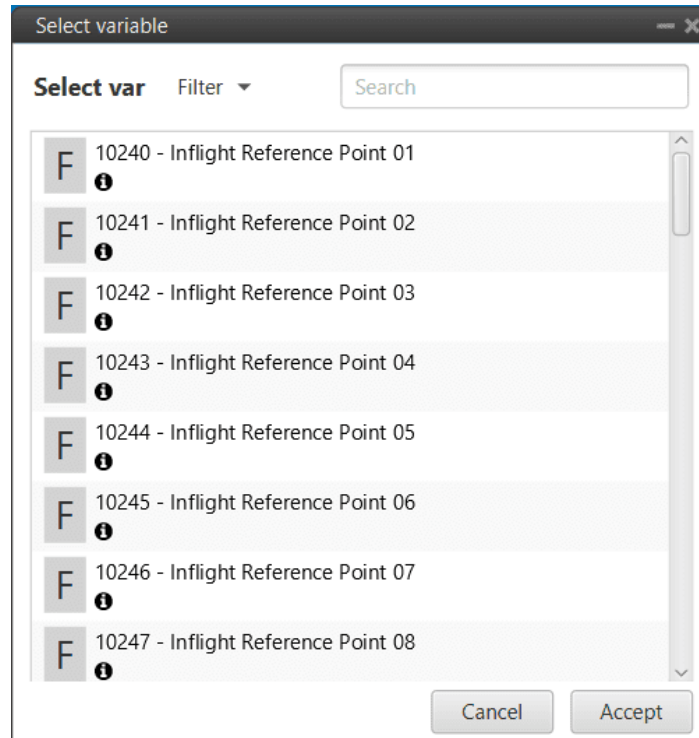


Fig. 366: **Write Feature** block

- **Input**
 - **Pin 0:** Position to write.
- **Configuration menu:**

Fig. 367: **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

Altimeter sensor block configures the parameters of external altimeters.

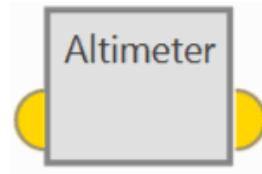


Fig. 368: Altimeter block

- **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:**

 A configuration window titled "Altimeter settings" with a standard window control bar (minimize, maximize, close). The window contains five rows of settings:

- Altitude measurement:** A dropdown menu currently showing "Lidar 1 Distance".
- Maximum variance:** A text input field containing "0.009999999776482582" followed by a unit box containing "m²".
- Minimum variance:** A text input field containing "0.009999999776482582" followed by a unit box containing "m²".
- Variance down tau:** A text input field containing "1.0" followed by a unit box containing "s".
- Sensor timeout:** A text input field containing "1.0" followed by a unit box containing "s".

 At the bottom right of the window is an "Accept" button.

Fig. 369: 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.
- **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.

2.9.12.2 GNSS sensor

GNSS sensor block configures GNSS receivers, RTK and External source.

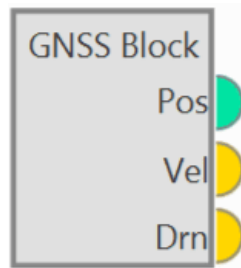


Fig. 370: GNSS sensor block

- **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. 371: **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.

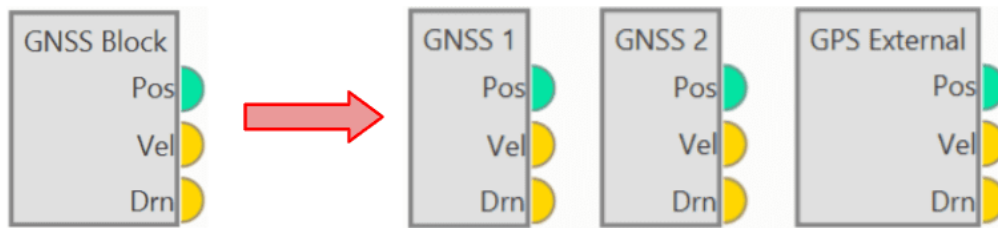


Fig. 372: GNSS sensor blocks

- **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).

The image shows a 'GNSS settings' dialog box. At the top, 'Select sensor' is set to 'GNSS 1' and 'Ublox preset' is set to 'Rover'. The 'Sensor Variance' tab is selected, showing a table of square error values and checkboxes for 'Use receiver value if present'. At the bottom, there are fields for 'Antenna position' (0.01 m, 0.0 m, 0.0 m) and 'Delay' (0.0 s), along with 'Accept' and 'Cancel' buttons.

	Square error	Use receiver value if present
Horizontal Position	1.0 m ²	<input checked="" type="checkbox"/>
Vertical Position	250.0 m ²	<input type="checkbox"/>
Horizontal Velocity	0.1 (m/s) ²	<input checked="" type="checkbox"/>
Vertical Velocity	100000.0 (m/s) ²	<input type="checkbox"/>
Relative Position	4.0 m	<input checked="" type="checkbox"/>

Antenna position: 0.01 m, 0.0 m, 0.0 m Delay: 0.0 s

Buttons: Accept, Cancel

Fig. 373: 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 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 GNSS Compass 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:

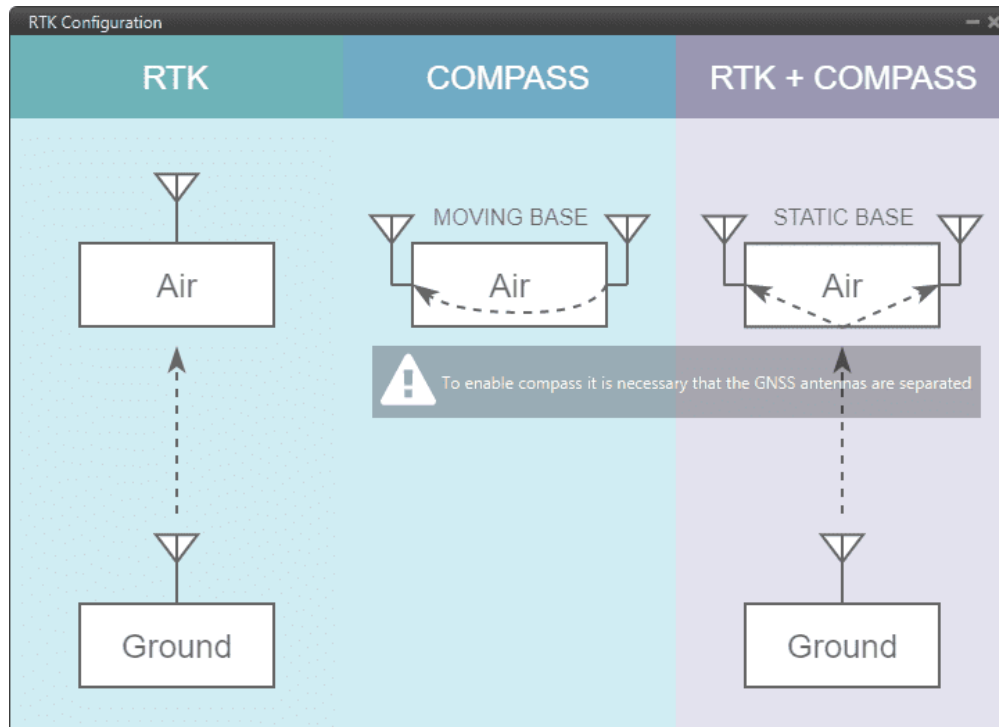


Fig. 374: GNSS sensor block configuration - RTK Wizard

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 'Rover' 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.

- * **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 receiver**.

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.

The screenshot shows the 'GNSS settings' window with the 'Configuration' tab selected. The 'Select sensor' dropdown is set to 'GNSS 1' and the 'Ublox preset' is 'Custom'. The 'RTK Wizard' button is visible in the top right. The 'Configuration' tab is active, showing the following settings:

- GNSS:**
 - Meas Rate: 0.25 s
 - ☐ Precise Point Positioning
- Survey In:**
 - ☐ Enabled
 - Minimum duration: 300.0 s
 - Position accuracy limit: 3.0 m
- SPI Port:**
 - Mask in:** ☒ ublox, ☐ nmea, ☐ rctm, ☐ rctm3
 - Mask out:** ☒ ublox, ☐ nmea, ☐ rctm3
- SCI Port:**
 - Mask in:** ☐ rctm3
 - Mask out:** ☐ rctm3

At the bottom, there are fields for 'Antenna position' (0.01 m, 0.0 m, 0.0 m) and 'Delay' (0.0 s). The text 'Compatible with Veronte 4.8' is displayed above the antenna position fields. 'Accept' and 'Cancel' buttons are at the bottom left.

Fig. 375: 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 communication 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 UBX, NMEA, RTCM and RTCM3.
- **Mask out:** Defines the outputs, i.e. sends the data (usually the ground unit). The available protocols are UBX, 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 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 disturbances, 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.



GNSS settings

Select sensor: GNSS 1 Ublox preset: Custom RTK Wizard

Configuration | **SBAS** | Message Rate | Constellations | Jamming | Advanced | Sensor Variance

☒ Automatic

120	121	122	123	124	125	126	127	128	129	130	131	132
133	134	135	136	137	138	139	140	141	142	143	144	145
146	147	148	149	150	151	152	153	154	155	156	157	158

Antenna position: 0.01 m 0.0 m 0.0 m Delay: 0.0 s

Accept Cancel

Fig. 376: 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.



The image shows a software window titled "GNSS settings" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are two dropdown menus at the top: "Select sensor" set to "GNSS 1" and "Ublox preset:" set to "Custom". To the right of these is a button labeled "RTK Wizard". Below the dropdowns is a tabbed interface with six tabs: "Configuration", "SBAS", "Message Rate" (which is currently selected and highlighted), "Constellations", "Jamming", and "Advanced". The "Message Rate" tab contains a grid of 15 controls, each with a label, a value, and a minus/plus slider. The controls are: Status (250 ms), PVT (250 ms), GPS Time (2.0 s), Survey in status (250 ms), Relative positi... (DISABLED), Time pulse (DISABLED), Jamming Status (250 ms), RTCM3 input ... (DISABLED), RTCM3 1005 (DISABLED), RTCM3 1087 (DISABLED), RTCM3 1077 (DISABLED), RTCM3 1127 (DISABLED), RTCM3 1230 (DISABLED), RTCM3 4072 S... (DISABLED), and an empty slot. At the bottom of the window, there is an "Antenna position" section with three input fields for X, Y, and Z coordinates (all set to 0.0) and a "Delay" input field (set to 0.0). Below these are "Accept" and "Cancel" buttons.

GNSS settings

Select sensor: GNSS 1 Ublox preset: Custom RTK Wizard

Configuration SBAS **Message Rate** Constellations Jamming Advanced Sensor Variance

Status 250 ms	PVT 250 ms	GPS Time 2.0 s	Survey in status 250 ms	Relative positi... DISABLED
Time pulse DISABLED	Jamming Status 250 ms	RTCM3 input ... DISABLED	RTCM3 1005 DISABLED	RTCM3 1087 DISABLED
RTCM3 1077 DISABLED	RTCM3 1127 DISABLED	RTCM3 1230 DISABLED	RTCM3 4072 S... DISABLED	

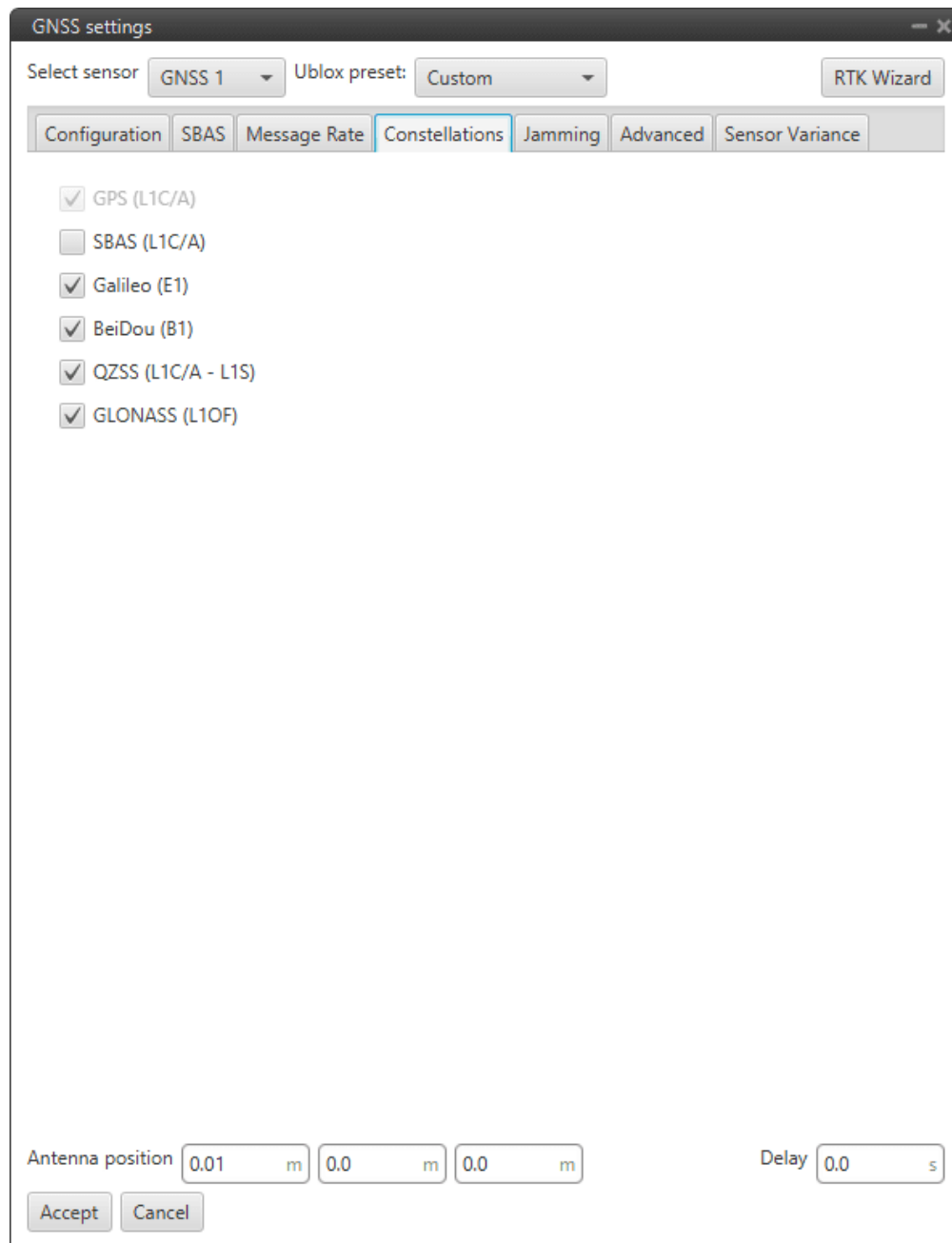
Antenna position: 0.01 m 0.0 m 0.0 m Delay: 0.0 s

Accept Cancel

Fig. 377: 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:



The image shows a software window titled "GNSS settings". At the top, there is a "Select sensor" dropdown menu set to "GNSS 1" and a "Ublox preset:" dropdown menu set to "Custom". To the right of these is a button labeled "RTK Wizard". Below the dropdowns is a horizontal tab bar with six tabs: "Configuration", "SBAS", "Message Rate", "Constellations" (which is currently selected and highlighted with a blue border), "Jamming", and "Sensor Variance". The main area of the window contains a list of satellite constellations, each with a checkbox:

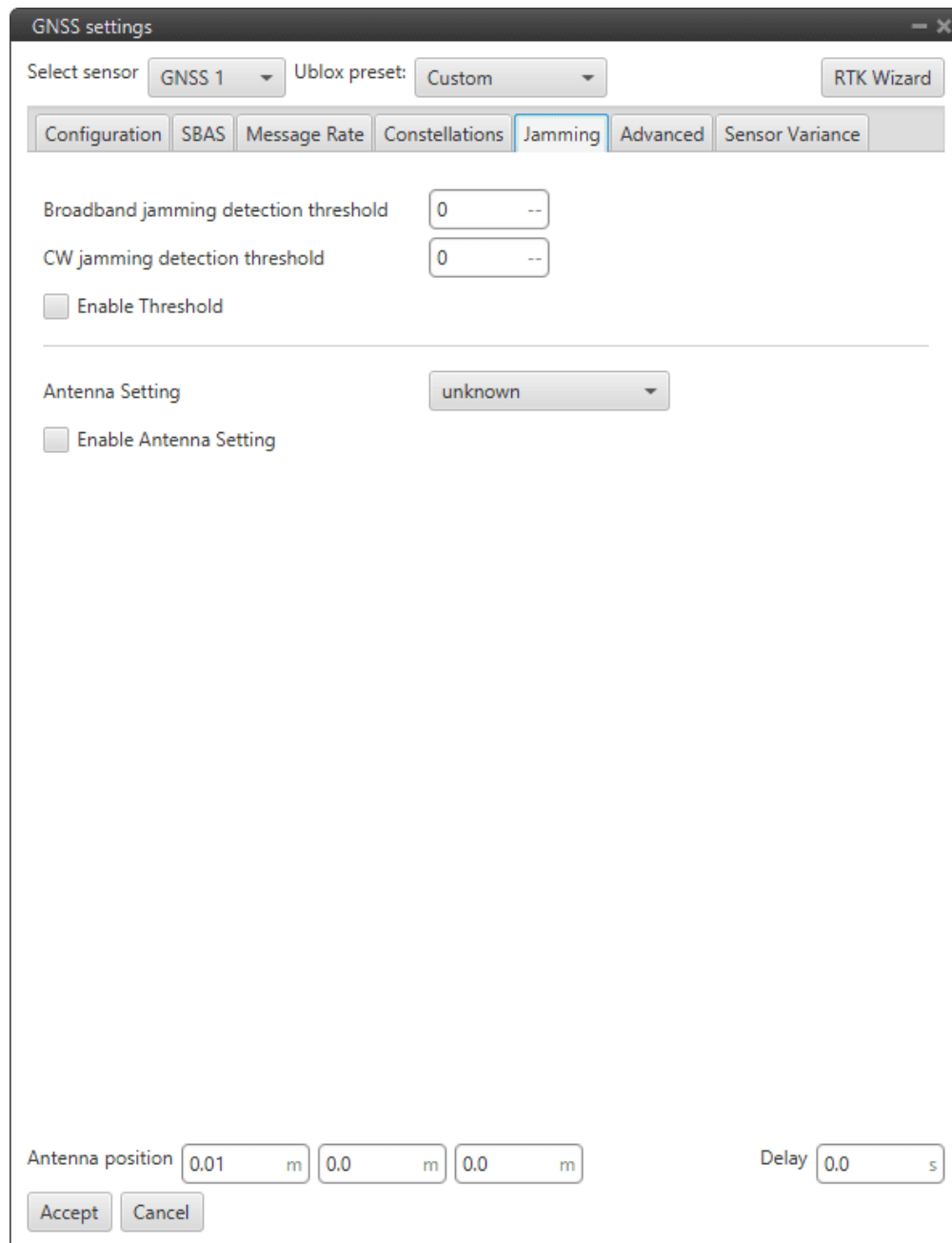
- ☒ GPS (L1C/A)
- ☐ SBAS (L1C/A)
- ☒ Galileo (E1)
- ☒ BeiDou (B1)
- ☒ QZSS (L1C/A - L1S)
- ☒ GLONASS (L1OF)

 At the bottom of the window, there is a section for "Antenna position" with three input fields containing the values "0.01", "0.0", and "0.0", each followed by a unit "m". To the right of these is a "Delay" input field containing the value "0.0" followed by a unit "s". At the very bottom are two buttons: "Accept" and "Cancel".

Fig. 378: 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.



The image shows a software window titled "GNSS settings". At the top, there is a "Select sensor" dropdown menu set to "GNSS 1" and a "Ublox preset:" dropdown menu set to "Custom". To the right of these is a button labeled "RTK Wizard". Below this is a horizontal tab bar with several tabs: "Configuration", "SBAS", "Message Rate", "Constellations", "Jamming" (which is currently selected and highlighted), "Advanced", and "Sensor Variance".

Under the "Jamming" tab, there are two input fields for detection thresholds, both currently set to "0" with "--" to their right. The first is labeled "Broadband jamming detection threshold" and the second is labeled "CW jamming detection threshold". Below these is a checkbox labeled "Enable Threshold".

Further down, there is a section for "Antenna Setting" with a dropdown menu currently showing "unknown". Below this is a checkbox labeled "Enable Antenna Setting".

At the bottom of the window, there is a section for "Antenna position" with three input fields for x, y, and z coordinates, all set to "0.0" with "m" (meters) units. To the right of these is a "Delay" input field set to "0.0" with "s" (seconds) units. At the very bottom are two buttons: "Accept" and "Cancel".

Fig. 379: 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 behaviour 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:

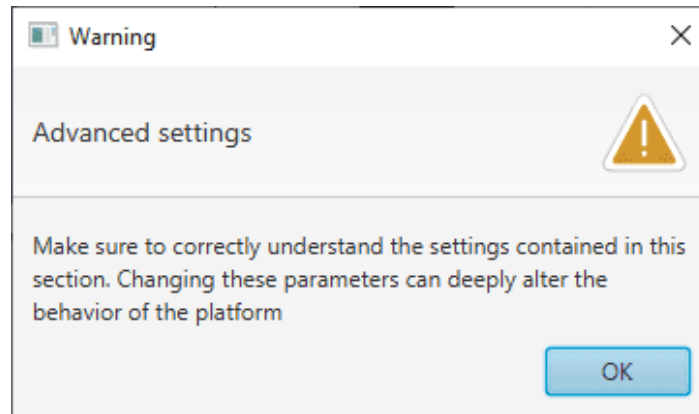


Fig. 380: GNSS 1/2 sensor block configuration - Warning advanced tab

GNSS settings

Select sensor: GNSS 1 Ublox preset: Custom RTK Wizard

Configuration SBAS Message Rate Constellations Jamming Advanced Sensor Variance

Minimum satellites number: 4 --

Maximum satellites number: 20 --

Minimum satellite elevation: 5.0 --

PDOP Mask: 250.0 --

TDOP Mask: 1000.0 --

P Acc Mask: 100.0 m

T Acc Mask: 100.0 m

Dynamic model: Airbone 4G

Antenna position: 0.01 m 0.0 m 0.0 m Delay: 0.0 s

Accept Cancel

Fig. 381: 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.

GNSS settings

Select sensor: GNSS 1 Ublox preset: Custom RTK Wizard

Configuration SBAS Message Rate Constellations Jamming Advanced **Sensor Variance**

	Square error	Use receiver value if present
Horizontal Position	1.0 m ²	<input checked="" type="checkbox"/>
Vertical Position	250.0 m ²	<input type="checkbox"/>
Horizontal Velocity	0.1 (m/s) ²	<input checked="" type="checkbox"/>
Vertical Velocity	100000.0 (m/s) ²	<input type="checkbox"/>
Relative Position	4.0 m	<input checked="" type="checkbox"/>

Antenna position: 0.01 m 0.0 m 0.0 m Delay: 0.0 s

Accept Cancel

Fig. 382: 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 (RS 232, RS485, CAN, ...) the GPS External variables of interest must be filled in this interface:

– **Configuration:**

GNSS settings

Select sensor: GPS External

Configuration | Sensor Variance

☒ Enable

Period: 0.25 s

Fix Bit: BIT Dummy Error

Time of week: Rvar Disabled

GPS Week: Rvar Disabled

☒ Enable position

GPS Position: Value used when invalid ID is...

Horizontal Position Error: 2.0 m

Vertical Position Error: 5.0 m

☒ Enable velocity

Horizontal Velocity Error: 1.0 m/s

Vertical Velocity...: 1.0 m/s

Velocity North: Rvar Disabled

Velocity East: Rvar Disabled

Velocity Down: Rvar Disabled

Antenna position: 0.0 m 0.0 m 0.0 m

Delay: 0.0 s

Accept Cancel

Fig. 383: 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.

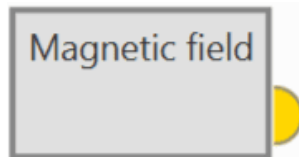
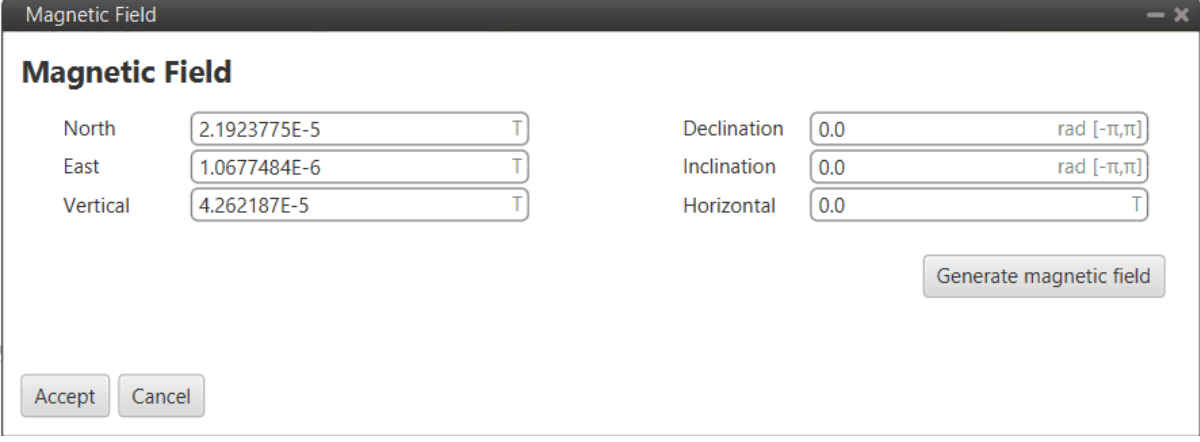


Fig. 384: **Magnetic Field** block

- **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)
- **Configuration menu:**



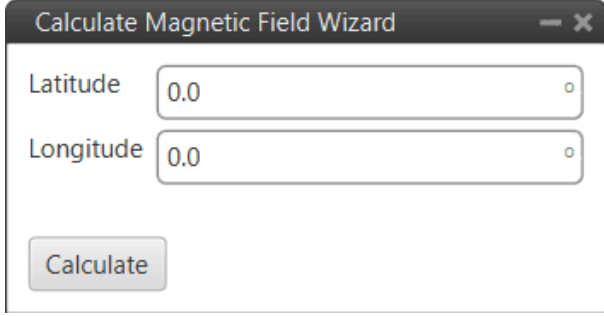
Magnetic Field

North	<input type="text" value="2.1923775E-5"/>	T	Declination	<input type="text" value="0.0"/>	rad [-π,π]
East	<input type="text" value="1.0677484E-6"/>	T	Inclination	<input type="text" value="0.0"/>	rad [-π,π]
Vertical	<input type="text" value="4.262187E-5"/>	T	Horizontal	<input type="text" value="0.0"/>	T

Fig. 385: Magnetic Field block configuration

- **North/East/Vertical:** The magnetic vector of the mission's area should be entered.
- **Declination/Inclination/Horizontal:** It is recommended to select 'Generate magnetic field' to take the magnetic declination information of the mission area.

The following window will appear to introduce the latitude and longitude of the mission area to generate the magnetic field there:



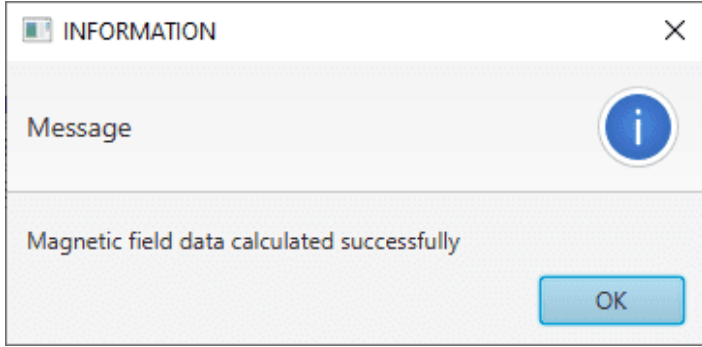
Calculate Magnetic Field Wizard

Latitude


Longitude

Fig. 386: Magnetic Field block configuration - Generate magnetic field

When generated, this message will appear and the configuration parameters will be automatically updated with the calculated values:



INFORMATION

Message 

Magnetic field data calculated successfully

Fig. 387: Magnetic Field block configuration - Generate magnetic field message

2.9.12.4 Magnetometer

Magnetometer sensor block returns the magnetic field being read by the selected sensor in the body frame.

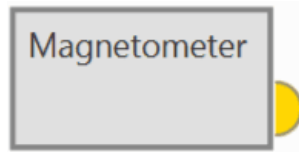


Fig. 388: **Magnetometer** block

- **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:**

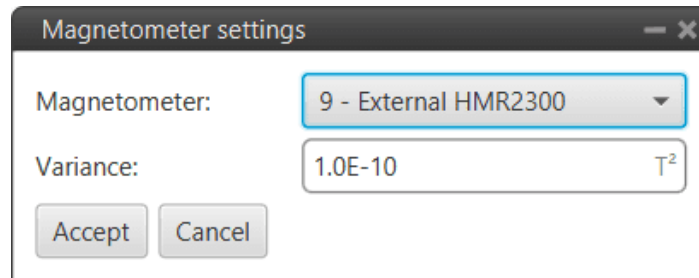


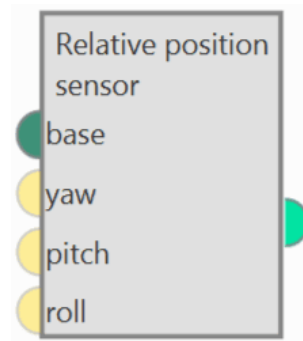
Fig. 389: **Magnetometer** block configuration

- **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 independently from the Magnetometer selected.

2.9.12.5 Relative position

Relative position sensor block works in conjunction with the [Interne configuration section](#) to configure the **Interne system** as an ultrasound sensor that calculates the position of Veronte Autopilot 1x.

Fig. 390: **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:**

Relative Position Sensor settings

Increment of horizontal variance with distance: 0.0

By user Horizontal sensor variance: 0.009999999776482582 m²

Increment of vertical variance with distance: 0.0

By device Vertical sensor variance: 0.009999999776482582 m²

Distance to mass center:

x 1.0 -- y 2.0 -- z 3.0 --

☐ Invert measurements (multiply them by -1)

Accept

Fig. 391: **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 interest position in xy/z planes.

Note: If the option *By device* is selected, these parameters are automatically set by the autopilot.

- **x/y/z:** Defines the distance between the Internet 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:

$$\text{square error} = \text{position error} + \text{increment error} \cdot \text{distance}$$

- *position error* : Horizontal/Vertical sensor variance position
- *increment error* : Increment of horizontal/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.

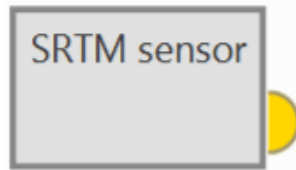


Fig. 392: SRTM height block

• 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:

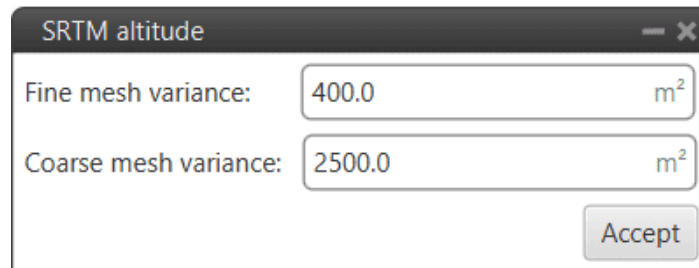


Fig. 393: SRTM height block configuration

- **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.

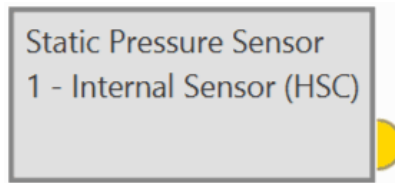


Fig. 394: **Static Pressure** block

- **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:**

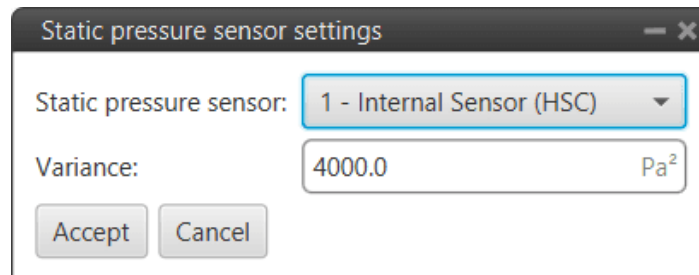


Fig. 395: **Static Pressure** block configuration

- **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.

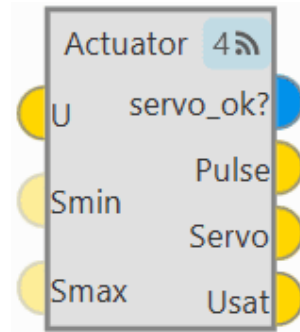


Fig. 396: **Actuator** block

- **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:**

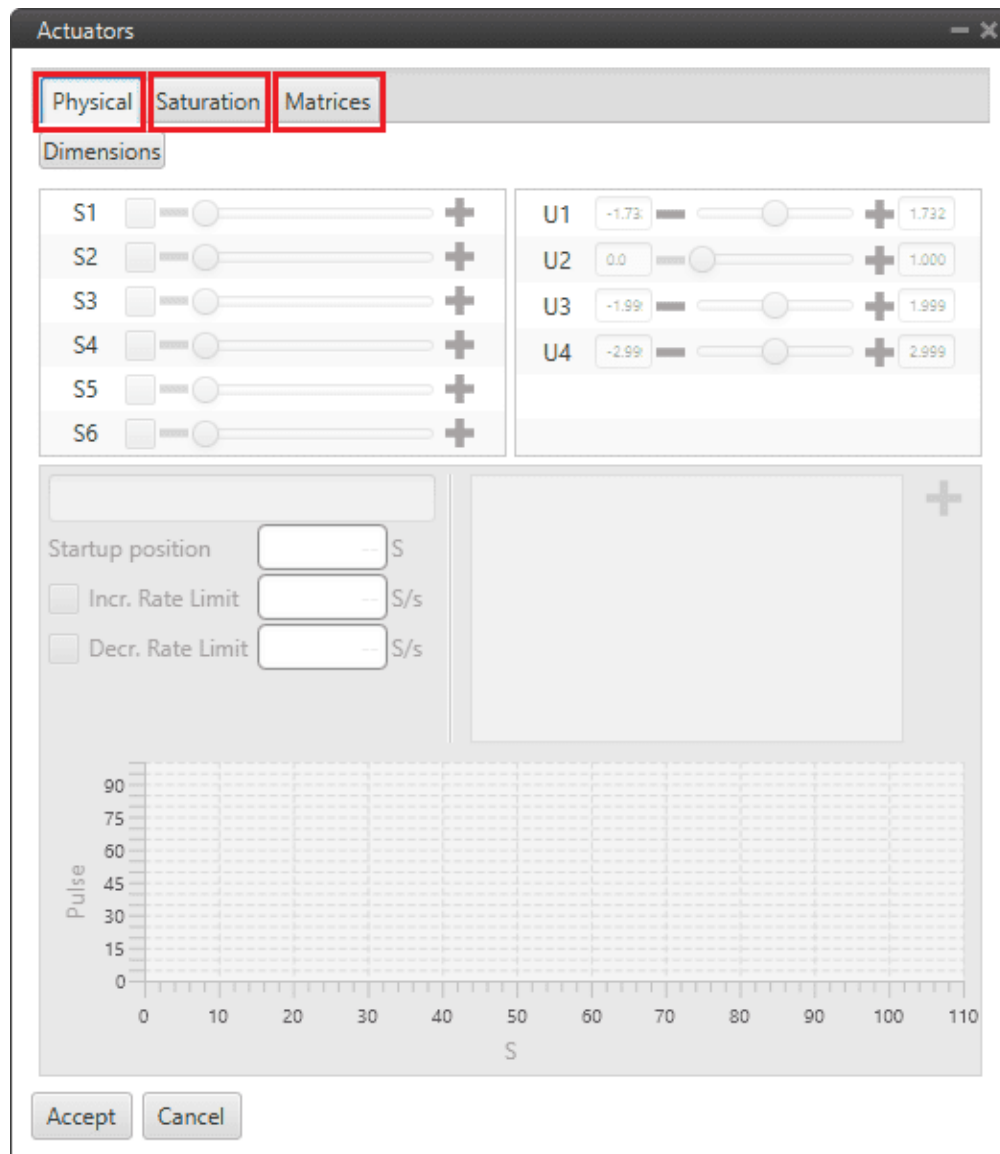


Fig. 397: Actuator block configuration

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. 398: Actuator block configuration - Physical

1. **Dimensions:** Set the number of servos and control outputs.



Fig. 399: 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.
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**.

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, I S position corresponds to a 100 % pulse to be sent to the corresponding servo (*Motor I*).

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 behaviour of the platform when one or more of its actuators is/are in saturation state.

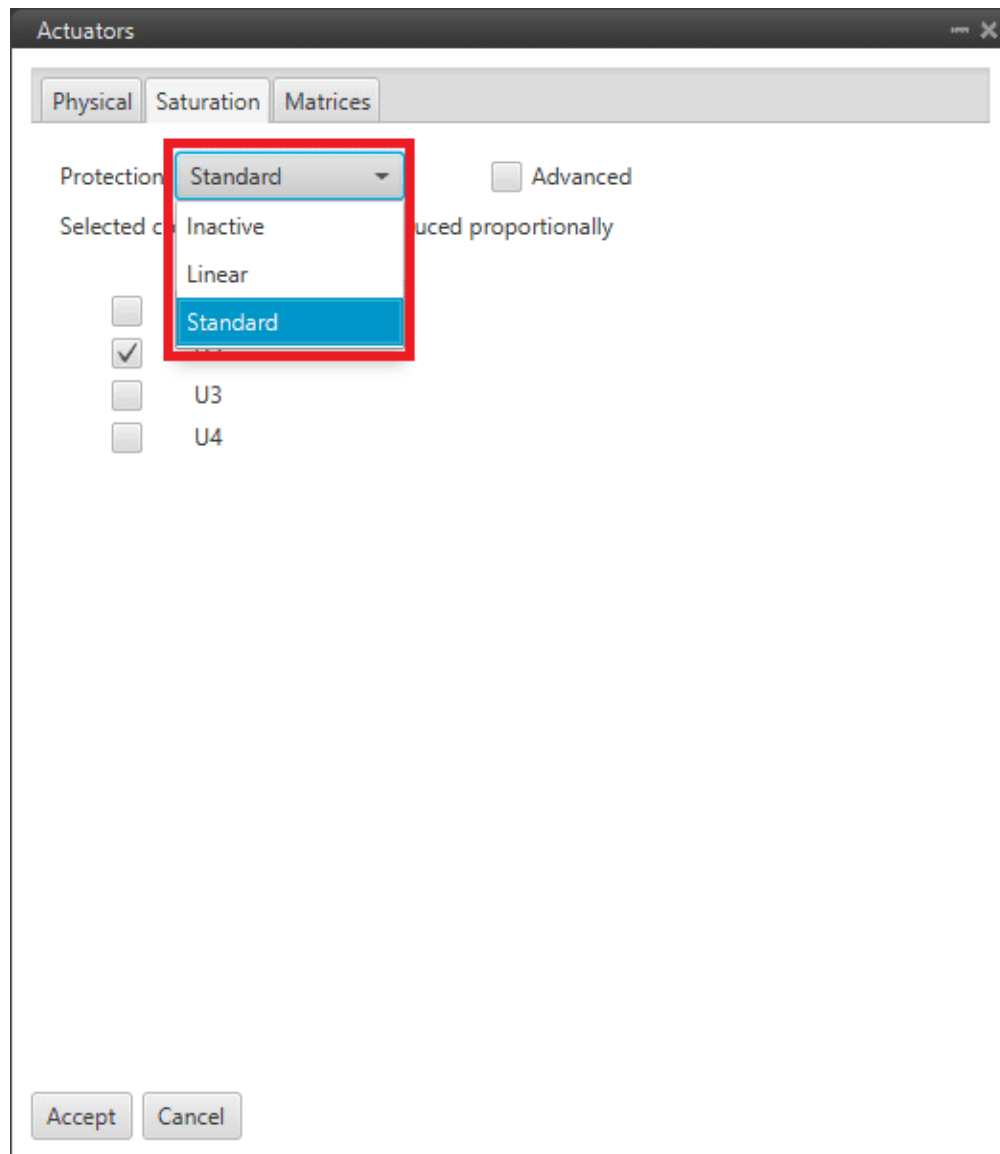


Fig. 400: Actuator block configuration - Saturation

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).

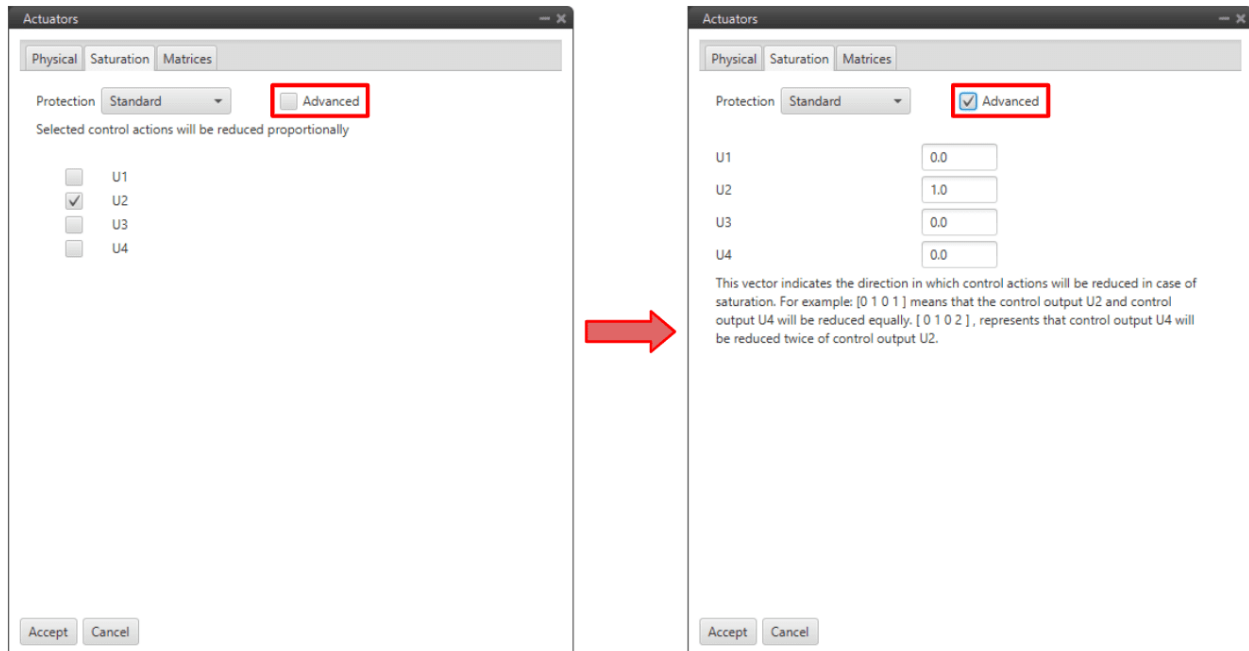


Fig. 401: Actuator block configuration - Saturation Standard

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.

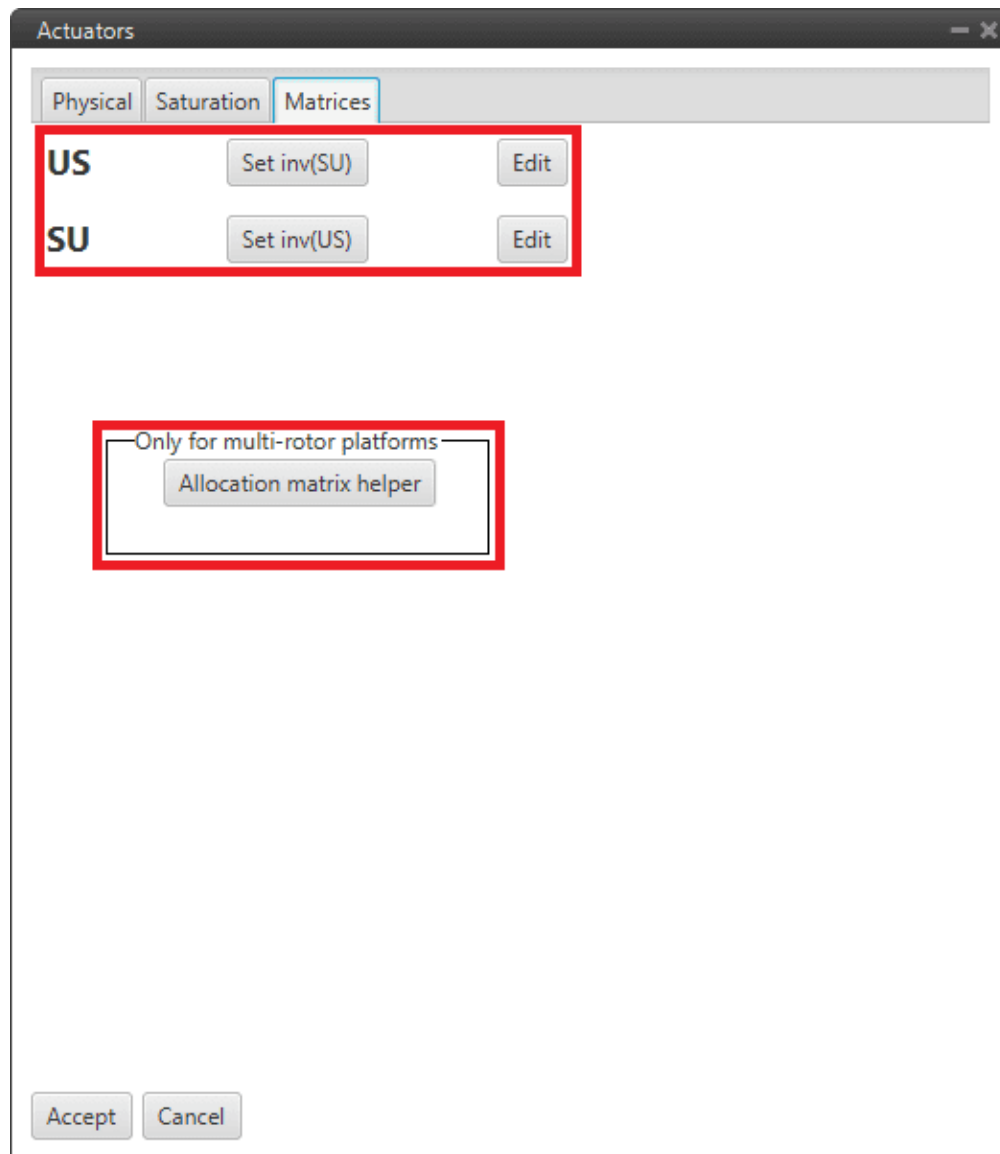


Fig. 402: Actuator block configuration - Matrices

- * 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 actuator outputs S on the rows. Clicking on the '+' sign allows the user to add a new U or S , by adding a new column, a row will appear and vice versa.

		U1	U2	U3	U4
-	Motor 1	-0.28867513	1.0	-0.16667	0.16667
-	Motor 2	0.0	1.0	-0.33334	-0.16667
-	Motor 3	0.28867513	1.0	-0.16667	0.16667
-	Motor 4	0.28867513	1.0	0.16667	-0.16667
-	Motor 5	0.0	1.0	0.33334	0.16667
-	Motor 6	-0.28867513	1.0	0.16667	-0.16667
		+			

Apply

Fig. 403: 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:

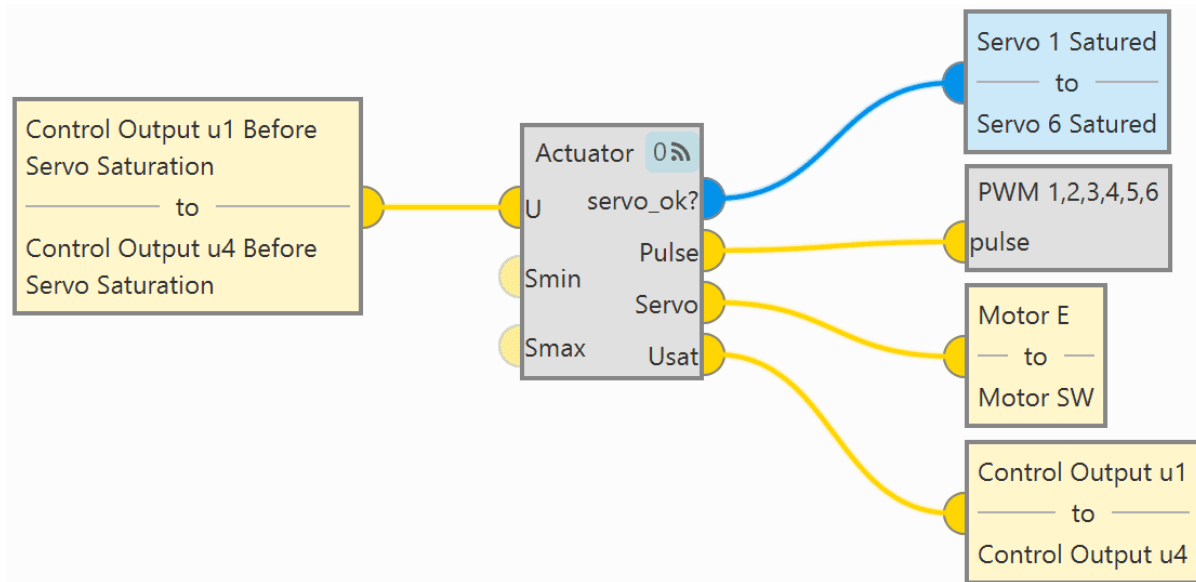


Fig. 404: Actuator block - Example of use

2.9.13.2 Arc Trim

Arcade stick trim block is used to set the zero-stick position for the *Arcade Mode*.

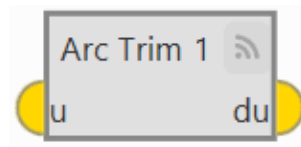


Fig. 405: Arc Trim block

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.

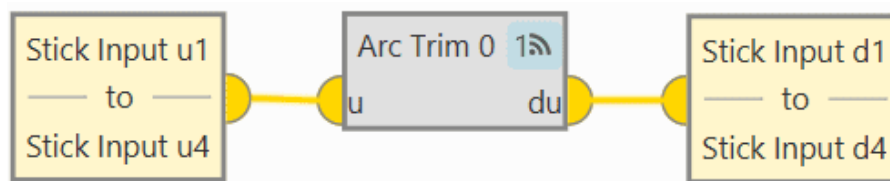
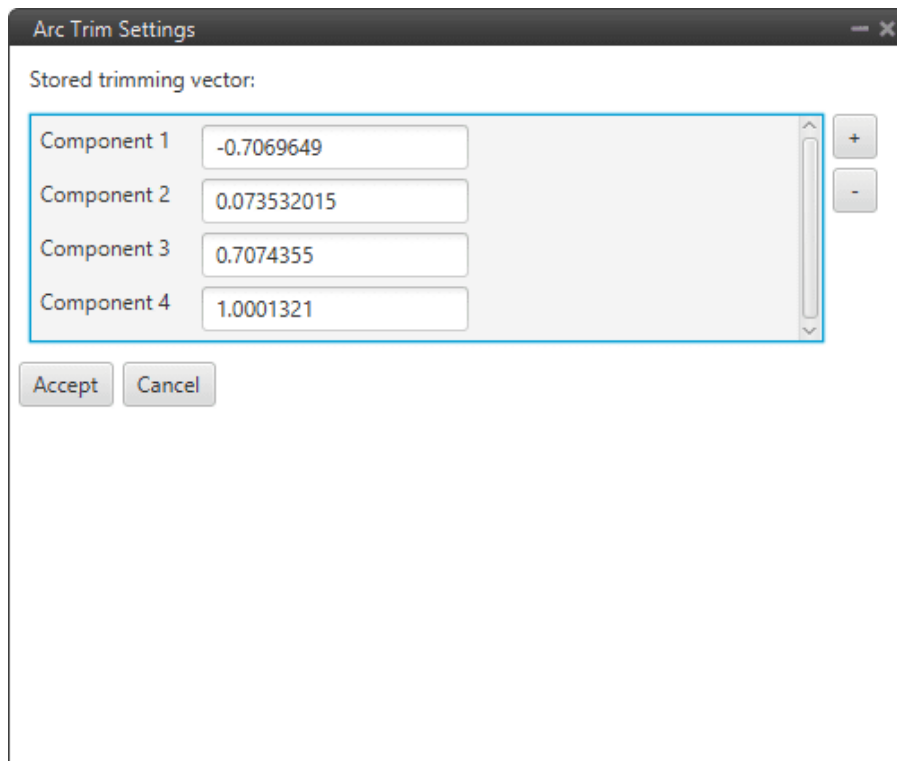


Fig. 406: Arc Trim block - Example of use

- **Input**
 - **u**: Input stick vector to trim.
- **Output**
 - **du**: Trimmed stick vector.

- **Configuration menu:**



Arc Trim Settings

Stored trimming vector:

Component 1	-0.7069649
Component 2	0.073532015
Component 3	0.7074355
Component 4	1.0001321

Accept Cancel

Fig. 407: 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 [Arcade trim automation](#) section.

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.

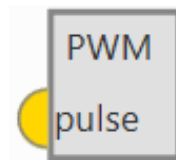


Fig. 408: PWM block

- **Input**

● **pulse:** Input vector of pulses to apply.

- Configuration menu:

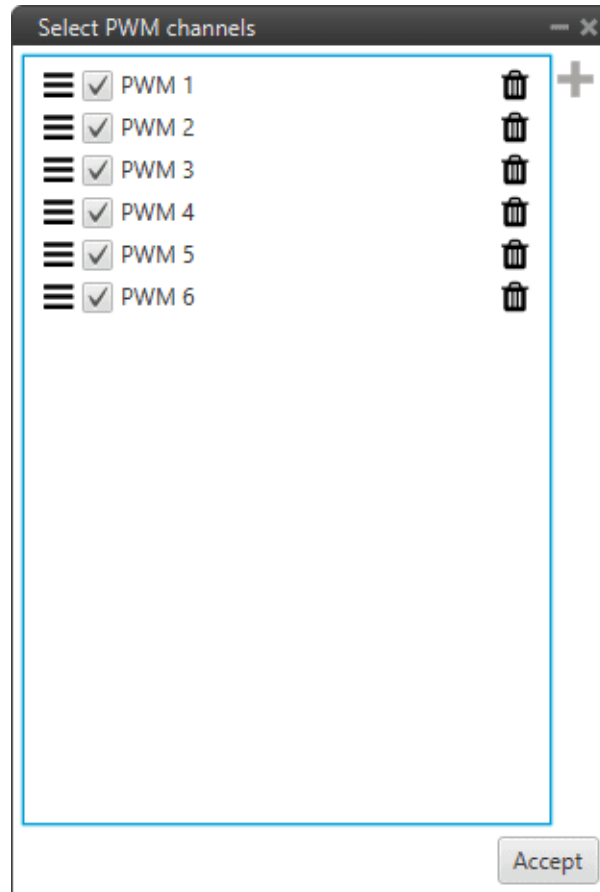





Fig. 409: PWM block configuration

-  : 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 *PWM - Servos* of the **Integration examples** section.

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. 410: 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. 411: 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 Bound

Bound block limits the input signal and produces a bit to indicate if it was within the allowed range.

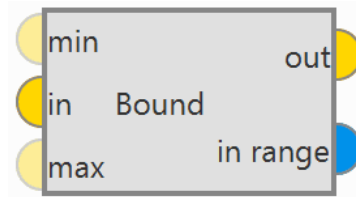


Fig. 412: **Bound** block

- **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.

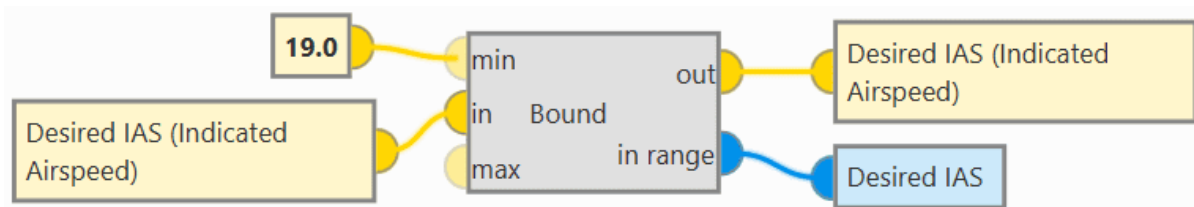


Fig. 413: **Bound** block - Example of use

2.9.14.3 EWMA Tau filter

EWMA (Exponentially Weighted Moving Average) Tau filter block is a simple first order filter with configurable time constant τ .

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 ($t=0$)

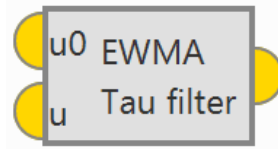


Fig. 414: EWMA Tau filter block

- **Inputs**
 - **u0**: Initialization value (set in on_focus).
 - **u**: Current value to filter.
- **Output**
 - **Pin 0**: Filtered value.
- **Configuration menu:**



Fig. 415: EWMA Tau filter block configuration

- **Tau**: The time constant Tau must be entered.

2.9.14.4 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.

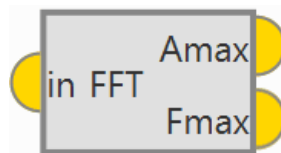


Fig. 416: FFT block

- **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:**

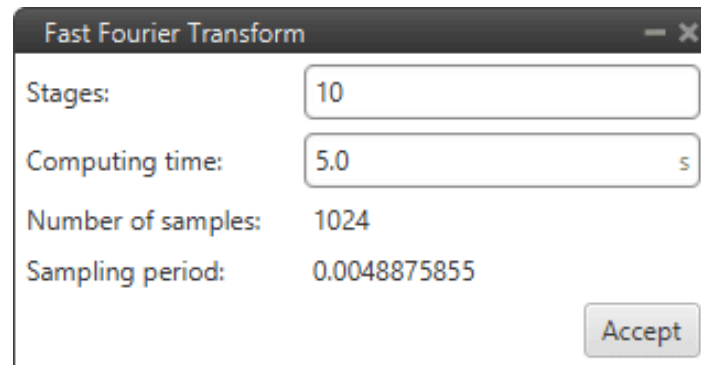


Fig. 417: FFT block configuration

- Stages.
- Computing time.

2.9.14.5 Hysteresis

Hysteresis block applies hysteresis to input signal to prevent changes in output signal when the input is close to zero.

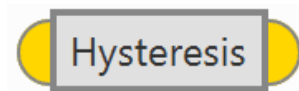


Fig. 418: Hysteresis block

The behaviour is as shown in the following diagram:

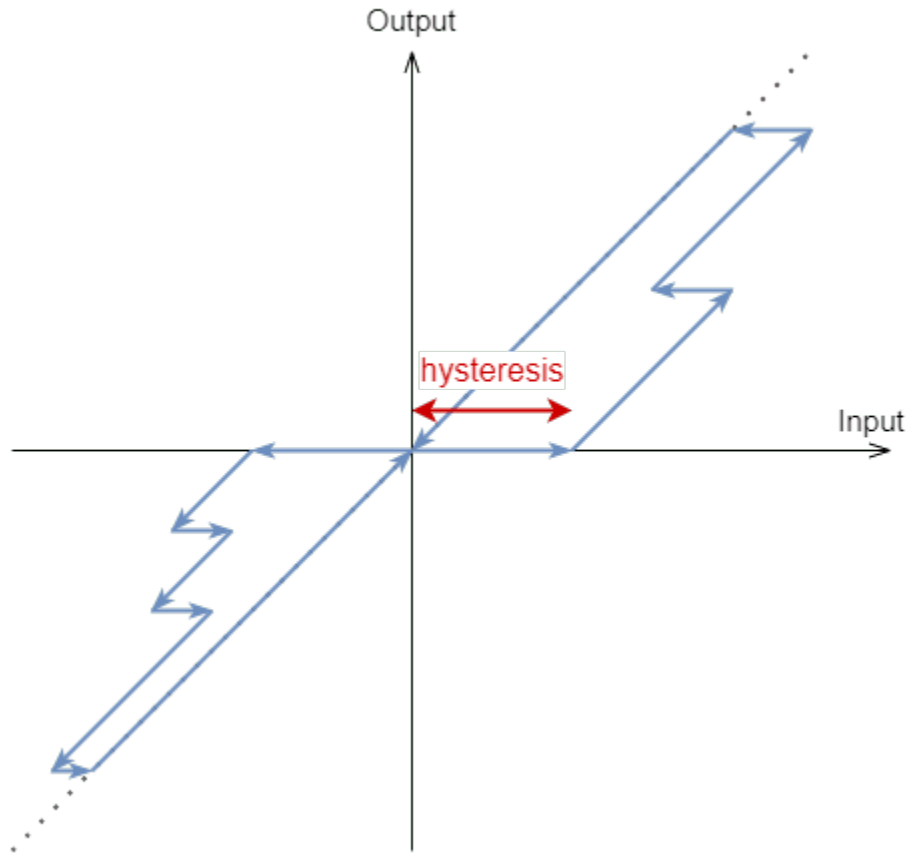


Fig. 419: Hysteresis diagram

- **Input**
 - **Pin 0:** Input signal.
- **Output**
 - **Pin 0:** Output signal.
- **Configuration menu:**

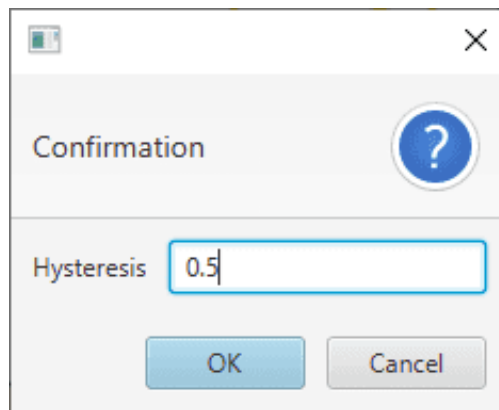


Fig. 420: Hysteresis block configuration

- **Hysteresis:** Users must enter the magnitude of the hysteresis.

2.9.14.6 IIR Filter

IIR Filter block allows the user to define an Infinite Input Response filter, it applies a **Z-transform**.



Fig. 421: IIR Filter block

- **Input**
 - **Pin 0:** Input signal.
- **Output**
 - **Pin 0:** Output signal.
- **Configuration menu:**

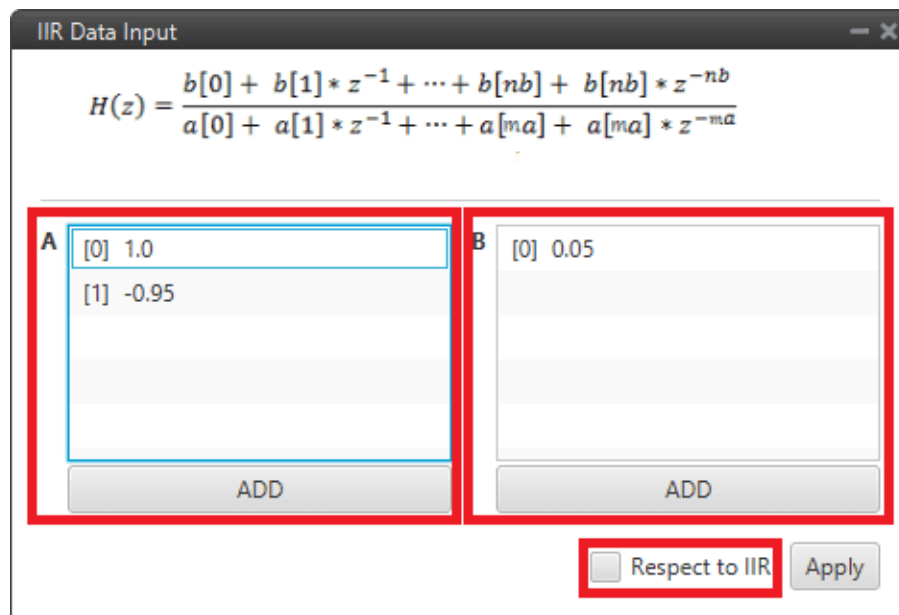


Fig. 422: IIR Filter block configuration

- **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:

IIR Data Input

$$H(z) = \frac{b[0] + b[1] * z^{-1} + \dots + b[nb] + b[nb] * z^{-nb}}{a[0] + a[1] * z^{-1} + \dots + a[ma] + a[ma] * z^{-ma}}$$

A

[0]	1.0
[1]	-0.999

B

[0]	0.001
[1]	-0.001

ADD ADD

☐ Respect to IIR Apply

Fig. 423: IIR Filter block - Derivative example

2.9.14.7 Interpolation Vector

Interpolation Vector block applies the configured table interpolation over each of the components of the input vector.

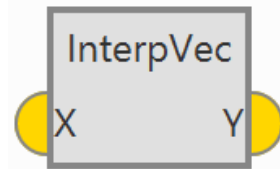


Fig. 424: Interp Vec block

- **Input**
 - X: Input vector.
- **Output**
 - Y: Interpolated output vector.
- **Configuration menu:**

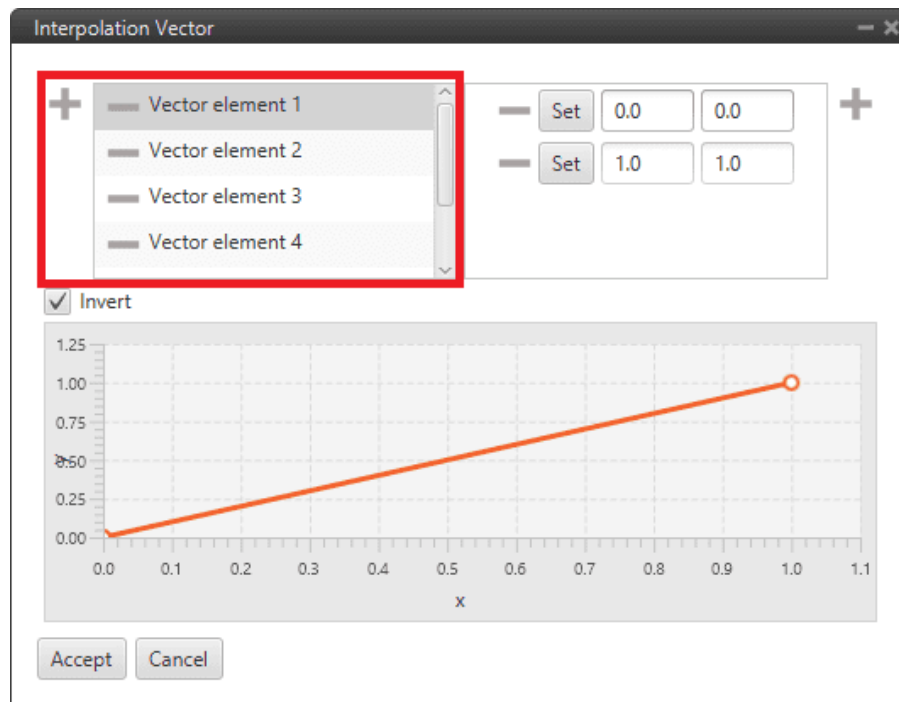


Fig. 425: Interp Vec block configuration

- **+**: 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.

2.9.14.8 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**.

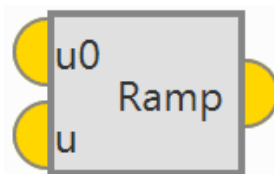


Fig. 426: Ramp block

- **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:

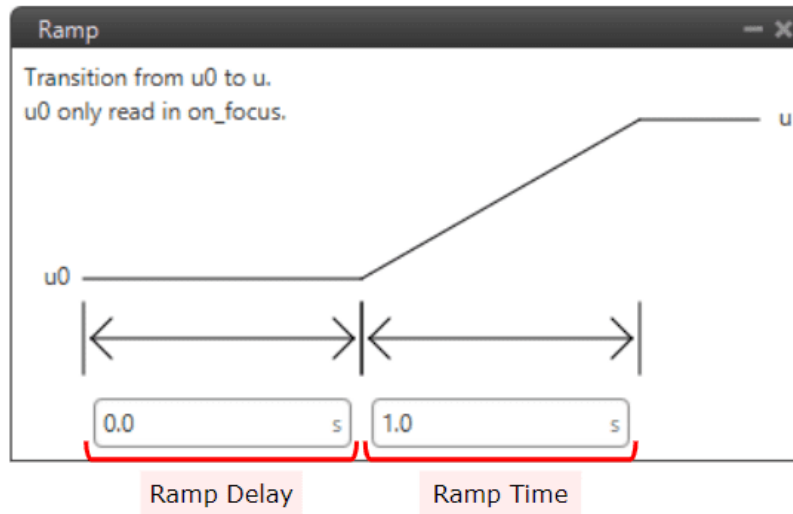


Fig. 427: Ramp block configuration

- **Ramp Delay:** Time before starting the ramp.
- **Ramp Time:** Time in which the variable must change from the initial value to the final value.

2.9.14.9 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.

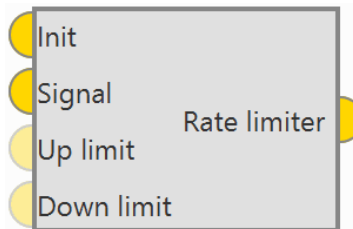


Fig. 428: Rate limiter block

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. 429: Rate limiter block configuration

- **Angle wrap**: Perform a $[-\pi, \pi]$ wrap. It should be enable when using angles.

This block can be used to avoid instantaneous spikes and spikes in control signals, effectively reducing control noise and smoothing flight:

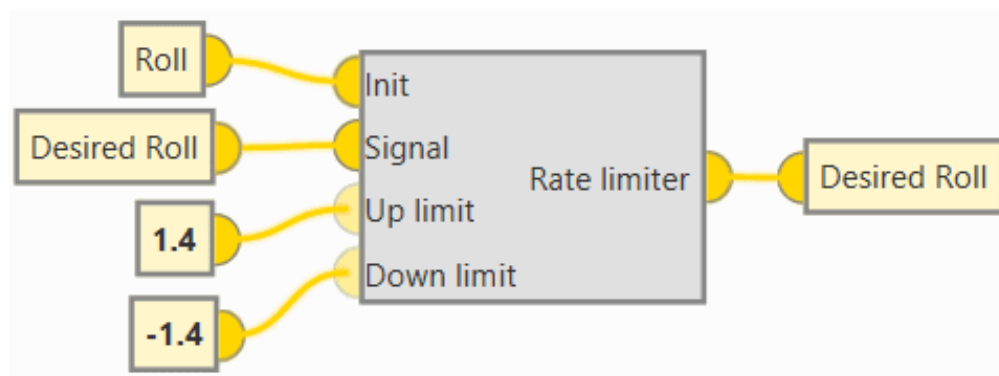


Fig. 430: Rate limiter block - Example of use

2.9.14.10 Signal generator

Signal generator block is a wave signal generator.

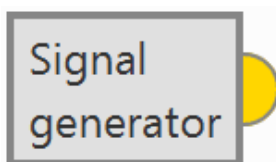


Fig. 431: Signal generator block

- **Output**
 - **Pin 0**: Signal generated according to the configured type.
- **Configuration menu:**

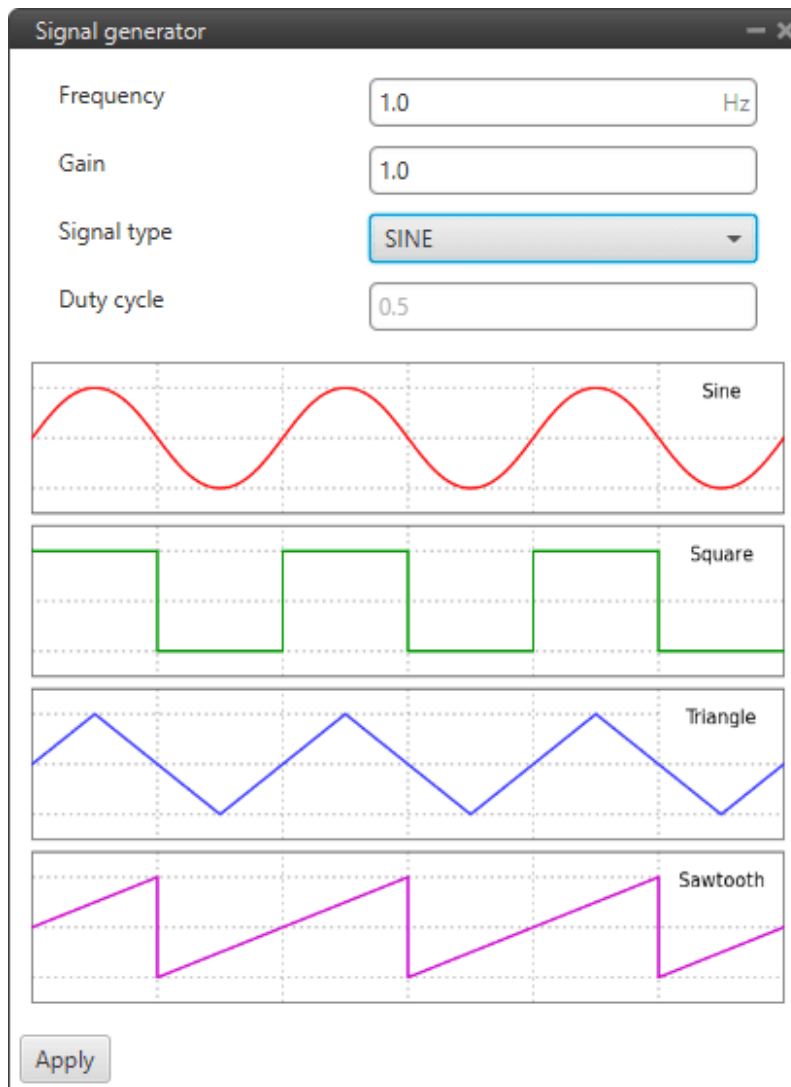


Fig. 432: Signal generator block configuration

- **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**, **Sqaure**, **Triangle** and **Sawtooth**.

Note: An example of the shape of each type of signal is shown below.

- **Duty cycle:** Duty cycle of the signal. Can **only** be modified when **Square signal** is selected.

An example is given below:

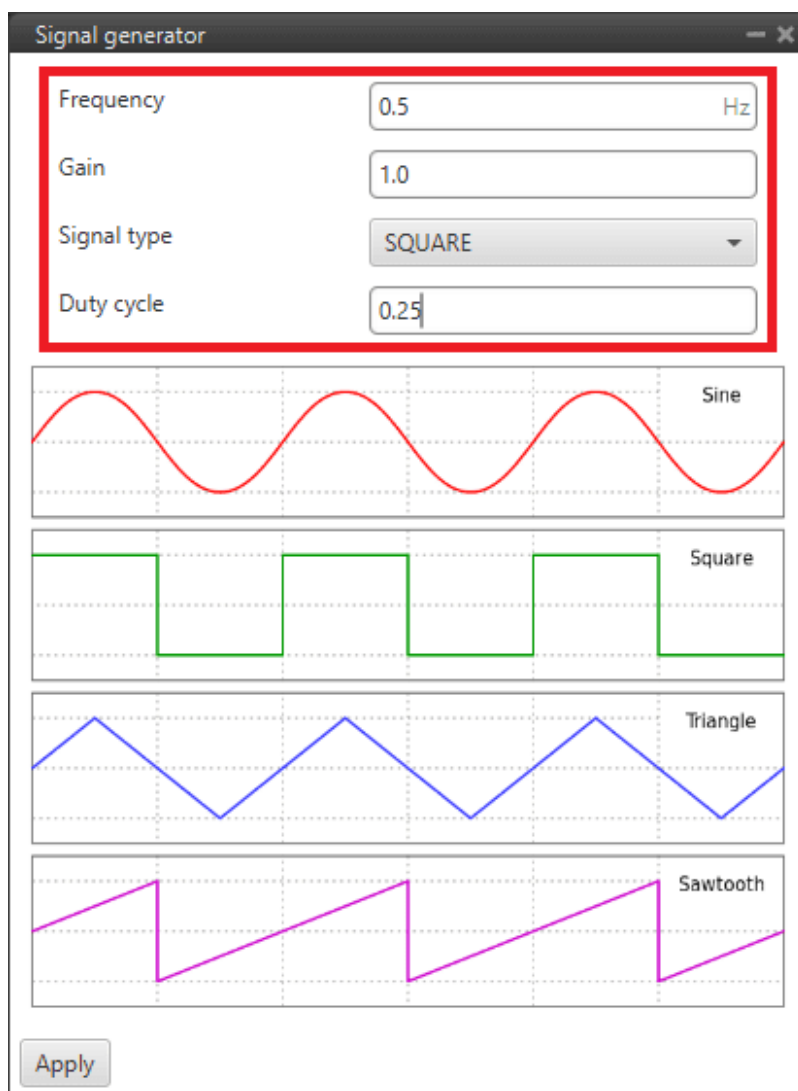


Fig. 433: Signal generator block - Configuration example

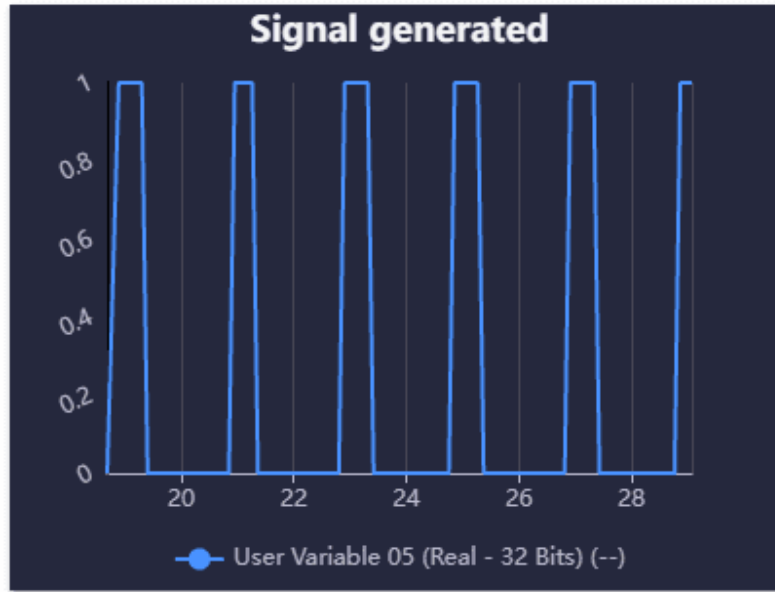


Fig. 434: Signal generator block - Example of signal generated

2.9.15 Type Casting blocks

These blocks allow to change from one data type to another. There are four different blocks available:

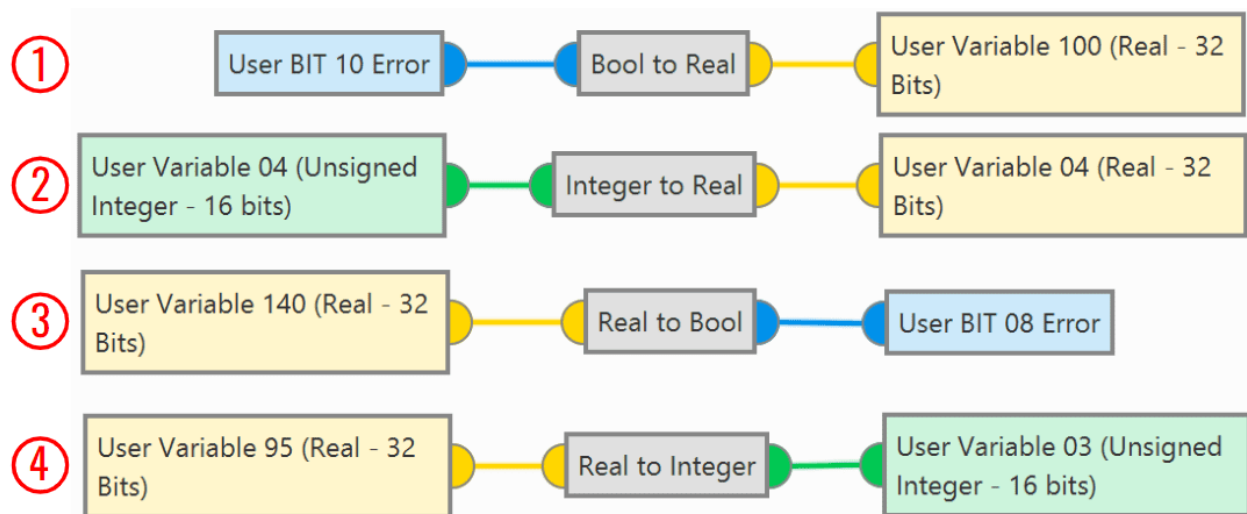


Fig. 435: Type Casting blocks

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 section will allow the user to configure different parameters from the available variety of payloads.

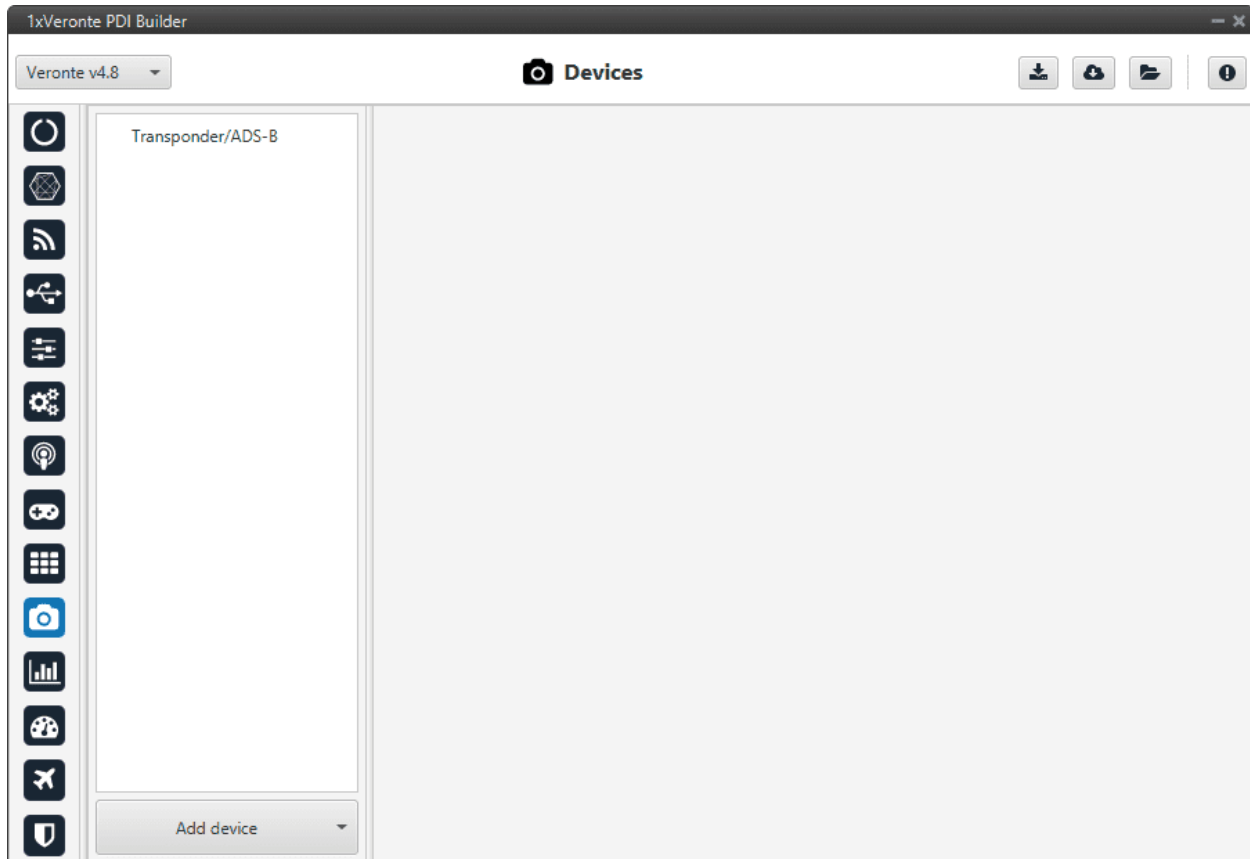


Fig. 436: **Devices menu**

By default, only the *Transponder/ADS-B* device is added to this menu (as shown in the figure above). However, users can add other devices supported by Veronte Autopilot 1x simply by clicking **Add device**:

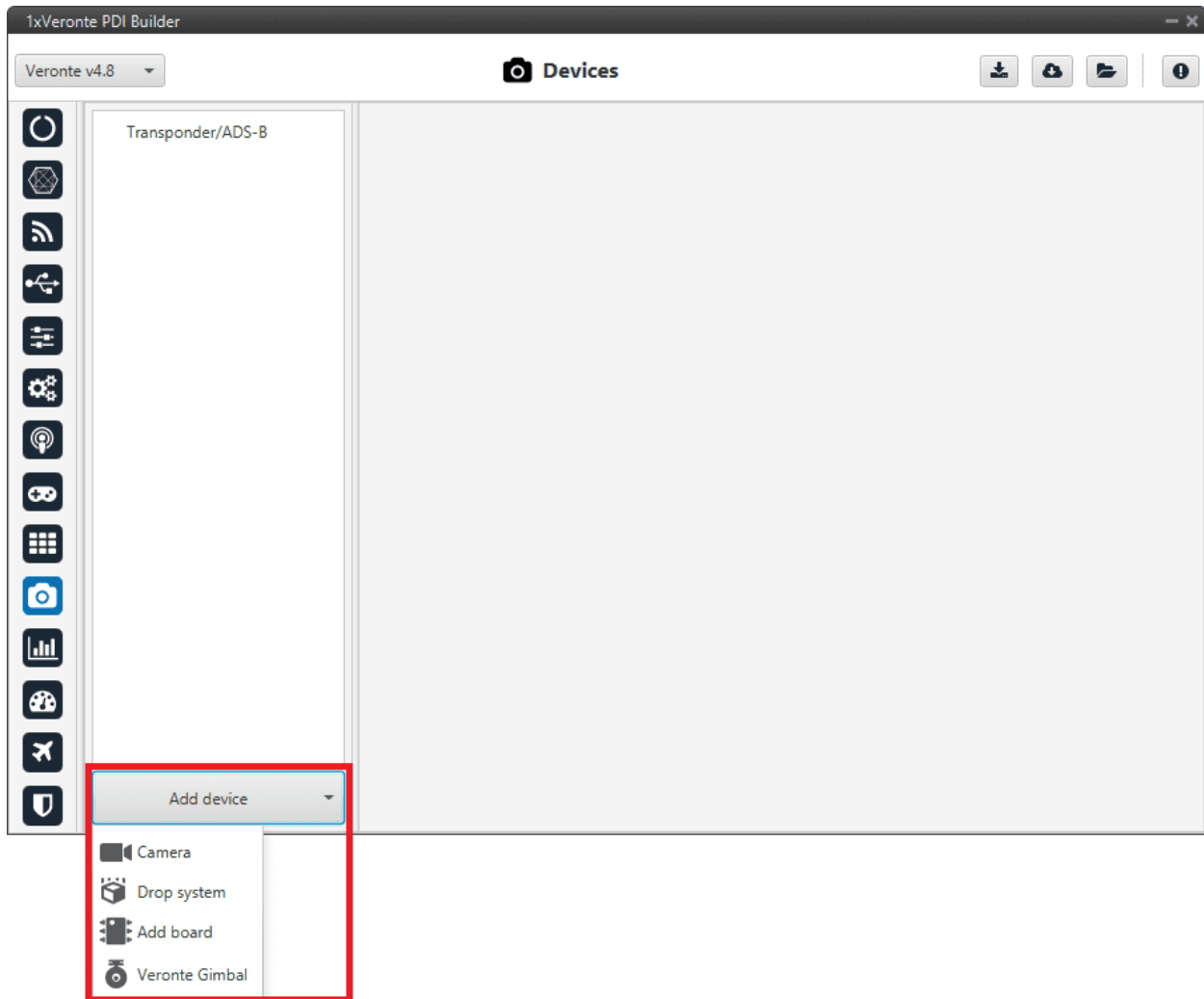


Fig. 437: Add devices

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**.

There are 2 different types of ADS-B:

- **ADS-B In:** When the transponder sends the detected UAVs.
- **ADS-B Out:** When the transponder wants to send the position of the UAV.

The interface includes both types and, depending on the model (type) selected, the configurable parameters for ADS-B In or ADS-B Out are enabled.

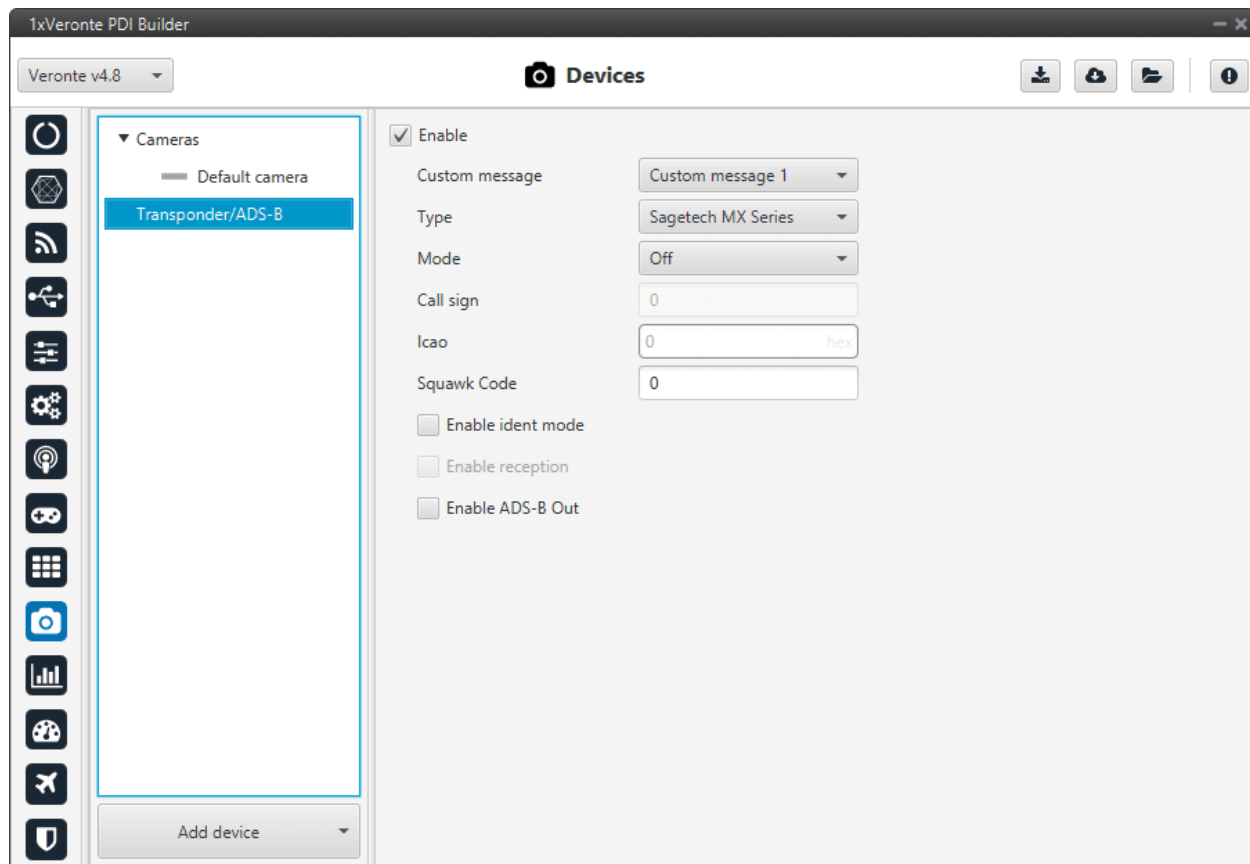


Fig. 438: Transponder/ADS-B section

The parameters that appear in this menu and that must be configured by the user are explained below:

- **Enable:** Check it to activate a transponder device.
- **Custom message:** Select the *Custom message 1-3* to be used for the information exchange, as it will be automatically filled with the information required by the transponder/ADS-B.
- **Type:** Veronte Autopilot 1x is compatible with:
 - Sagotech MX Series
 - uAvionix ping 20s
 - uAvionix ping 1090i
 - Daedalean
 - Sagotech XPS-TR
 - Sagotech XPG-TR
 - Sagotech XPS-TRB
- **Mode:** The available modes are:
 - **Off:** Transponder switched off.
 - **Standby:** Transponder will not respond to interrogation.
 - **ON:** Replies to interrogations with **4-digit squawk code**.

- **ALT:** Replies to interrogation with **altitude information**.

Important: The parameters detailed above are common for ADS-B In and Out, the following depends on the selected *Type*.

- **Call sign:** Communication call sign assigned as unique identifier to the aircraft.
- **Icao:** Unique ICAO 24-bit address permanent for the aircraft, which becomes a part of the aircraft's Certificate of Registration.

It is represented by six **hexadecimal** characters.

Note: Only available for *ALT mode*.

- **Squawk Code:** Users must introduce the Squawk Code provided. This is the transponder code to in order to identify the flight. For certain type of flights and/or situations, specific transponder codes are used.

These codes are four/digital octal numbers.

- **Enable Ident Mode:** This is an identification of the UAV **at the request of ATC**, in order to help them to locate the aircraft.

Note: Only available for *ON* and *ALT modes*.

- **Enable reception:** Enables reception (ADS-B IN).
- **Enable ADS-B Out:** Enables transmission (ADS-B OUT).

2.10.2 Cameras

Adding a camera will create a default Camera configuration menu.

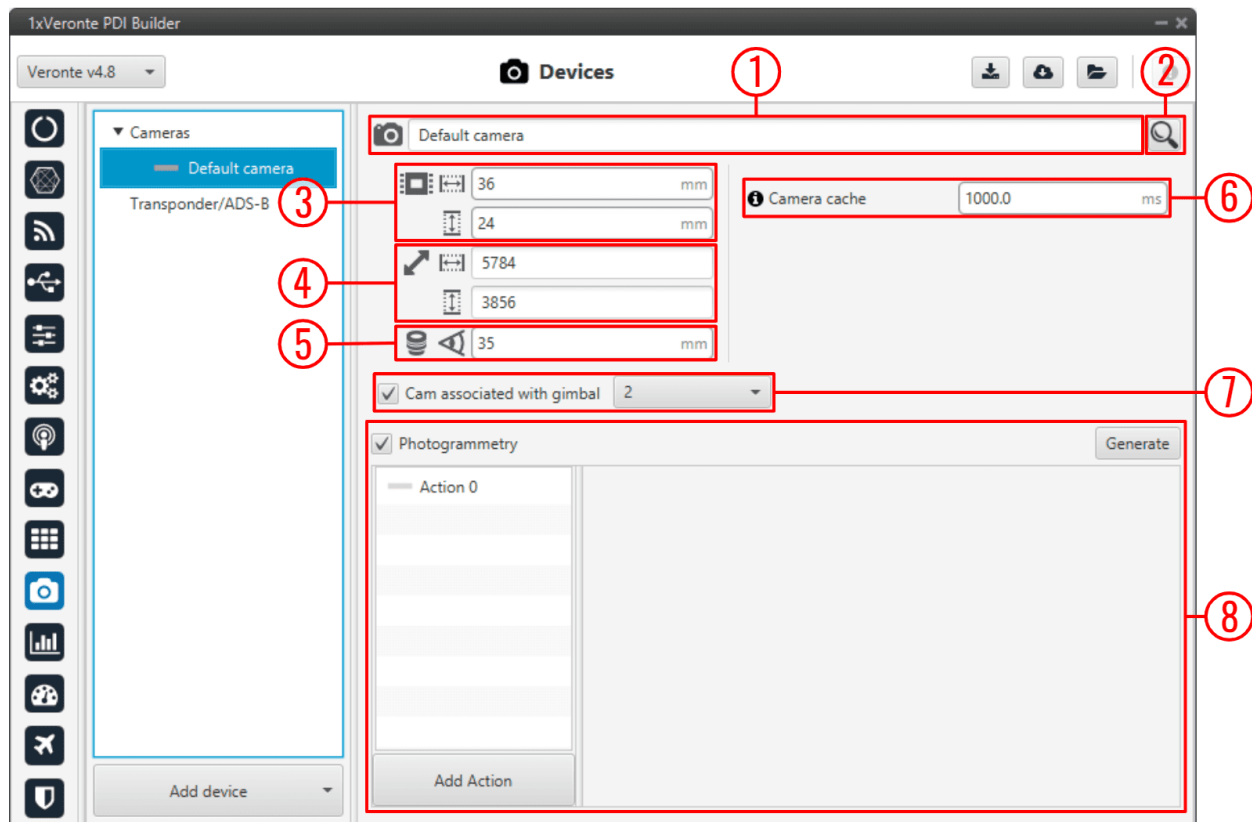


Fig. 439: Cameras section

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 **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*.

- **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:

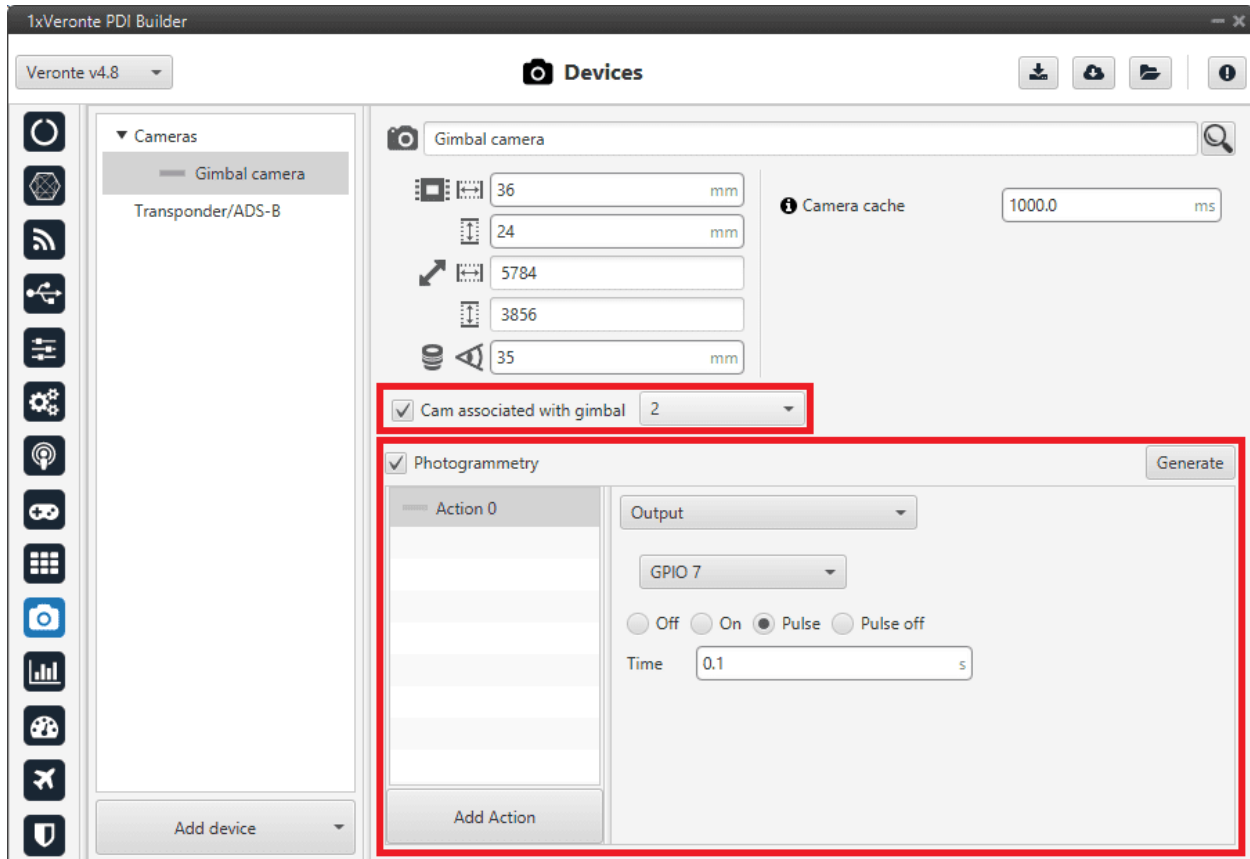
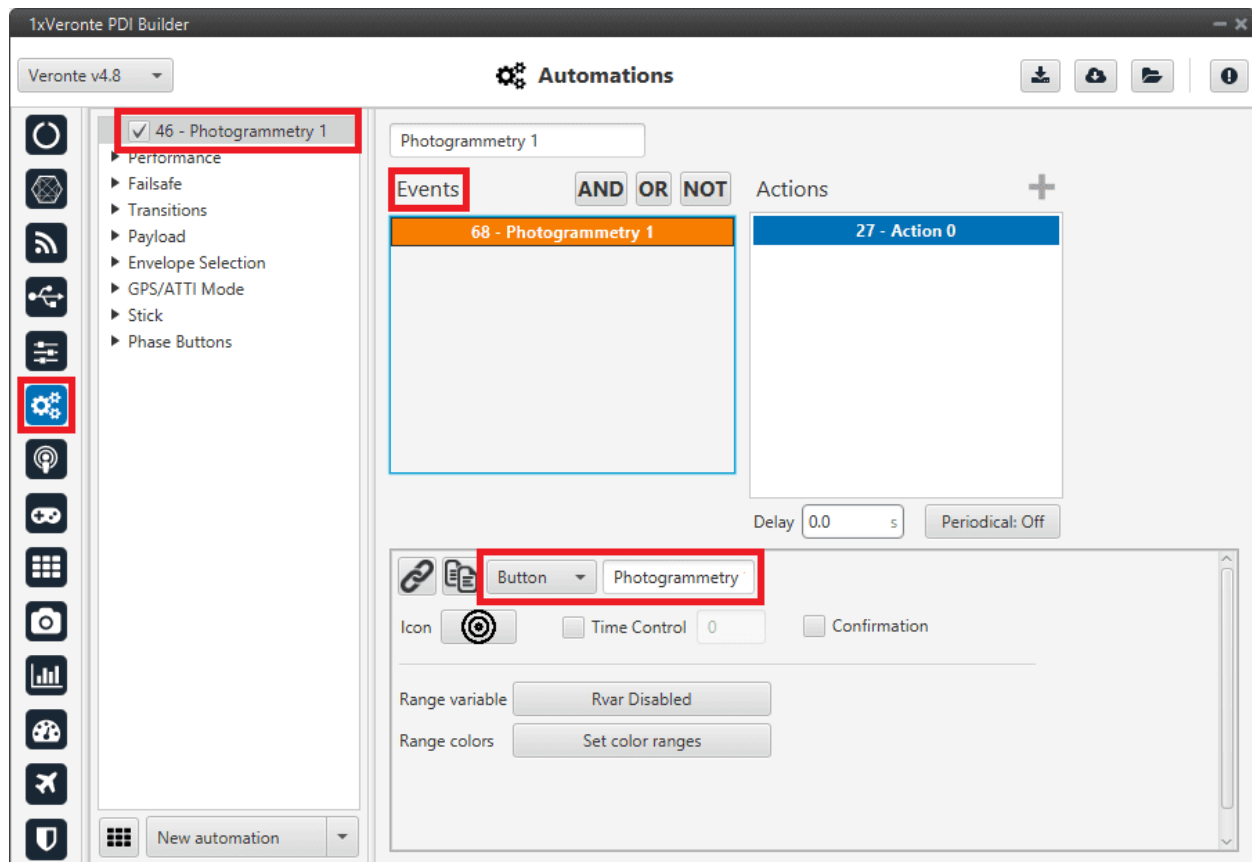


Fig. 440: Cameras - Example

The automation created for Photogrammetry is shown below:



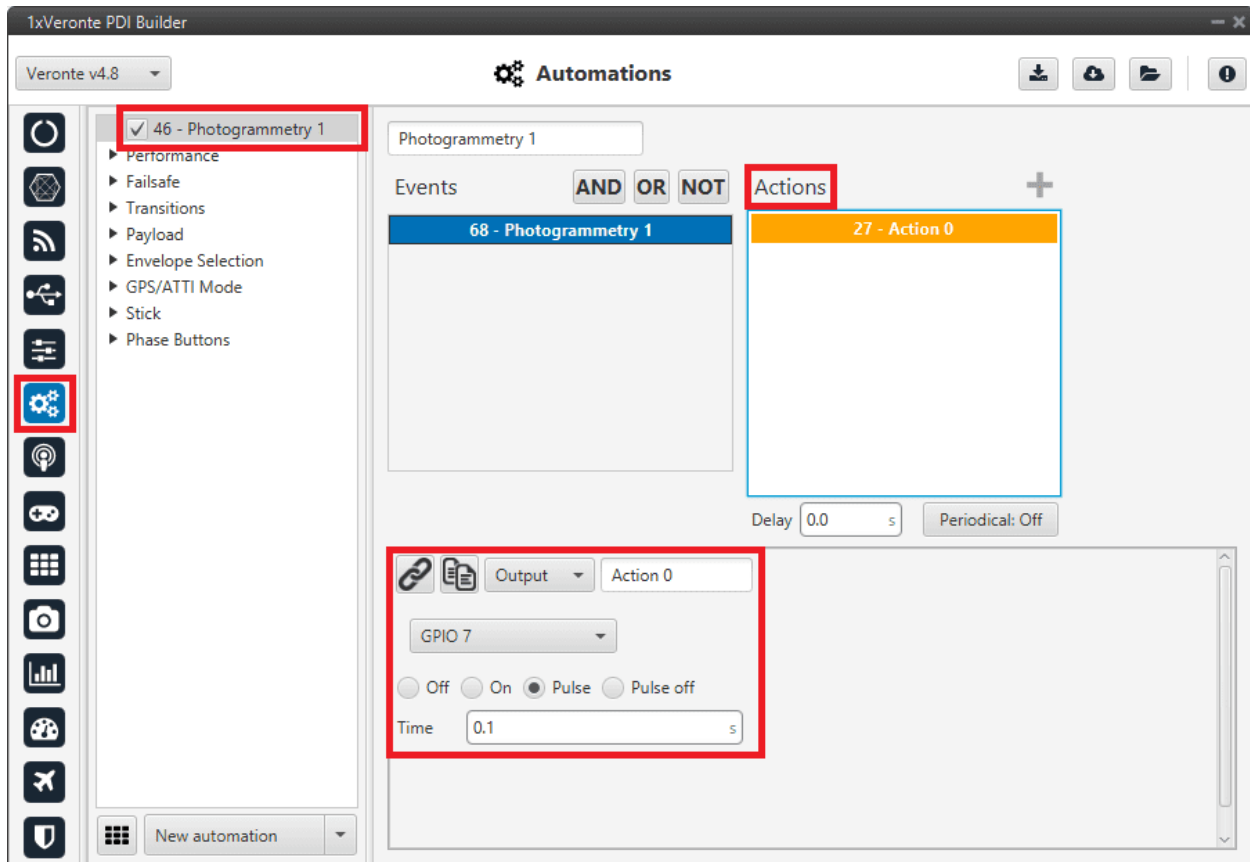


Fig. 441: Cameras - Automation example

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 - Integration examples* or *MC01 - Integration examples*.

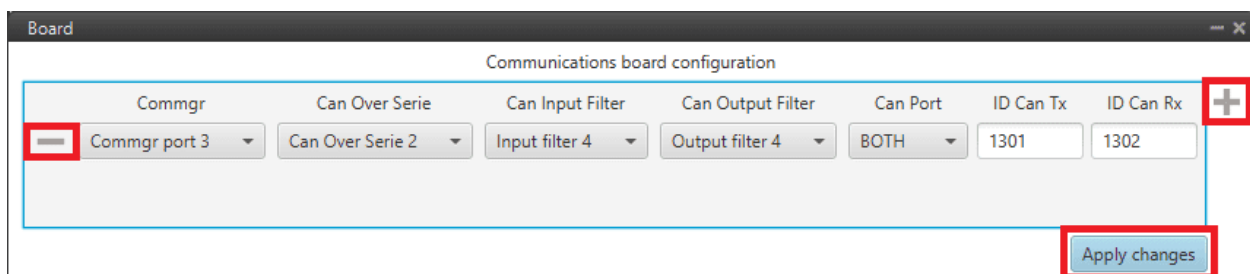



Fig. 442: Board section

- 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 1-6*.
 - **Can Over Serie**: Select the desired **Serial to CAN / CAN to serial**: *Serial to CAN 1-2/CAN to Serial 1-2*.
 - **Can Input Filter**: Select the desired input filter to use: *Input Filter 1-2*.

- **Can Output Filter:** Select the output filter the user wishes to use: *Output Filter 1-2*.
- **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:

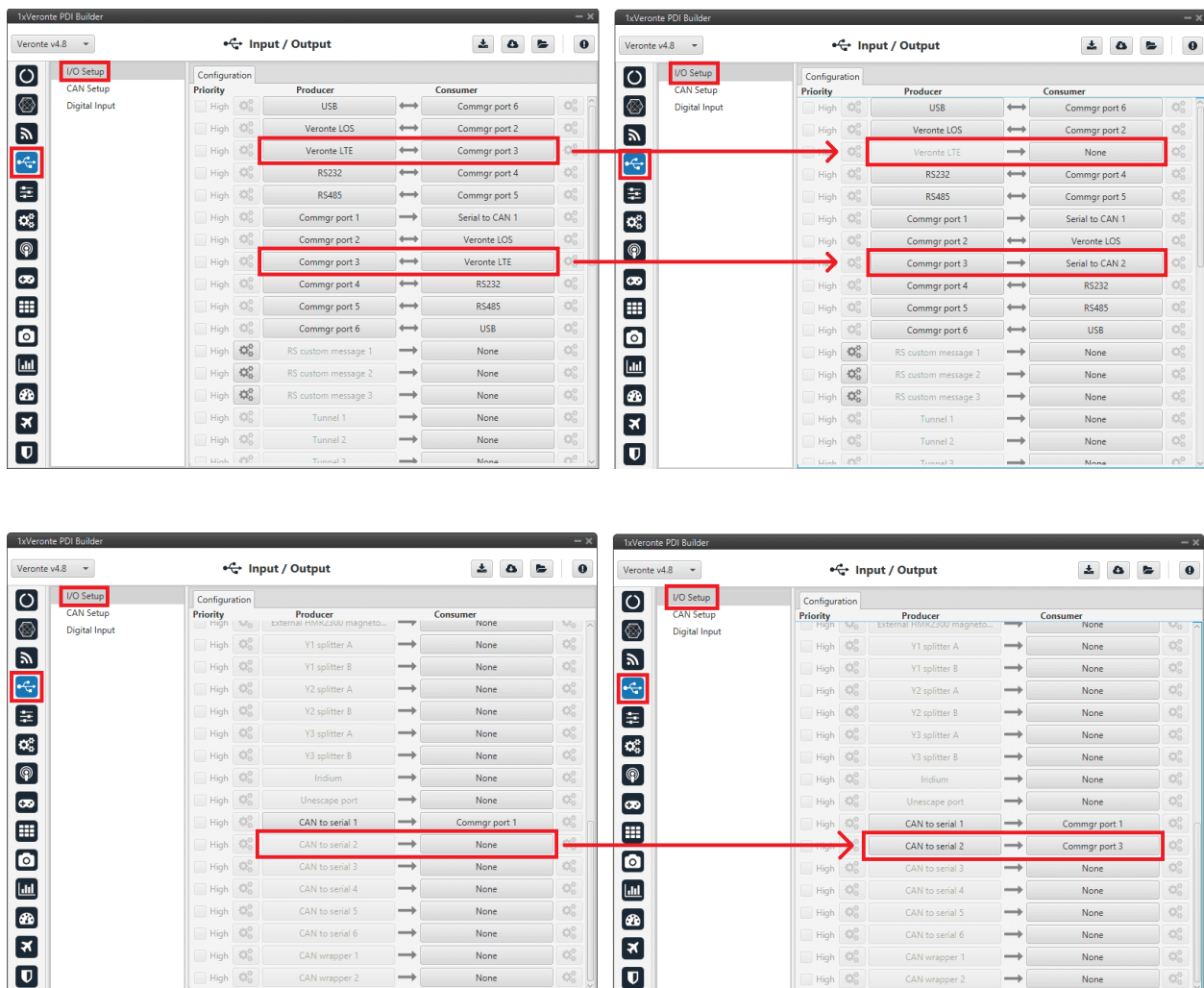


Fig. 443: Board - I/O Setup configuration

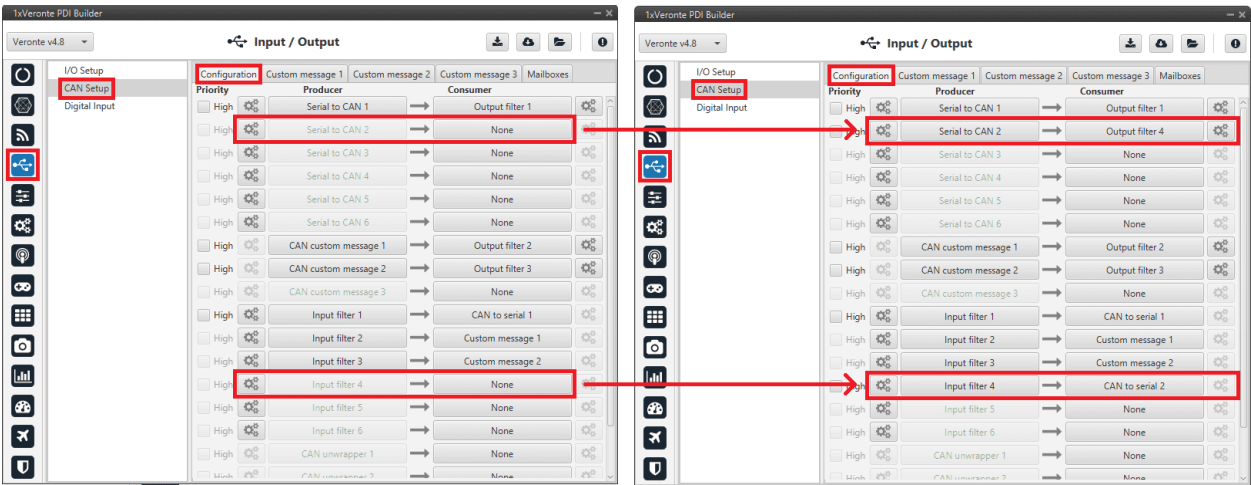


Fig. 444: Board - CAN Setup configuration

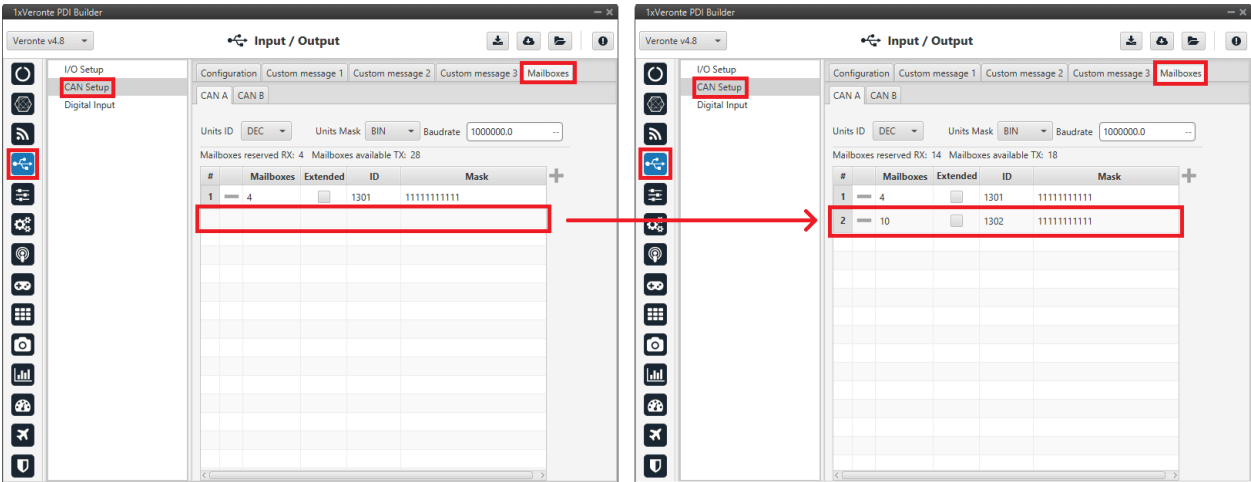


Fig. 445: Board - Mailboxes configuration

2.11 Telemetry

In this section there are 2 options available: **Telemetry** and **Sniffer**.

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 to VApp	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.

Data to VApp

This menu contains the variables sent **between 1x autopilots and Veronte Ops**. By default, the system provides one Data link that represents the connection between the air autopilot and the software (Veronte Ops).

1x autopilot air unit sends the variables to 1x autopilot ground unit, being processed when they arrive there by Veronte Ops. The variables indicated in **red** in a Data to Vapp are **required for correct operation** of Veronte Ops.

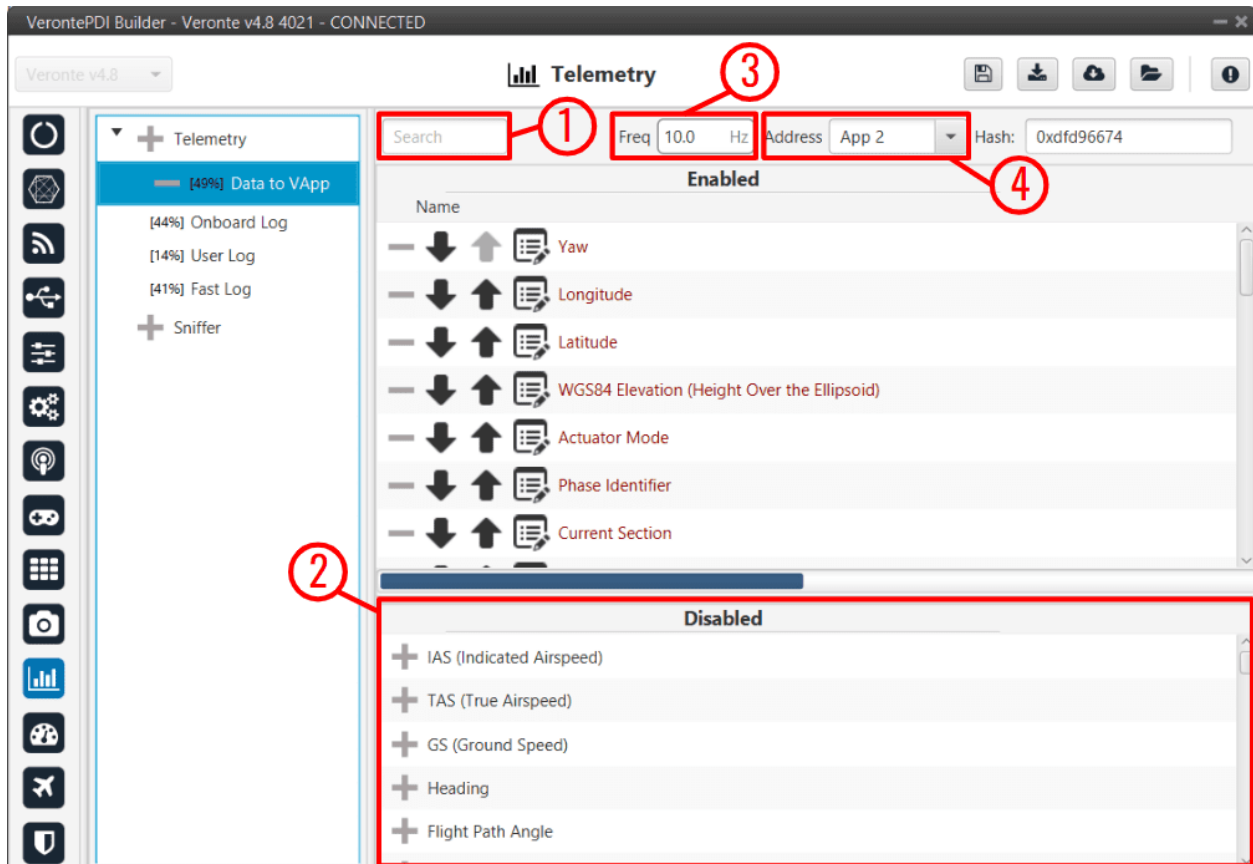


Fig. 446: Data to VApp section

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:** Select the corresponding address, the option available are *App 2* (Veronte Ops address), *Broadcast* (all units on the network) and *Veronte v4.X XXXX* (to a specific unit). Usually **App 2**.

1x PDI Builder allows the creation of more Data links, the user can add it 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 Ops) 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.

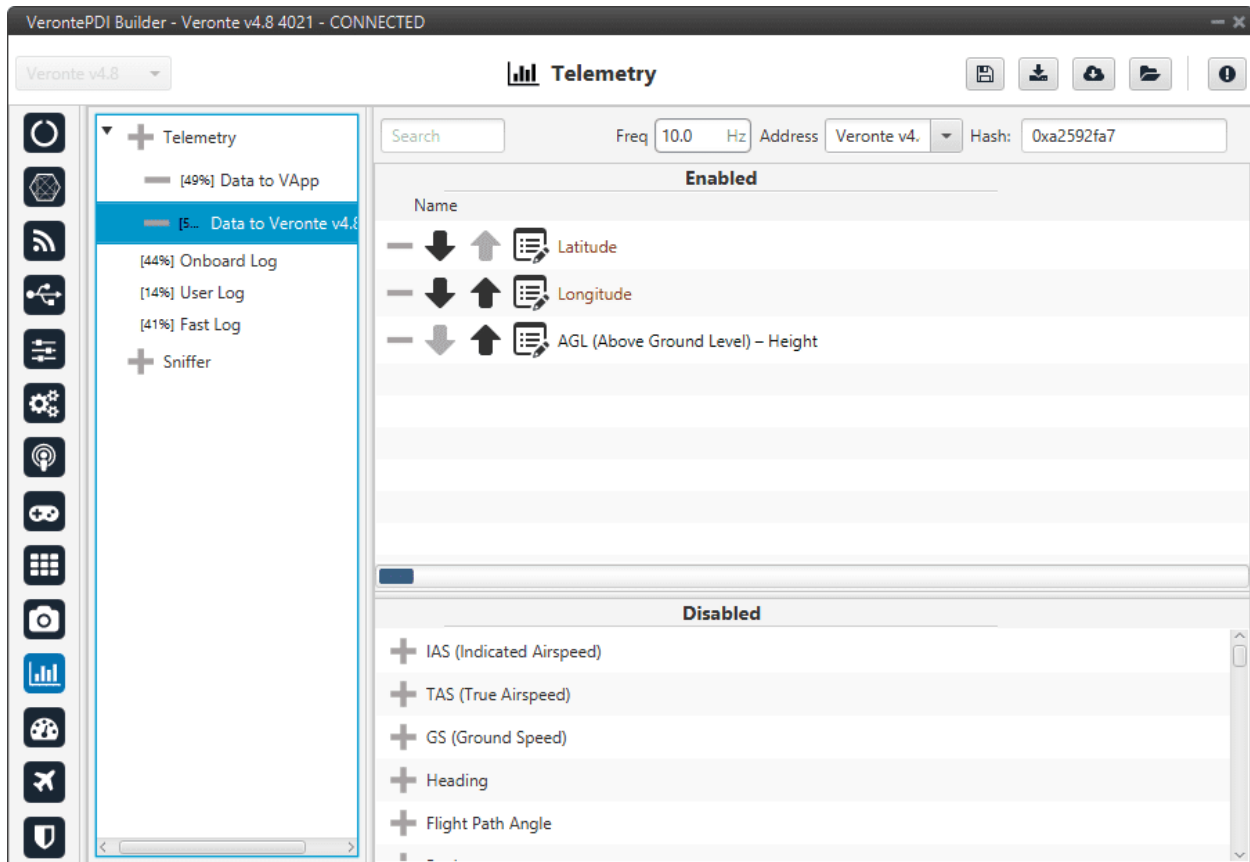


Fig. 447: Data Ground/Air

If we consider this configuration as that of the air unit, this 1x autopilot will send the Latitude, Longitude and AGL to the autopilot with address Veronte v4.8 4021. 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.

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**.

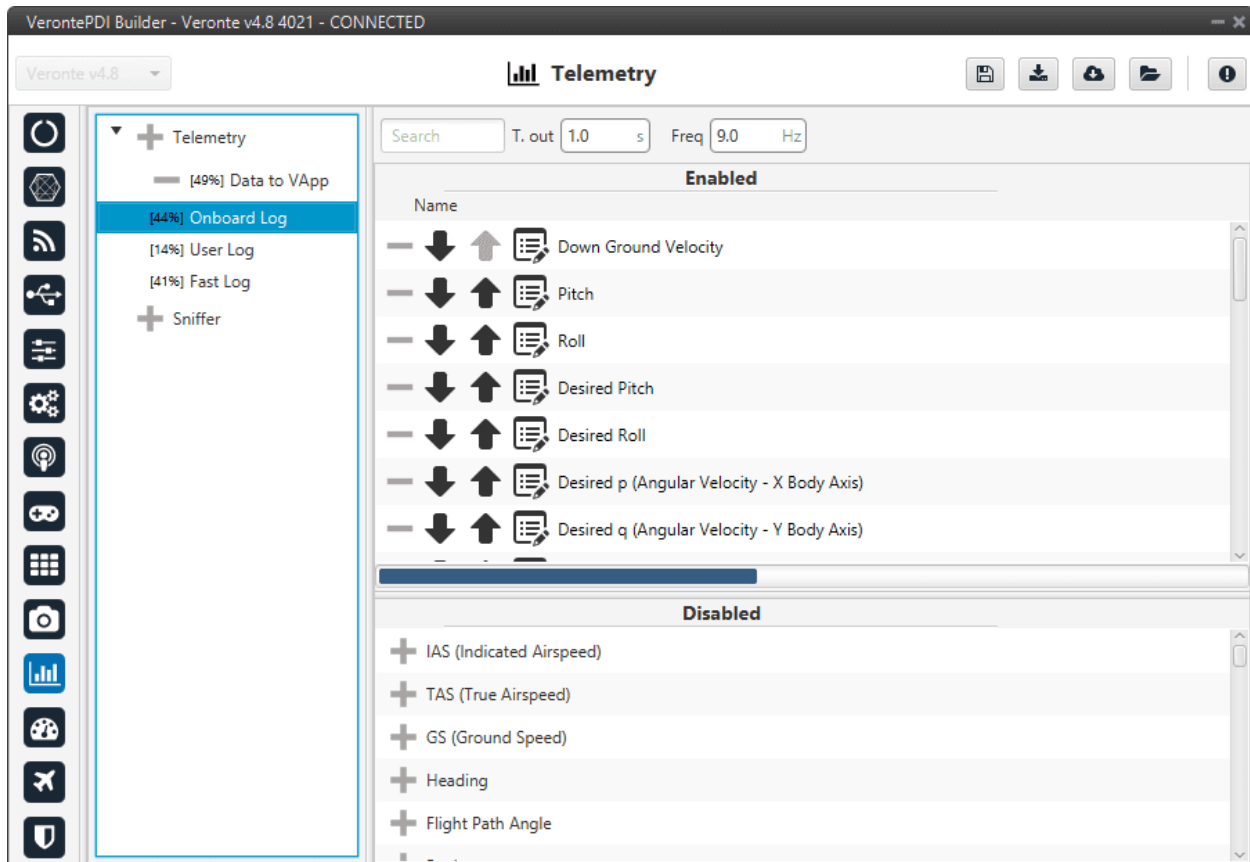


Fig. 448: Onboard Log section

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.

1x Autopilot 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
- $\frac{2.5 \text{ GB}}{400 \text{ bytes/s}} = 6710886 \text{ s} = 1864 \text{ h} = 77.7 \text{ days}$

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.

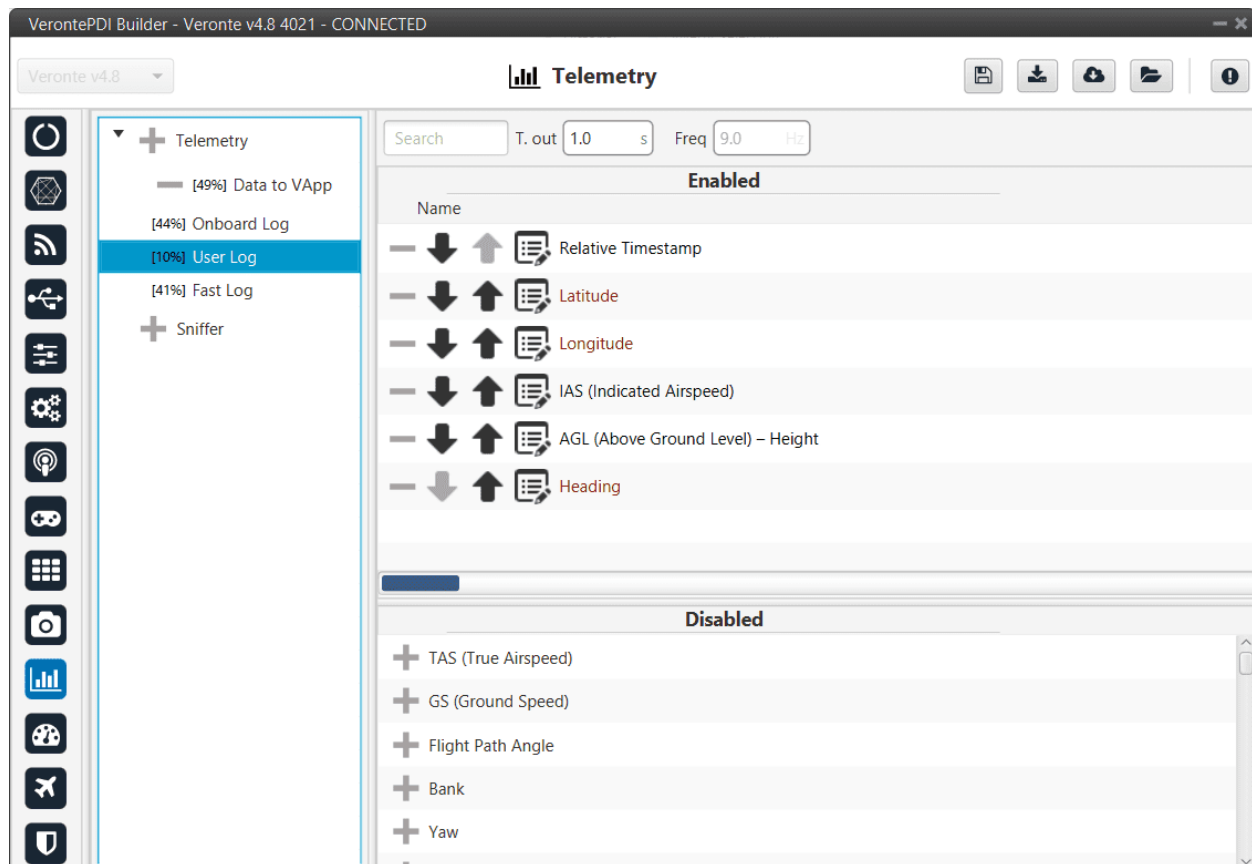


Fig. 449: User Log section

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 [Actions](#) section of the Automations menu.

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.

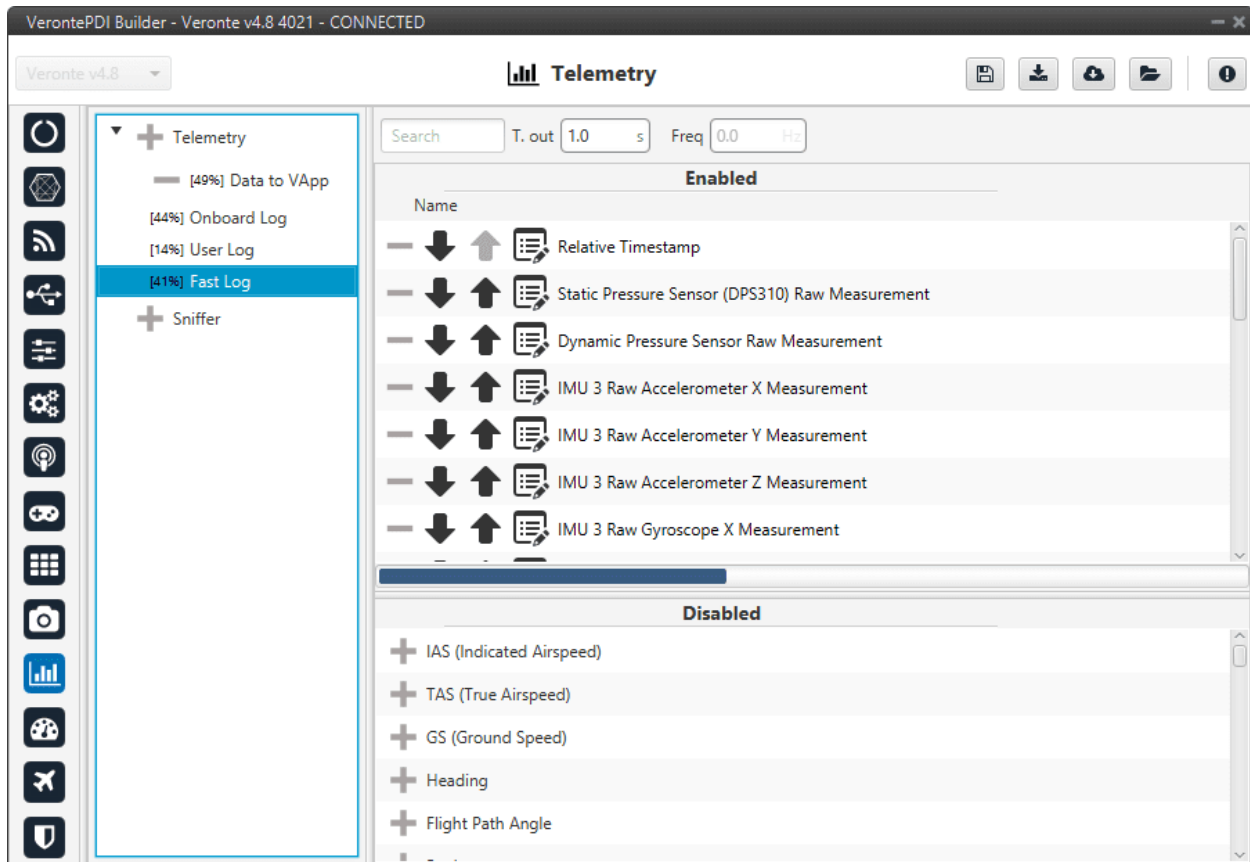


Fig. 450: Fast Log section

The Fast Log can be activate when the user want, but must be done in Veronte Ops. For more information about Fast Log, click on the [Veronte Ops manual](#).

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). Visit [FDR manual](#) for information related to **Onboard Log**, **User Log** and **Fast log** downloading. And [Veronte Link manual](#) for information about **Data to VApp**.

Besides, 1x autopilot includes some compression tools that may be useful for increasing the amount of information transmitted in a certain bandwidth or stored in a log. Each variable can be compressed separately in each log.

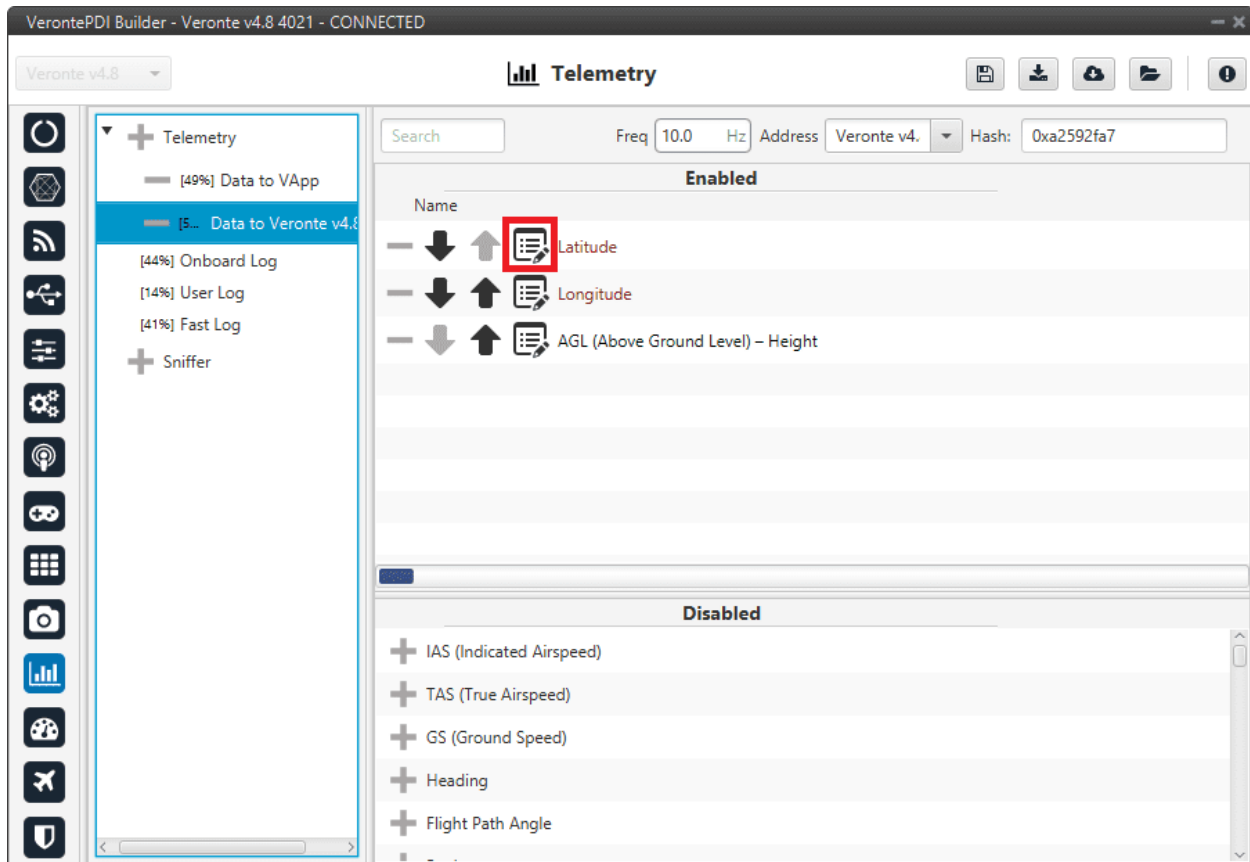


Fig. 451: Compression options

There are different types of compression available:

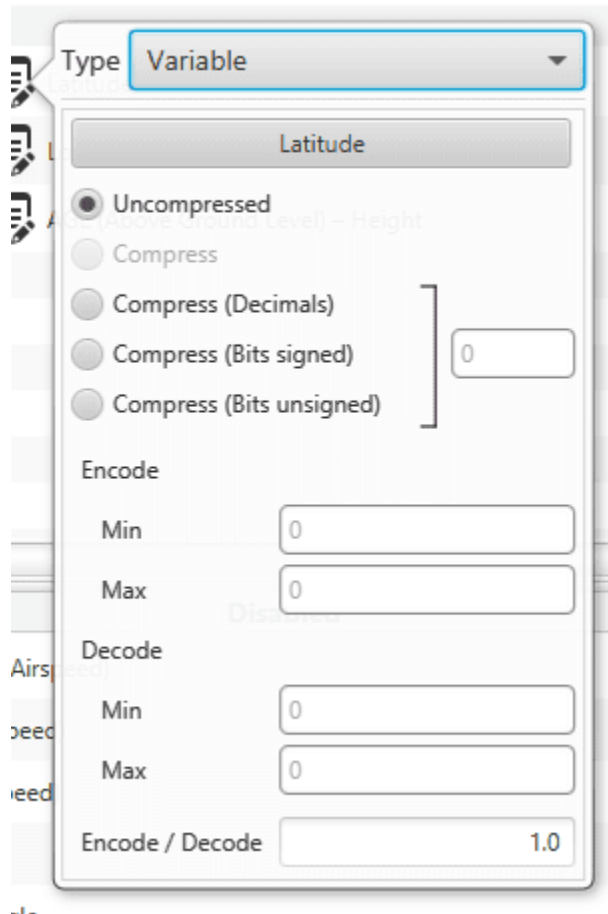


Fig. 452: Compression options panel

- **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.

The screenshot shows a configuration window for a variable named 'Height'. The 'Type' is set to 'Variable'. Under the 'Heading' section, there are five radio button options: 'Uncompressed', 'Compress', 'Compress (Decimals)', 'Compress (Bits signed)', and 'Compress (Bits unsigned)'. The 'Compress (Decimals)' option is selected. To the right of these options is a text box containing the number '3'. Below this, there are two sections: 'Encode' and 'Decode'. The 'Encode' section has 'Min' set to '0' and 'Max' set to '359.999'. The 'Decode' section has 'Min' set to '0' and 'Max' set to '0'. At the bottom, there is a field for 'Encode / Decode' with a value of '1.0'.

Fig. 453: Compression example

2.11.2 Sniffer

This menu is used to establish a telemetry communication between two autopilots. 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.

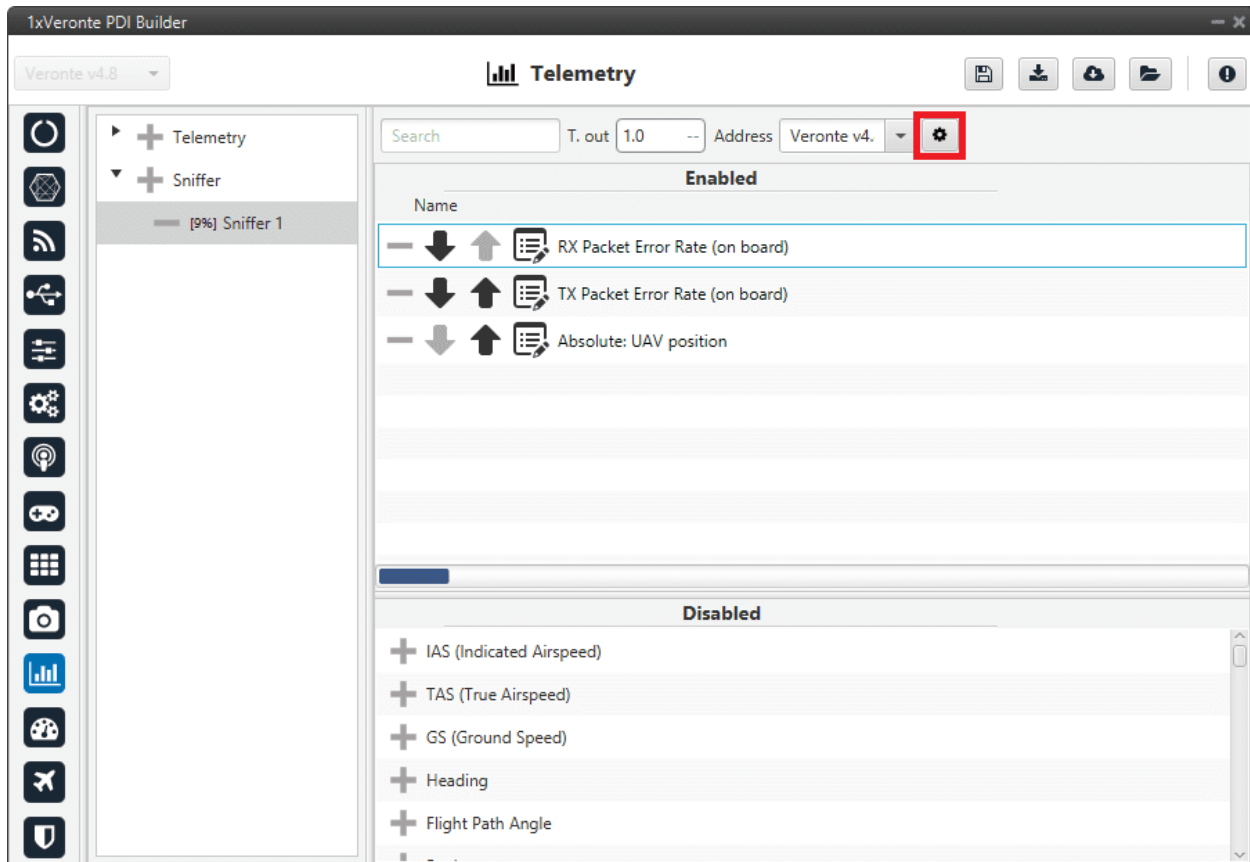



Fig. 454: Sniffer menu

The source UAV, in this case, is the ground station (Veronte v4.8 4021), which communicates to the 1x air unit 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 1x autopilot unit that sends the data has to be configured as well (1x ground unit), in the *Telemetry* section. 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 column **In**, and they are stored in the variables indicated in **Out** for its later use by the air 1x autopilot.

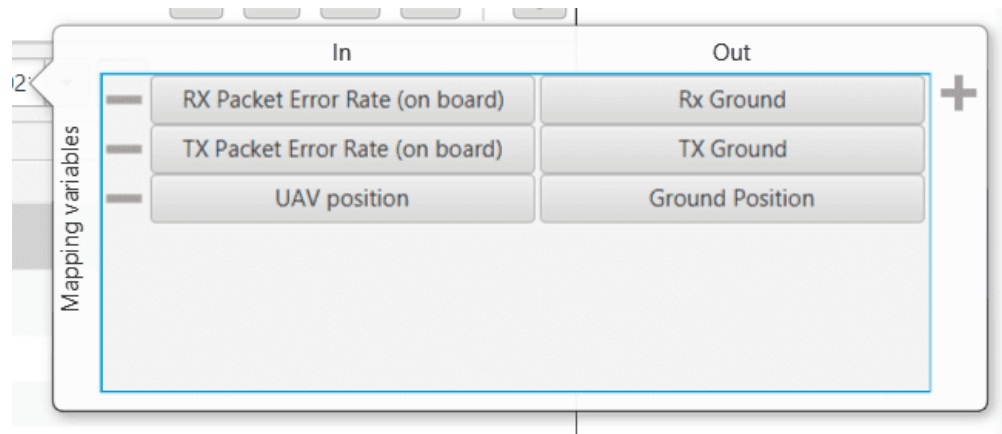


Fig. 455: Mapping variables option

An example of the configuration required for communication between 1x ground and air units can be found in [Data transmission between Veronte 1x Autopilots](#) -> *Integration examples* section of this manual.

2.12 UI

In this menu, the user can manage operation elements, system variables and the geoid world mesh.

2.12.1 Operation elements

In this section, the user can rename operation elements. These are variables that have no value until an operation is performed, e.g. cruise speed and altitudes, marks to initiate landing, the route, etc.

In addition, they must first be renamed in this section, so that they can then be referenced in the setup, as in Automations (see [Automations](#) section), and then defined in the operation, in **Veronte Ops** (visit [Veronte Ops manual](#)).

Operation elements are divided into 8 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.
- **Marks:** A mark is a reference that is placed in a patch and when the uav reaches it, an action takes place.
- **Polygons:** A polygon is a detection area.
- **Circles:** A circle is a circular polygon.
- **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.
- **Operation variables:** An operation guidance point is a value of the operation, such as the cruise speed.

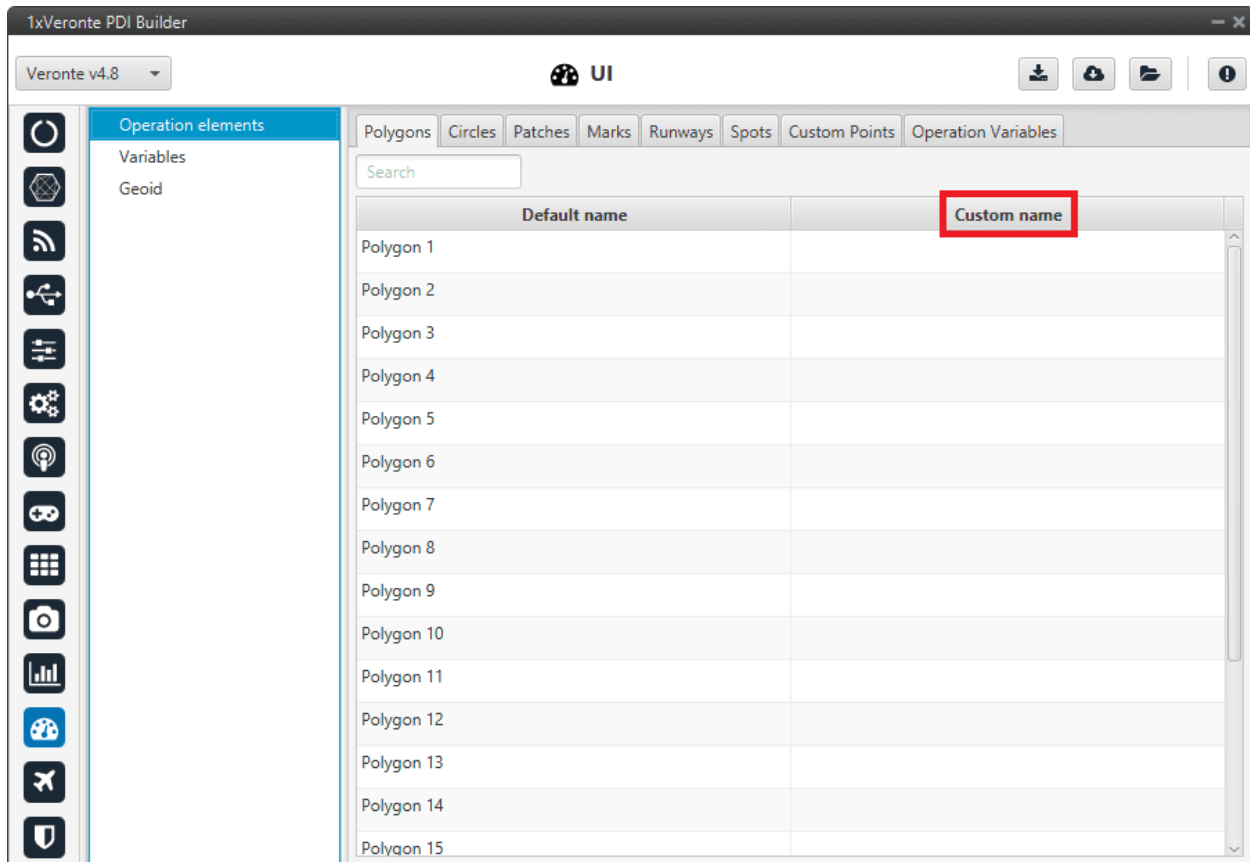


Fig. 456: Operation elements section

2.12.2 Variables

In this section, 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).

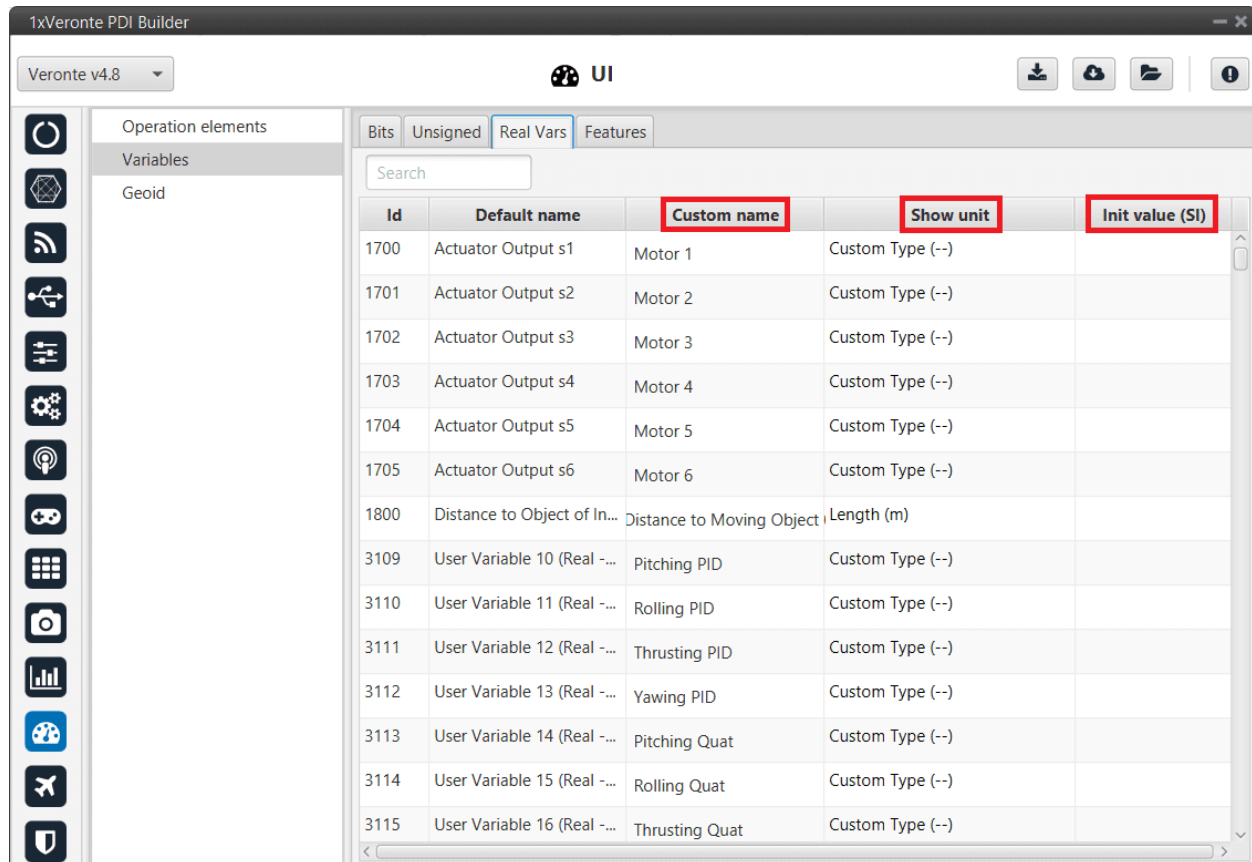


Fig. 457: Variables section

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 Units and the Initial Value (SI) cells.

By right-clicking on the **Show Units** cell, the user can select the desired units.

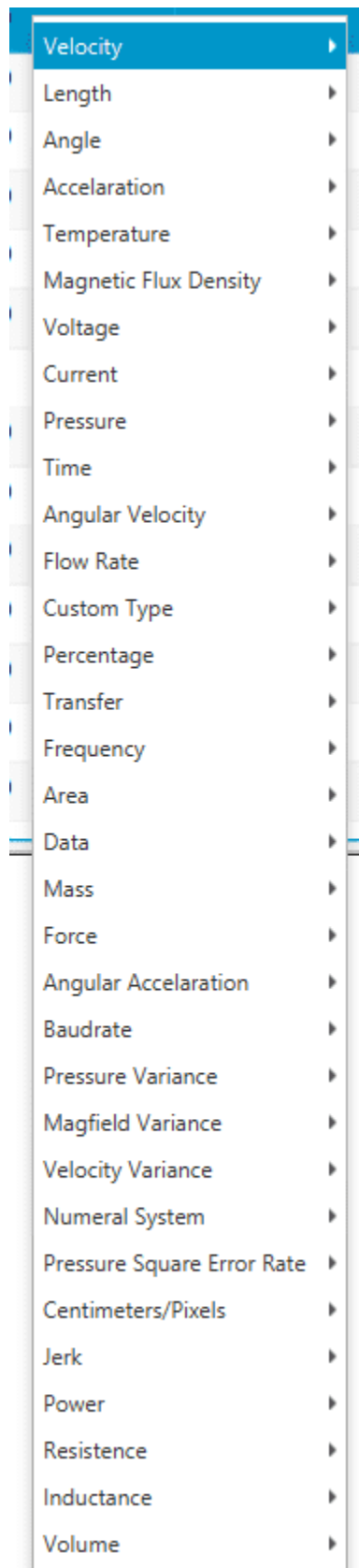


Fig. 458: Variables units

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[-;] °[-180;180] °[0;360] [° ‘ “] [rad] rad[0;2] °
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] [s] [ms] [Time]
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]
Resistance	[]
Inductance	[H]
Volume	[m ³] [dm ³] [mm ³] [L] [mL]

2.12.3 Geoid

In this section, the user can define the world mesh which provides the geoid altitude.

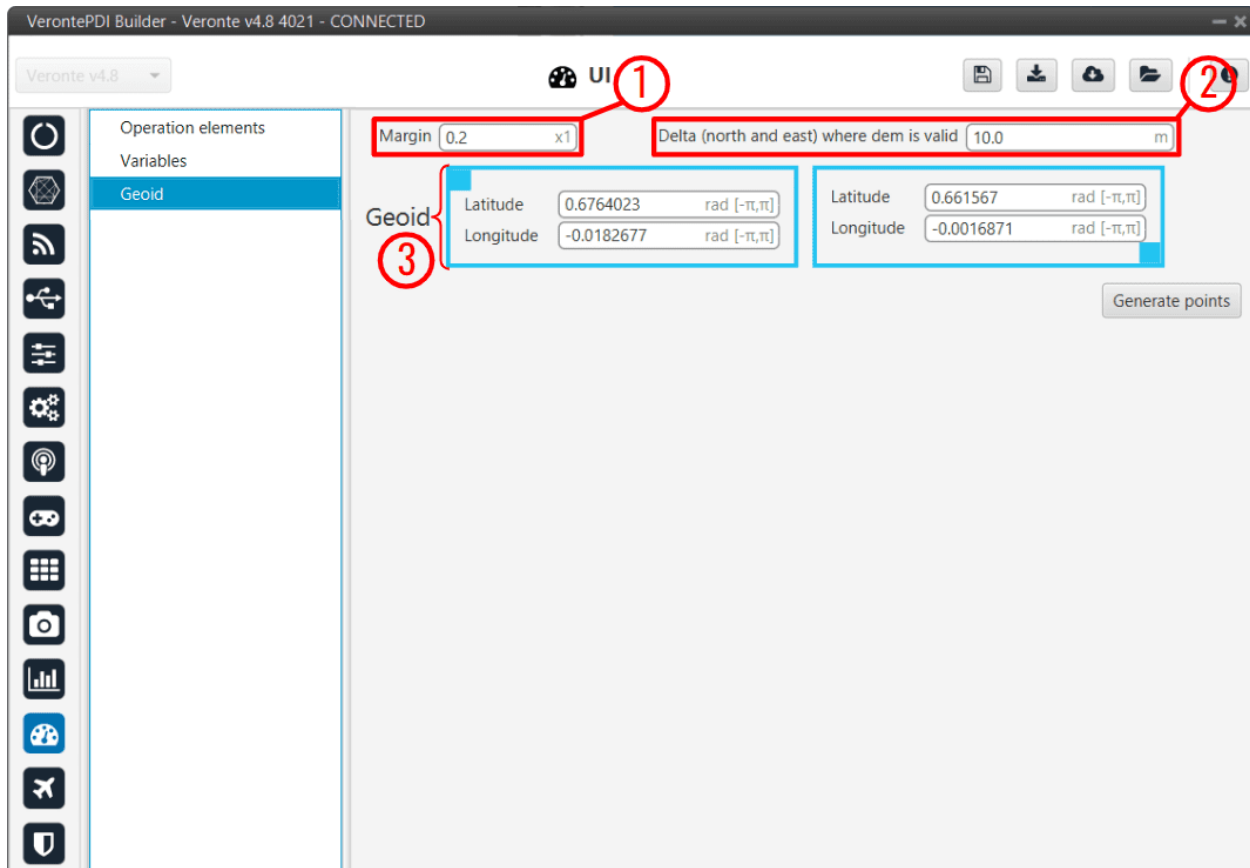


Fig. 459: Geoid section

1. **Margin:** This is the percentage at which the system will recalculate the mission if the route is displaced. In other words, if the mission is displaced 60% out of the area and the margin is set to 80%, the mission will be not recalculated. If the mission is 81% (or more) away from the previous one, the system will recalculate the mission. A low level (or zero) margin means more terrain profile precision but the system will have to recalculate the meshes more times (or each time) when the mission is modified.
2. **Delta:** The distance to the ground (AGL) may be measured through two or more systems. Usually, an altimeter like a LIDAR in conjunction with the GPS signal and the meshes information are used. During flight, it is possible that the GPS position error is large enough that the height provided by the meshes does not correspond with the actual position. In order to avoid problems, the Delta parameter can be defined. This parameter **defines a circumference radius where both systems will be used**. If the **estimated position error is bigger than the delta parameter**, only LIDAR data will be used.

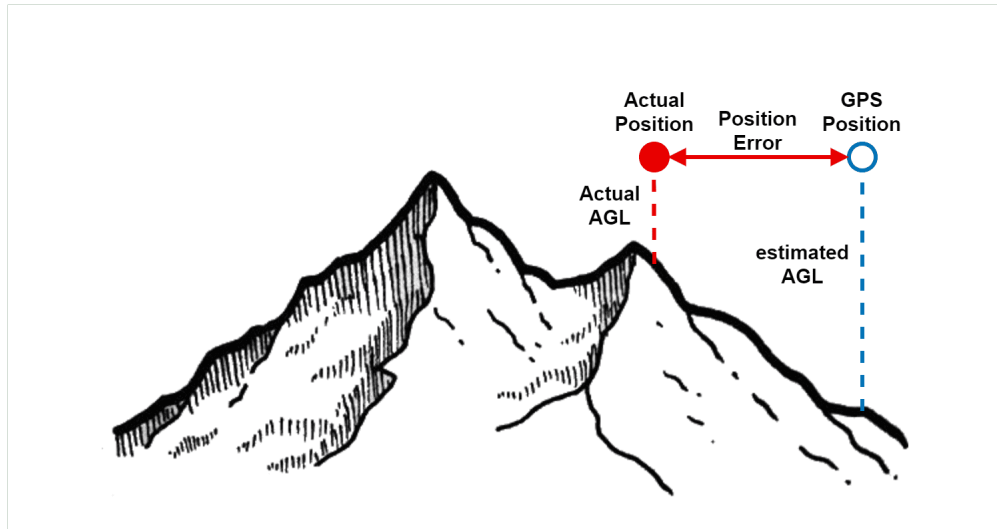


Fig. 460: Delta Parameter

3. **Geoid:** World mesh which provides the geoid altitude. The user must place and adjust the mesh manually, the coordinates of each upper left corner and lower right corner of the meshes' rectangle must be introduced. In general, increasing meshes size will mean lower area definition. And greater resolutions smaller meshes, because it implies heavier data files.

Warning: Check that meshes position are over the mission area before flying, especially if carrying out an operation in mountainous terrain.

2.13 HIL

Professional Hardware In the Loop (HIL) Simulator package is a powerful tool for 1x autopilot integration, development and operator training; allowing to extensively operate the system in a safe environment, prior to conducting real flight operations.

The user can link the variables on 1x autopilot with the corresponding ones in the simulator. In this panel, simulator variables are available on the left side (**Disables**). In addition, it can be seen 2 section more, **To Simulator** and **To Veronte**.

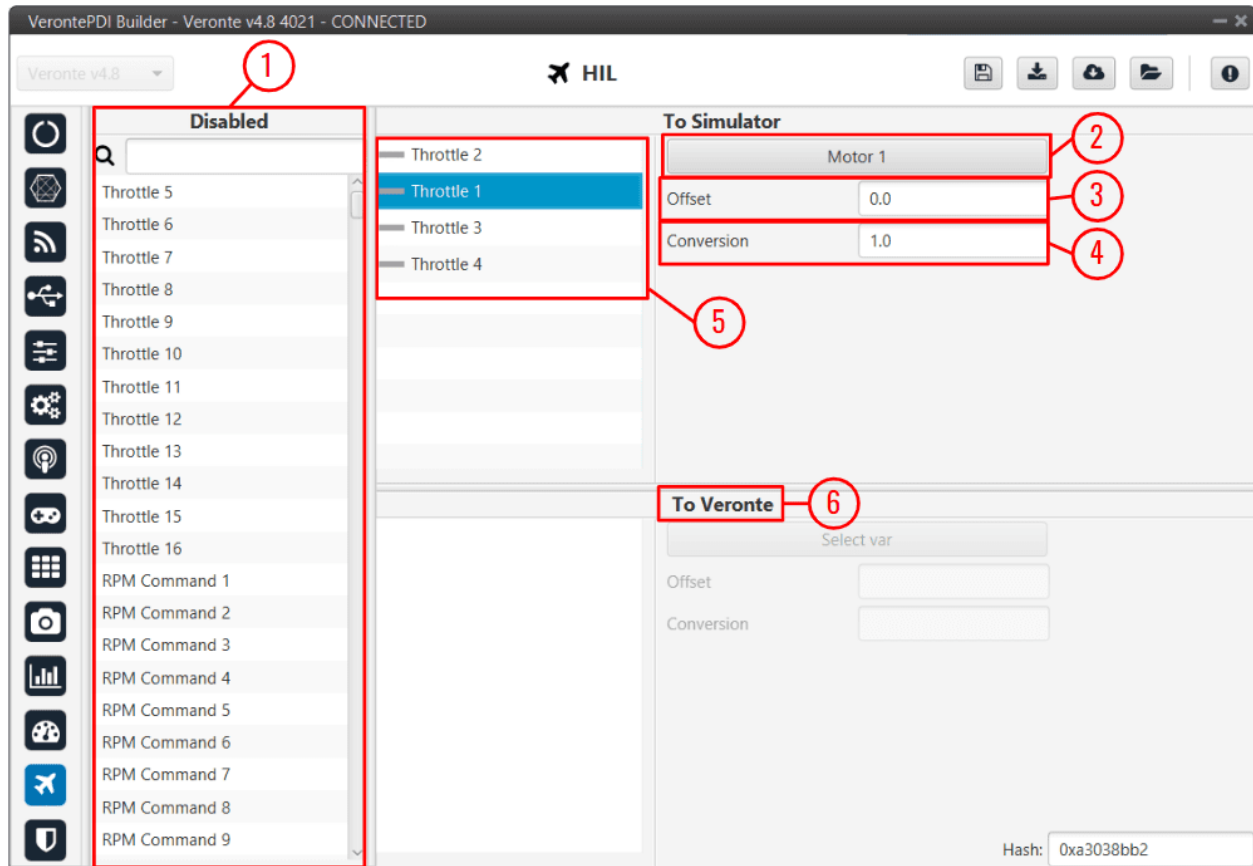


Fig. 461: HIL menu

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** section.
2. Select the actuator variable (Control Output) of 1x autopilot that matches with the one in the simulator. A new window will be displayed for each variable.

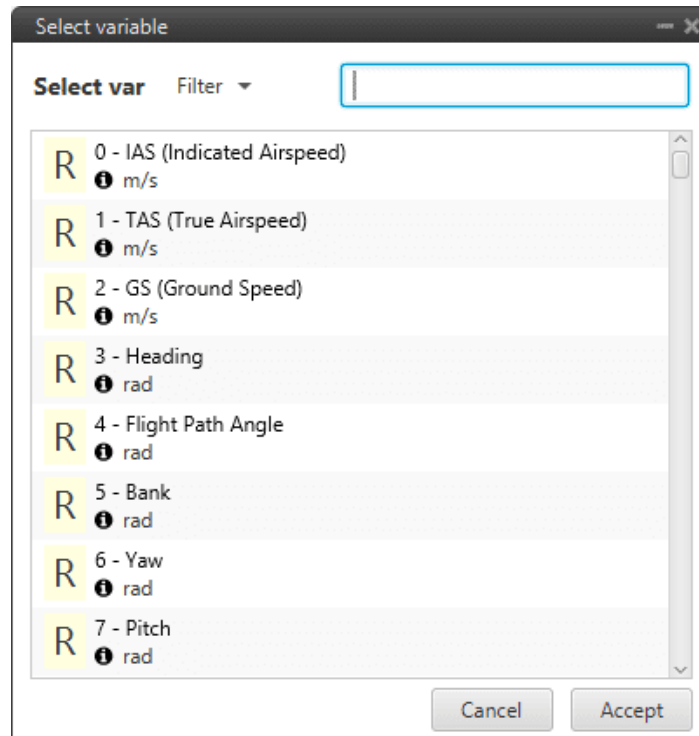


Fig. 462: 1x autopilot variables

4. **Offset:** Set an offset, if it is necessary.
3. **Conversion Factor:** Set a conversion factor, if it is necessary. It multiplies the 1x autopilot 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.

Warning: Always make sure that surfaces are moving in the right direction and with the correct deflection angle.

5. Here the user can see all the variables he has selected and sent to the simulator.
6. **To Veronte:** The user can also select variables to be sent from the simulator to 1x autopilot. An interesting variable could be the RPM of the motor.

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 1x Autopilot, such as “**In Range check**”.
- Checks that need a command to 1x Autopilot, e.g. “**Calibrate Atmosphere**”.
- Checks for operator information only, which are performed with type “**None**”.

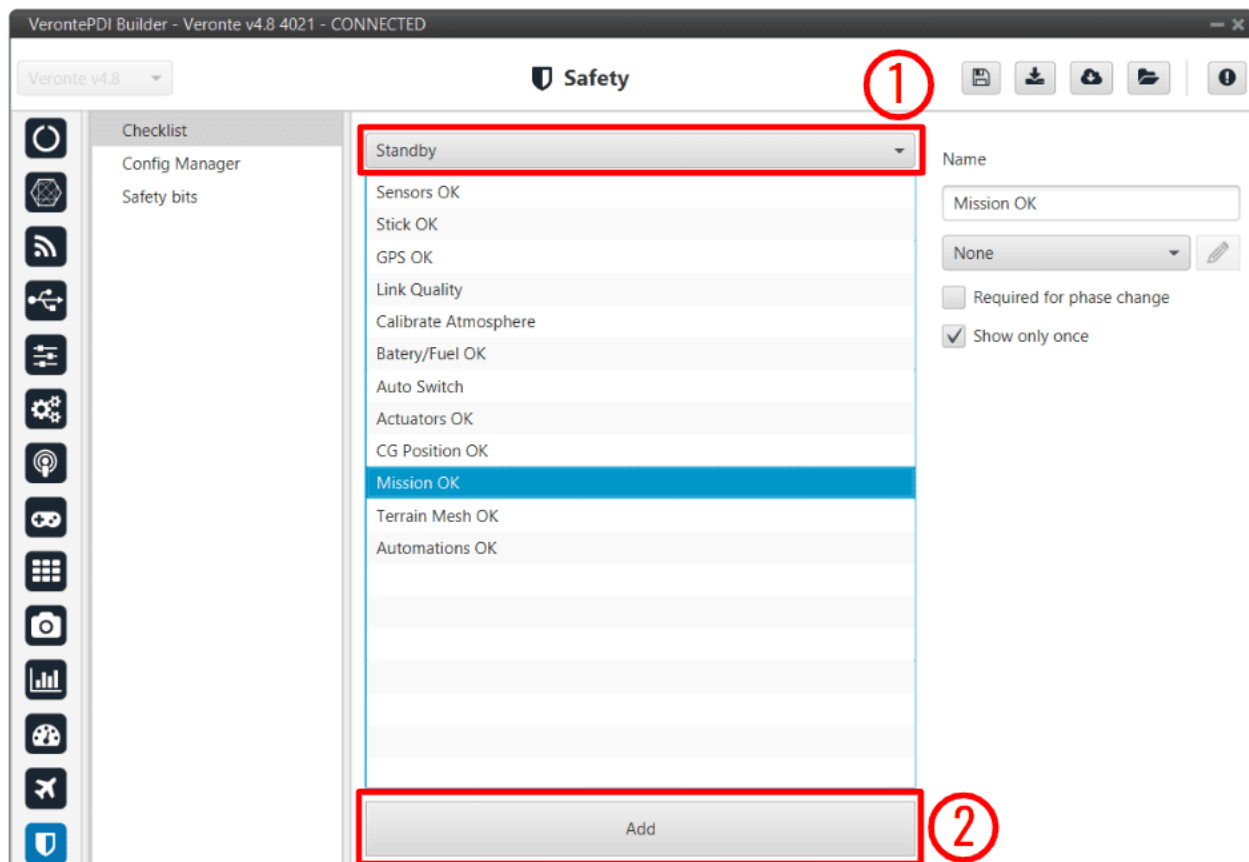


Fig. 463: Checklist section

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.
 - **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:

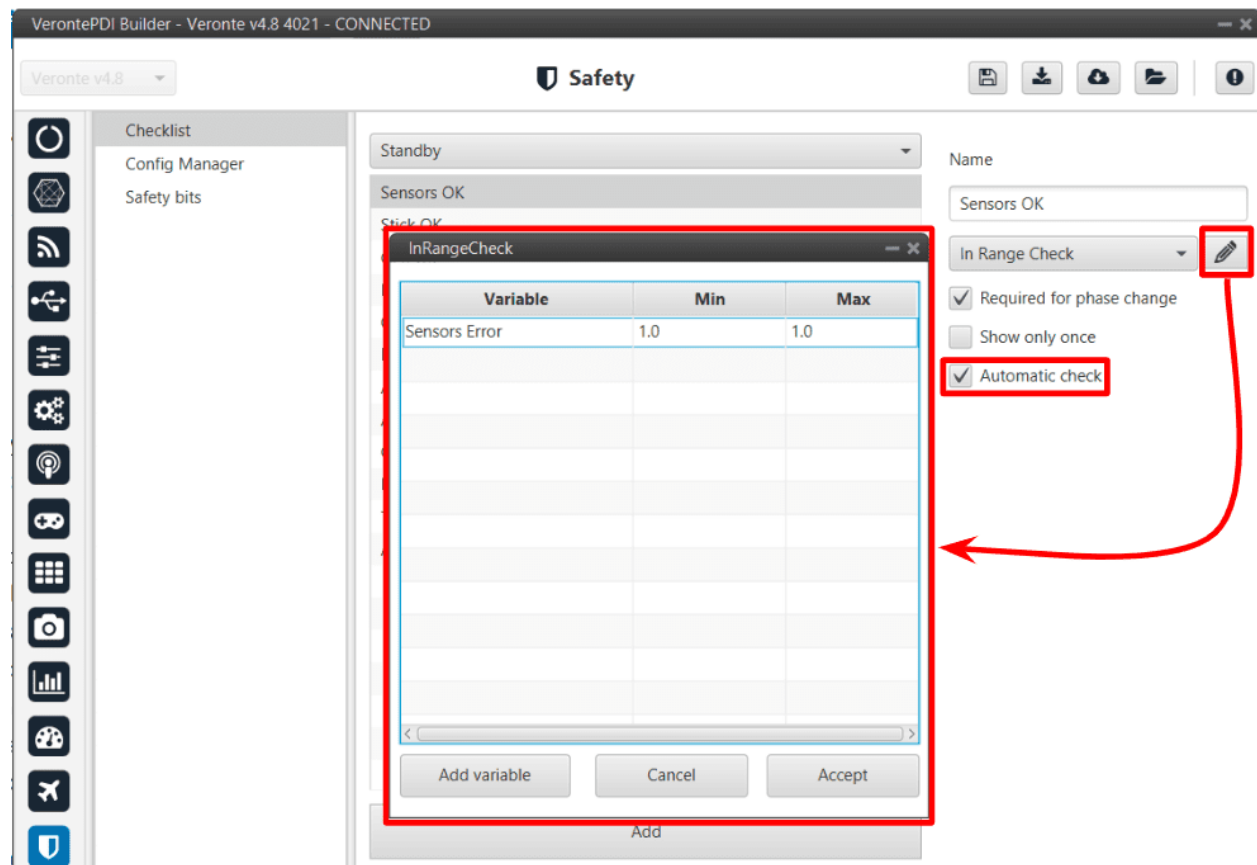


Fig. 464: Checklist example

2.14.2 Config Manager

Config manager avoid the user changing certain parameters, settings or programs of 1x autopilot. It is shown in the picture below:

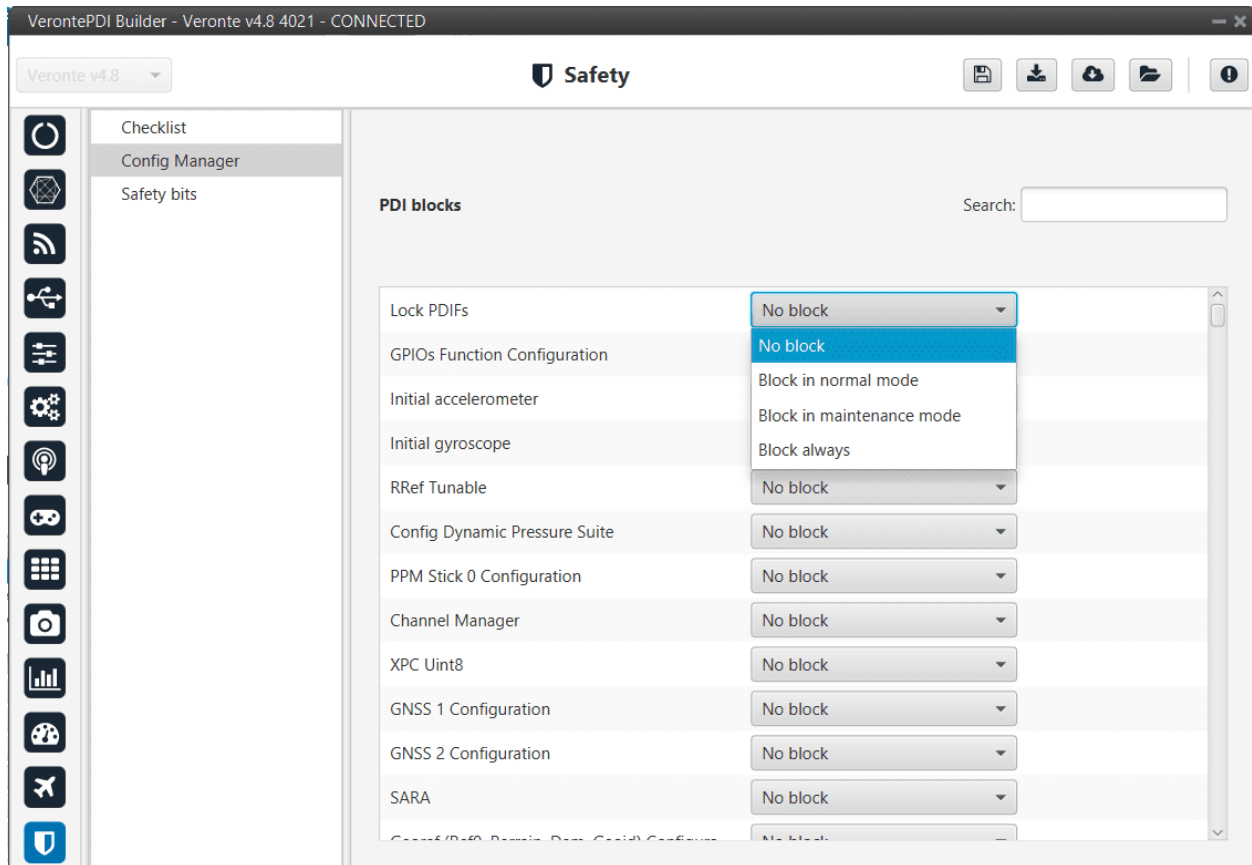


Fig. 465: Config manager section

The user can choose between:

- No block
- Block in normal mode
- Block in maintenance mode
- Block always

2.14.3 Safety bits

In this section 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**.

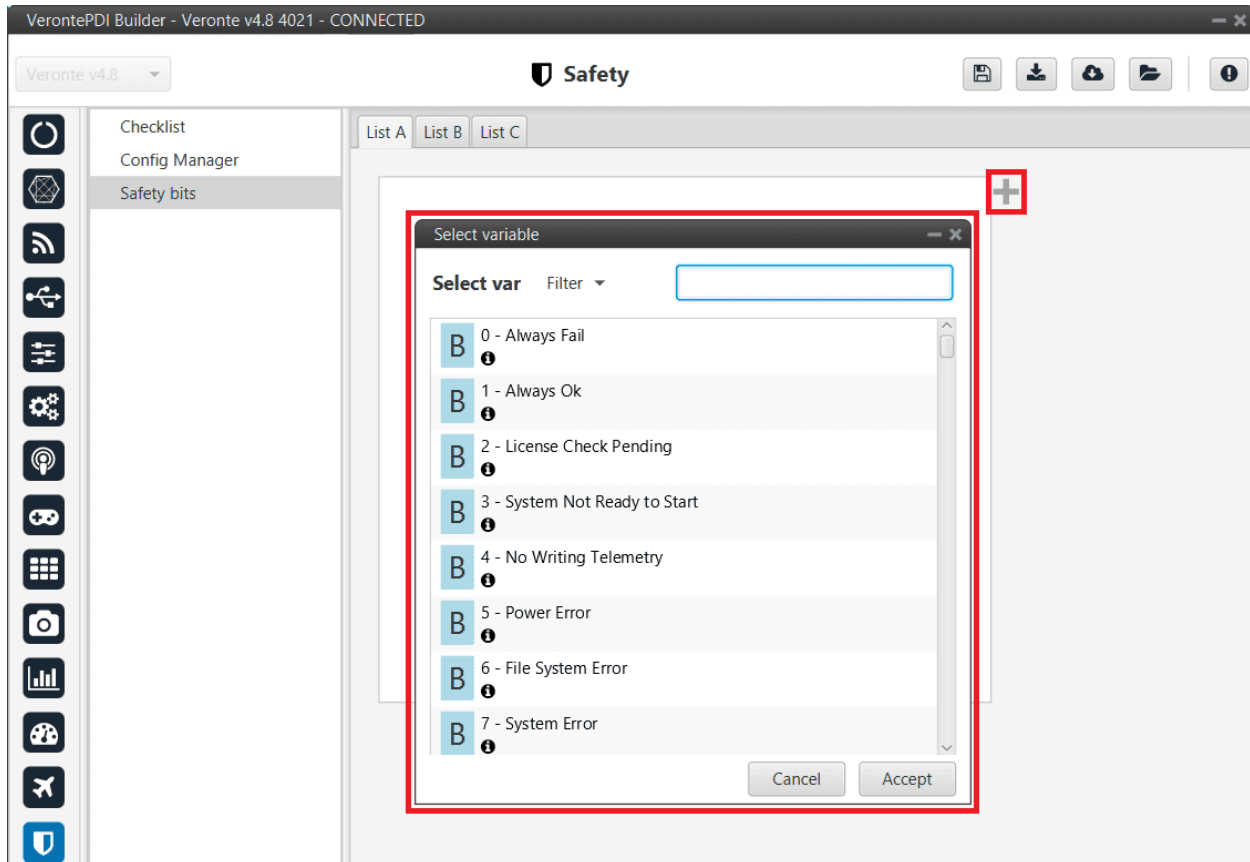


Fig. 466: Safety bits section

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, see [Actions](#) of the Automations menu.

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

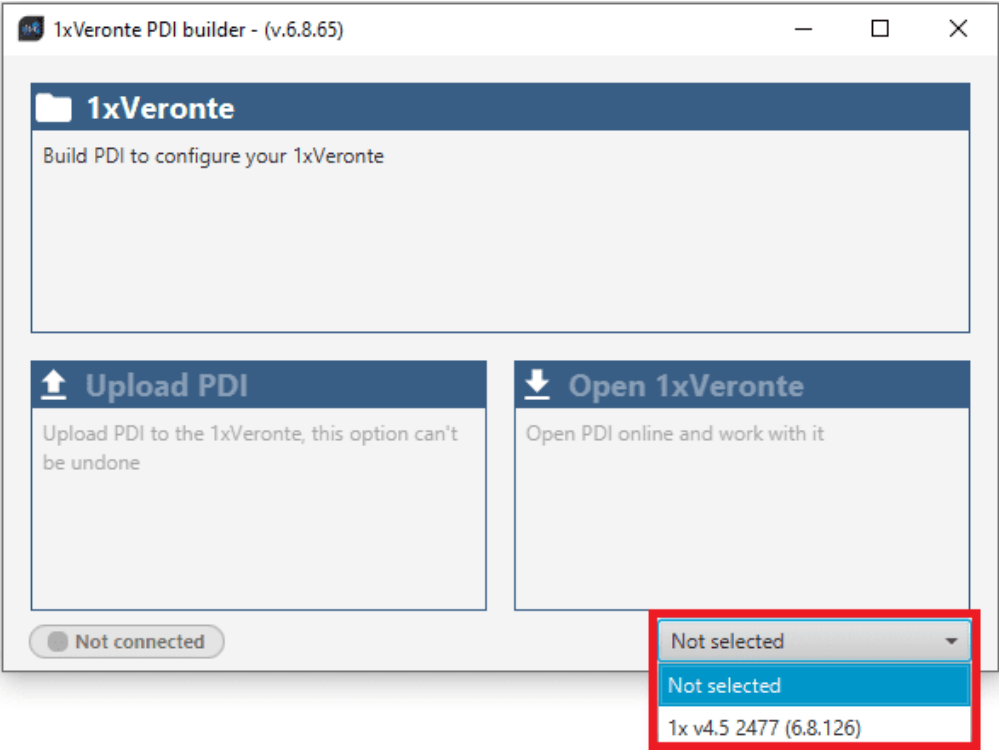


Fig. 467: 1x autopilot ID

If it is correctly connected, it should appear in **Normal mode** as shown in the following figure.

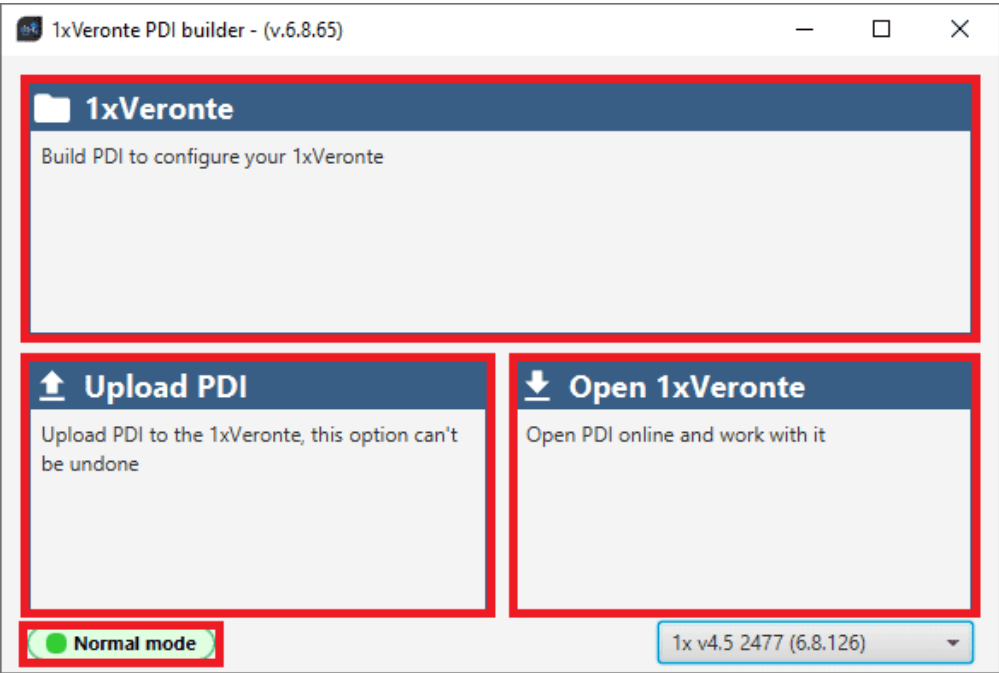


Fig. 468: 1x PDI Builder

1x unit can also appear as: Maintenance mode, 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 [Troubleshooting section](#) of this manual.

The user can access now to 3 configuration options:

- **Veronte:** It allows the user to work with **offline** configurations. A previously exported 1x PDI 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:

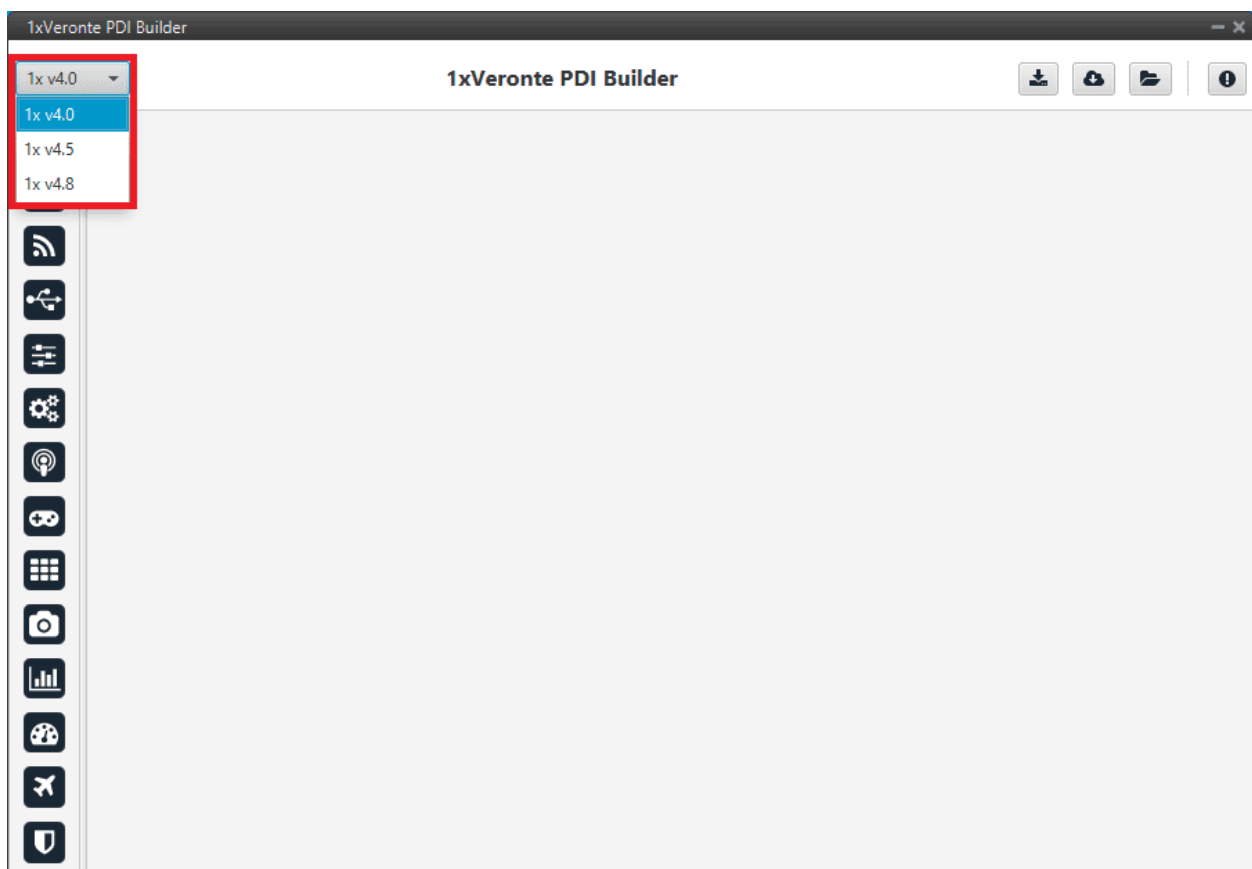


Fig. 469: **Hardware versions**

-
- **Upload PDI:** A previously exported 1x PDI configuration can be imported to the linked 1x.
 - **Open Veronte:** By clicking on this option, 1x PDI Builder configuration menu opens with the configuration (the PDI files) loaded in the 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 3 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, runaways, 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.
- **xsd:** This folder holds .xsd files. An XSD file is a definition file specifying the elements and attributes that can be part of an XML document. This ensures that data is properly interpreted, and errors are caught, resulting in appropriate XML validation. **Users should never delete, replace or modify it.**

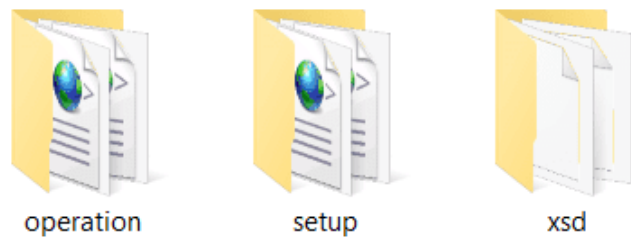


Fig. 470: PDIs files

Finally, click on 'Open Veronte' to open the configuration and start editing. The different 'buttons' that can be seen in the initial menu of the 1x PDI builder are explained below.

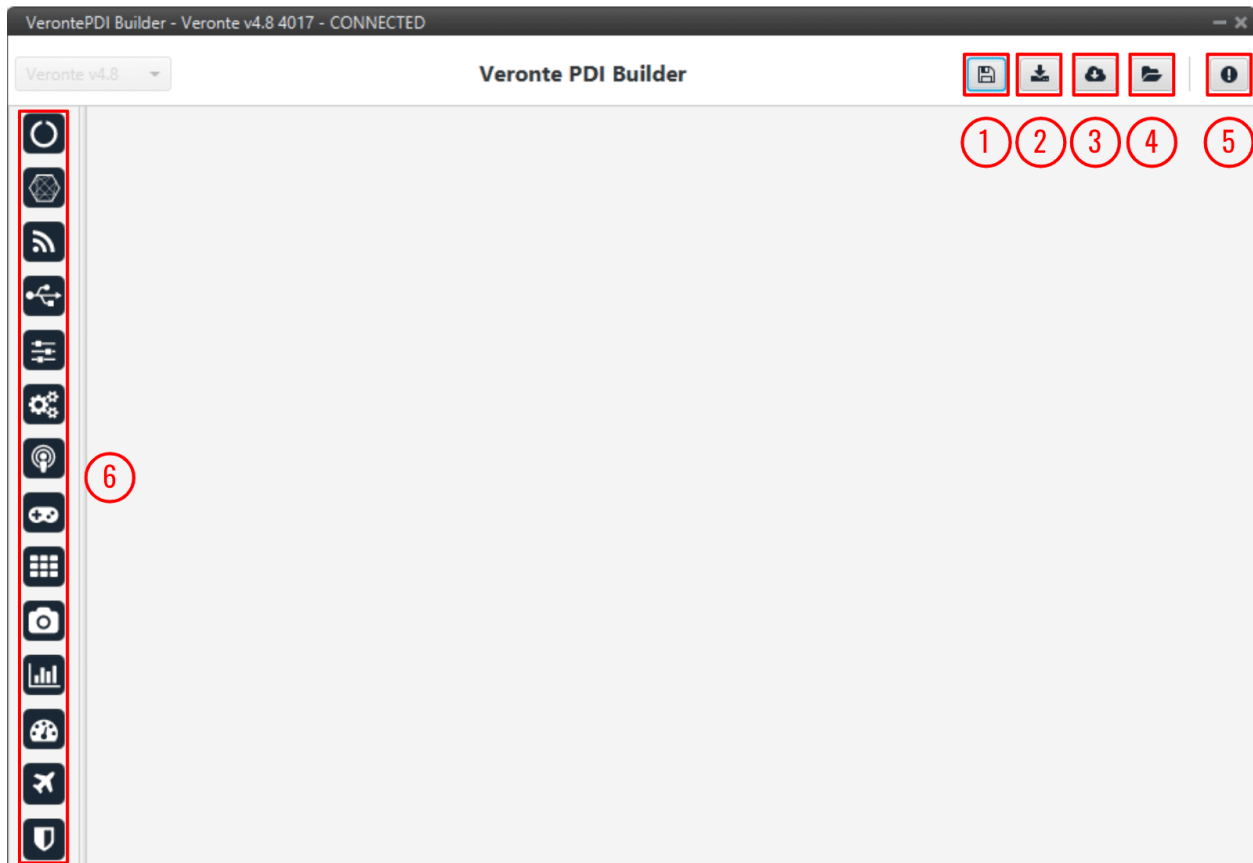


Fig. 471: Initial menu

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

While saving, a percentage of the progress of the saving process is displayed:

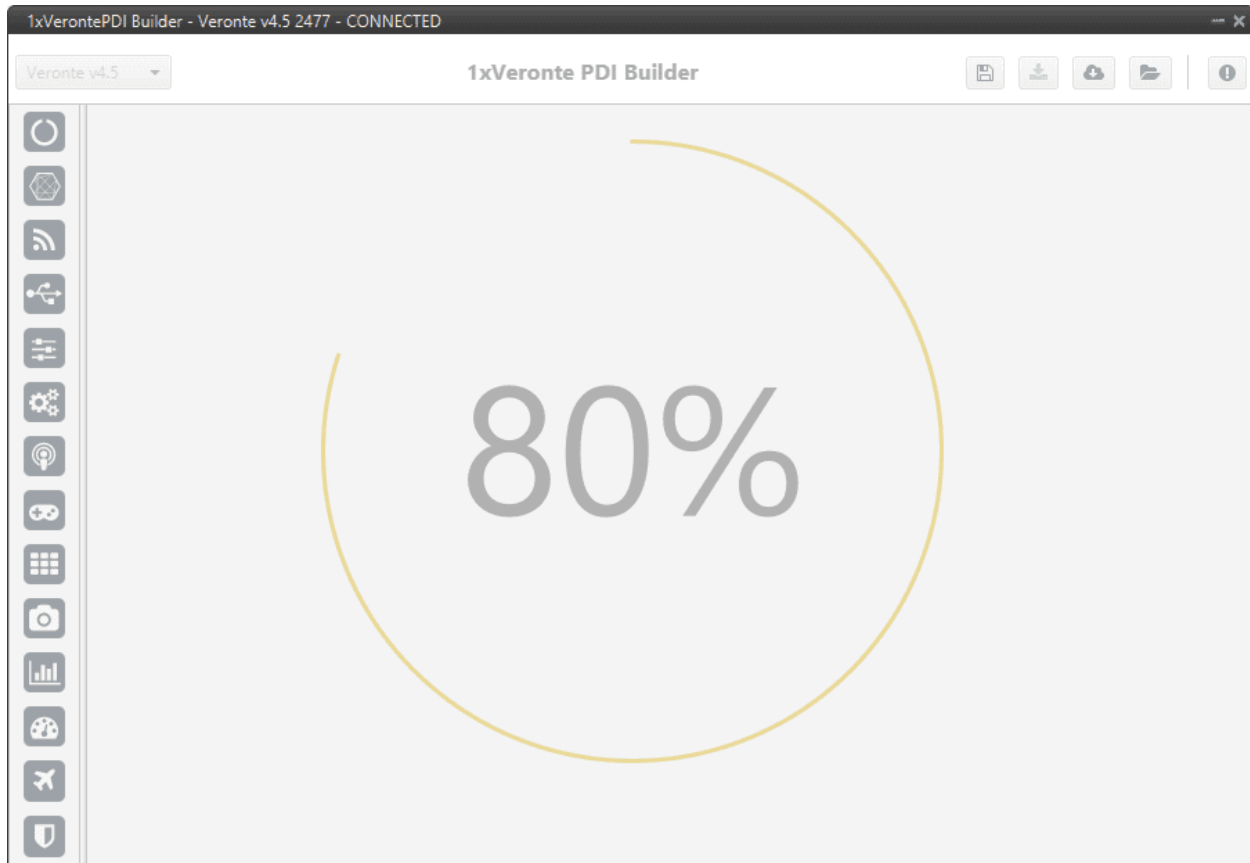


Fig. 472: Save PDI

After saving any changes, Veronte Autopilot 1x will **RESET** and the 1x PDI Builder software will **close**.

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 3 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.

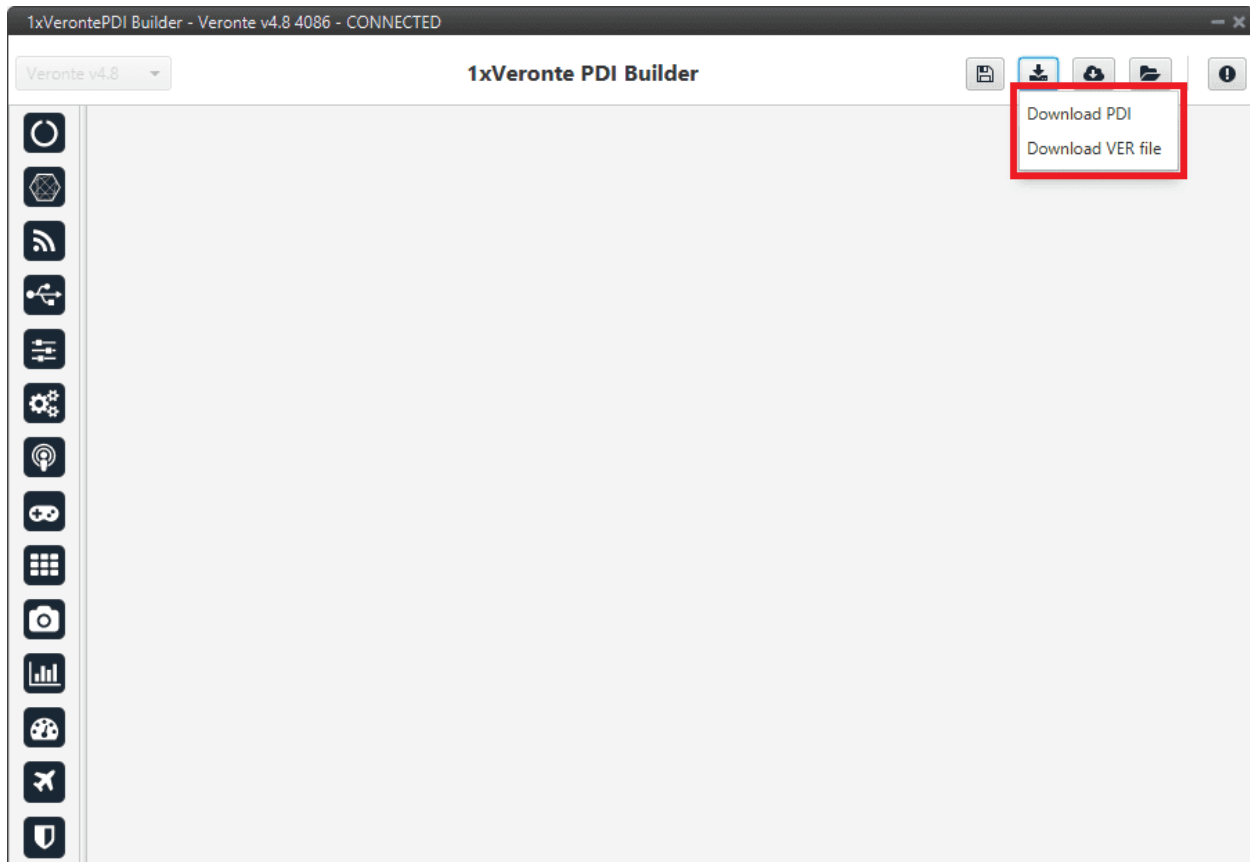


Fig. 473: Download option

3. **Import PDI from repo:** The user can import a configuration file from the repo and modify it. After that, if the save button 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 button is pressed, this configuration will be loaded into the 1x.
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](#).

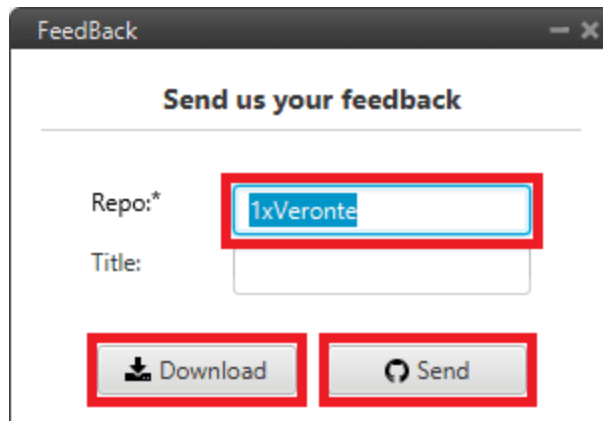


Fig. 474: Feedback

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

Icon	Item	Description
	<i>Veronte</i>	Introduce 1x autopilot information
	<i>Connections</i>	Configure I/O connections on 1x autopilot
	<i>Sensors</i>	Configure parameter sensors
	<i>Input/Output</i>	Configure external sensors/devices and I/O signals
	<i>Control</i>	Introduce Phases, Envelopes, Modes and Arcade axis configuration
	<i>Automations</i>	Configure automatic actions on event detection (go home, change phase...)
	<i>Communications</i>	Configure alternative communication channels, statistics and routing
	<i>Stick</i>	Cusomize transmitter configuration
	<i>Block Programs</i>	Customize algorithms executed by 1x autopilot
	<i>Devices</i>	Configure any connected devices: servo, radio, camera...
	<i>Telemetry</i>	Customize traffic: log, telemetry...
	<i>UI</i>	Customize variable names
	<i>HIL</i>	Configure parameters for XPlane Simulator
	<i>Safety</i>	Customize checklist, block user control in PDI configuration and safety bits

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 **1x Autopilot 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 section**. 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 section](#) of this manual.

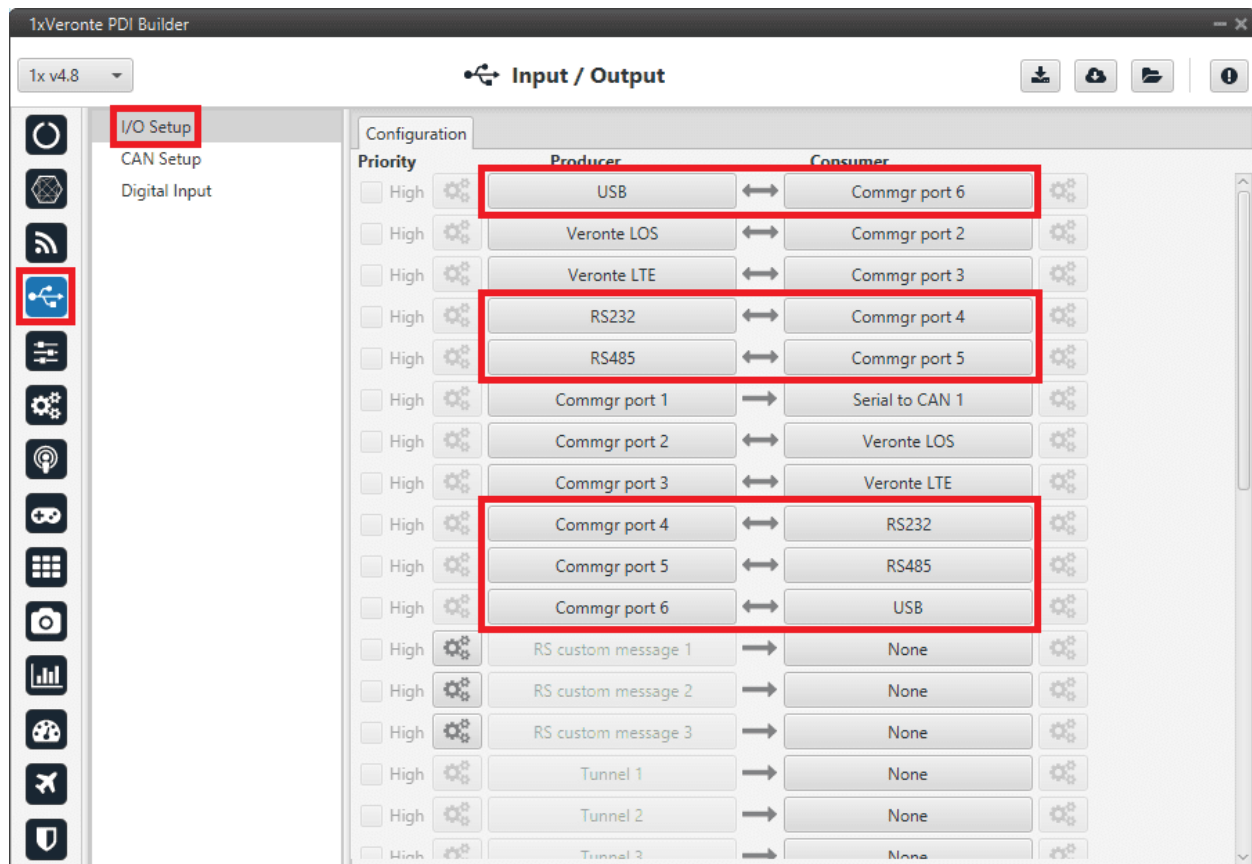



Fig. 1: USB/RS232/RS485 ↔ Commgr port

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 u1-u4** and the **output** variables **Stick Input d1-d4**.
- Finally, **enable the block to be commanded** by simply clicking on the  icon.

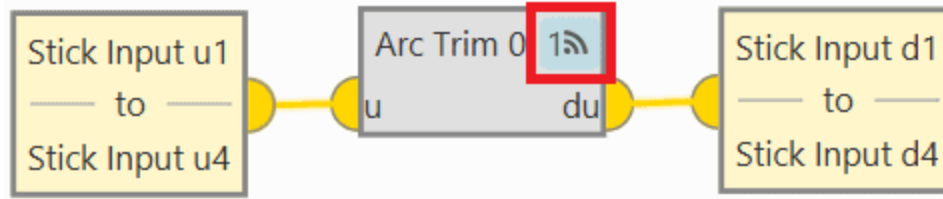


Fig. 2: Arc Trim block

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** \Rightarrow **0.5**.
- If the signal ranges from **-1 to 1** \Rightarrow **0**.

In this example, since the signal is in the range 0-1, 0.5 is set:

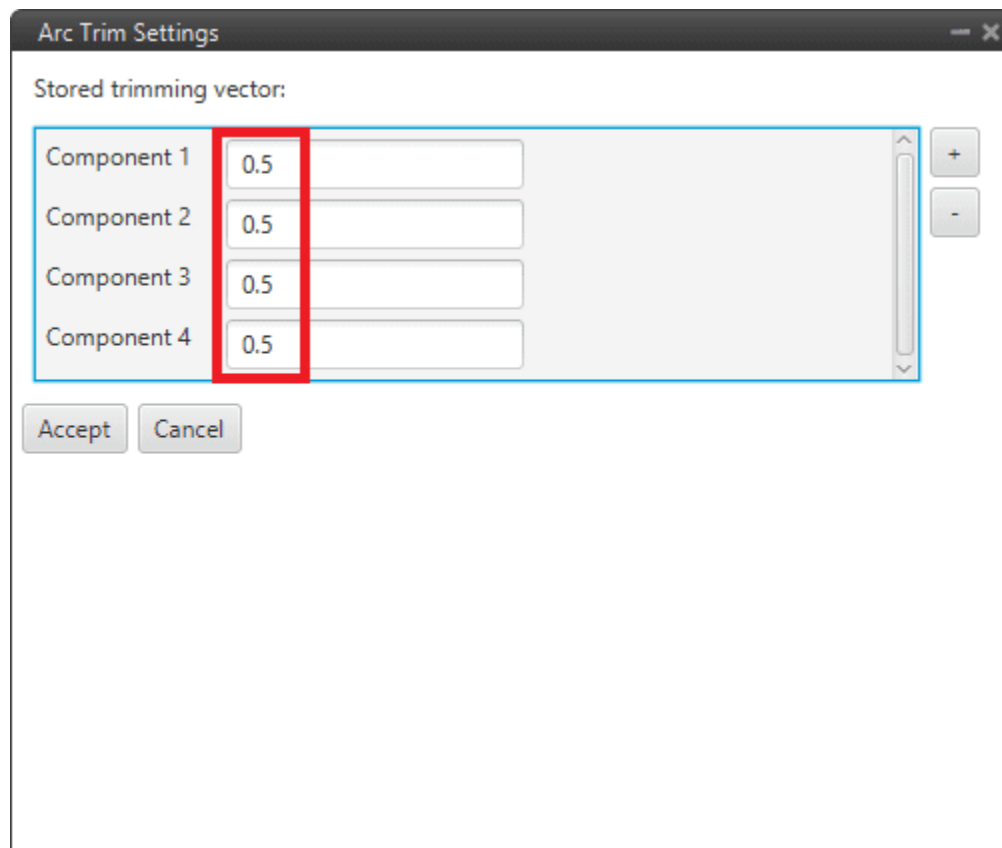


Fig. 3: Arc Trim block configuration

3. Go to **Automations** menu \rightarrow create a **New Automation** \rightarrow 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.

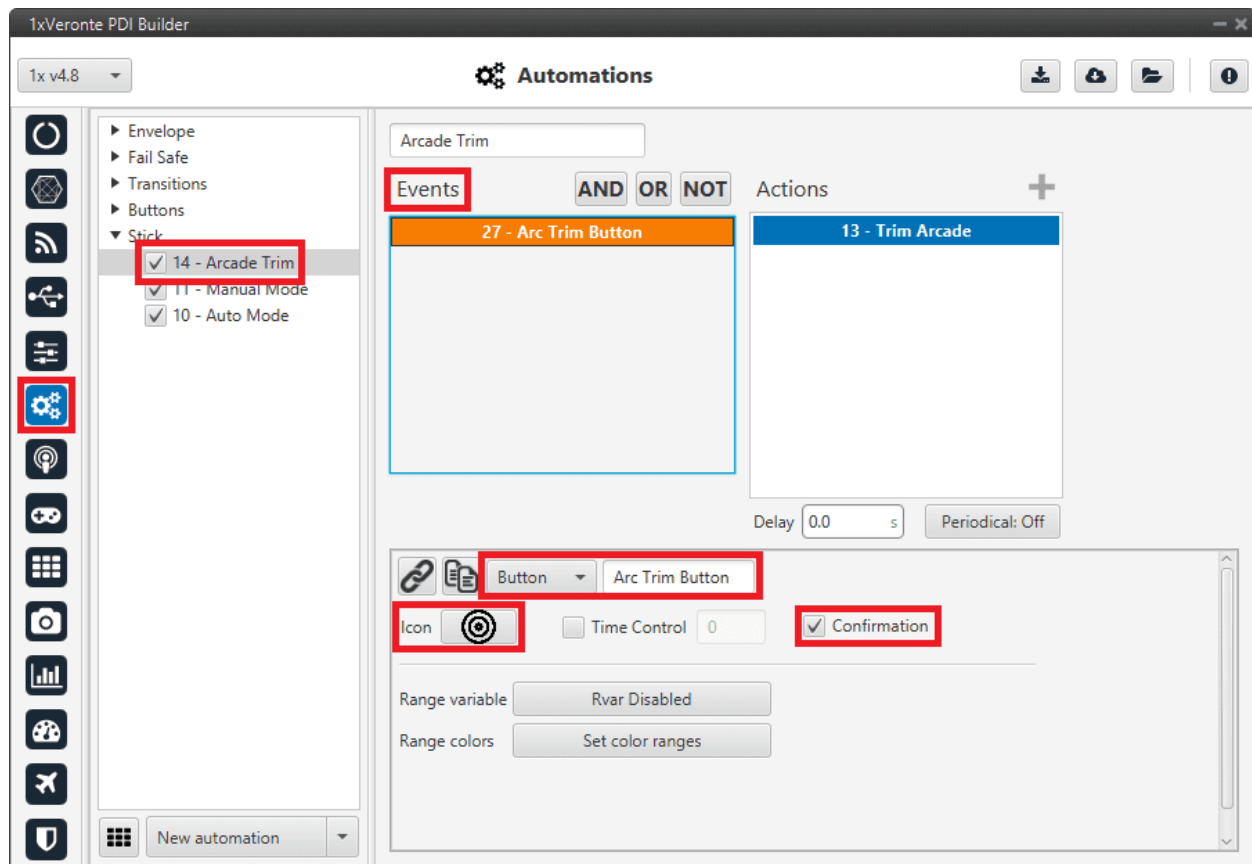


Fig. 4: Arcade trim automation - Events

4. In the created automation, go to **Actions**.
 - Add the **Command block** action.
 - Select **Arc Trim** block to command and choose the commandable **Id**.
 - Finally, it is recommended to activate both checkboxes:

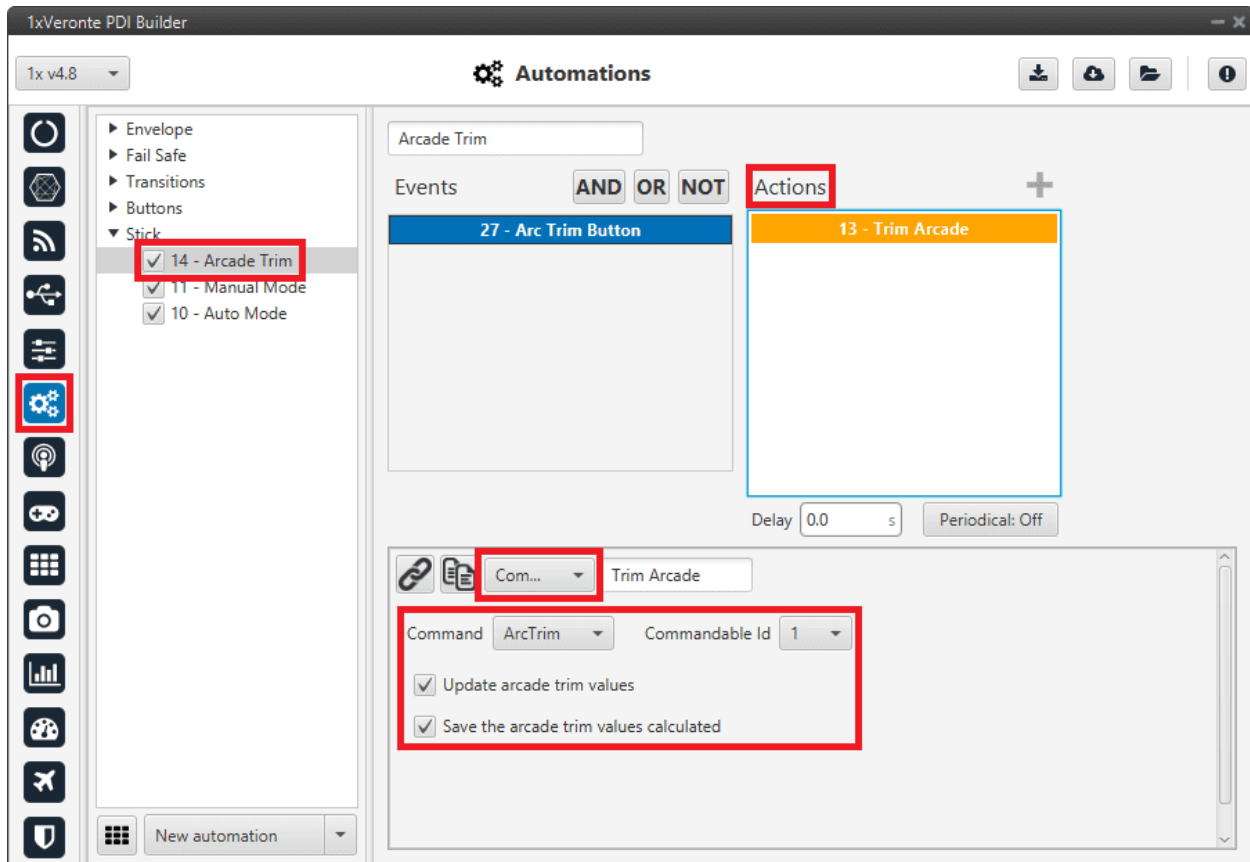


Fig. 5: Arcade trim automation - Actions

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](#) section on the **Veronte Ops** user manual.



Fig. 6: Arc Trim button - Veronte Panel

When clicking on it, the following confirmation message will be displayed (as the confirmation checkbox has been activated in the automation):

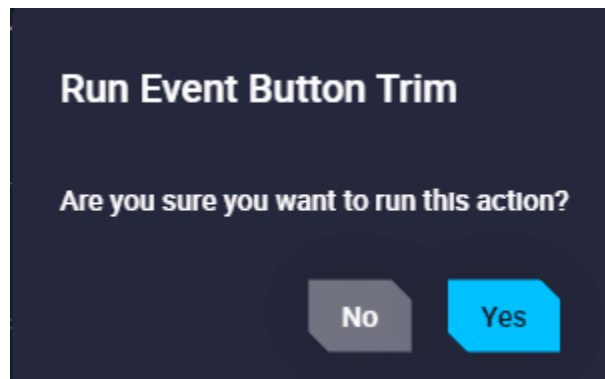


Fig. 7: Arc Trim button - Veronte Panel

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:

CAN messages reception

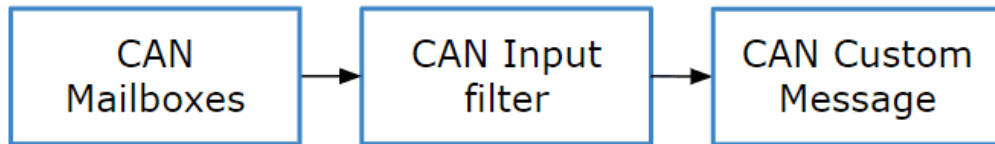


Fig. 8: CAN messages reception diagram

1. Go to Input/Output menu → CAN Setup section → **Mailboxes tab**.

Configure the mailbox to receive a message with the appropriate **ID** (in this example ID 28 has been configured):

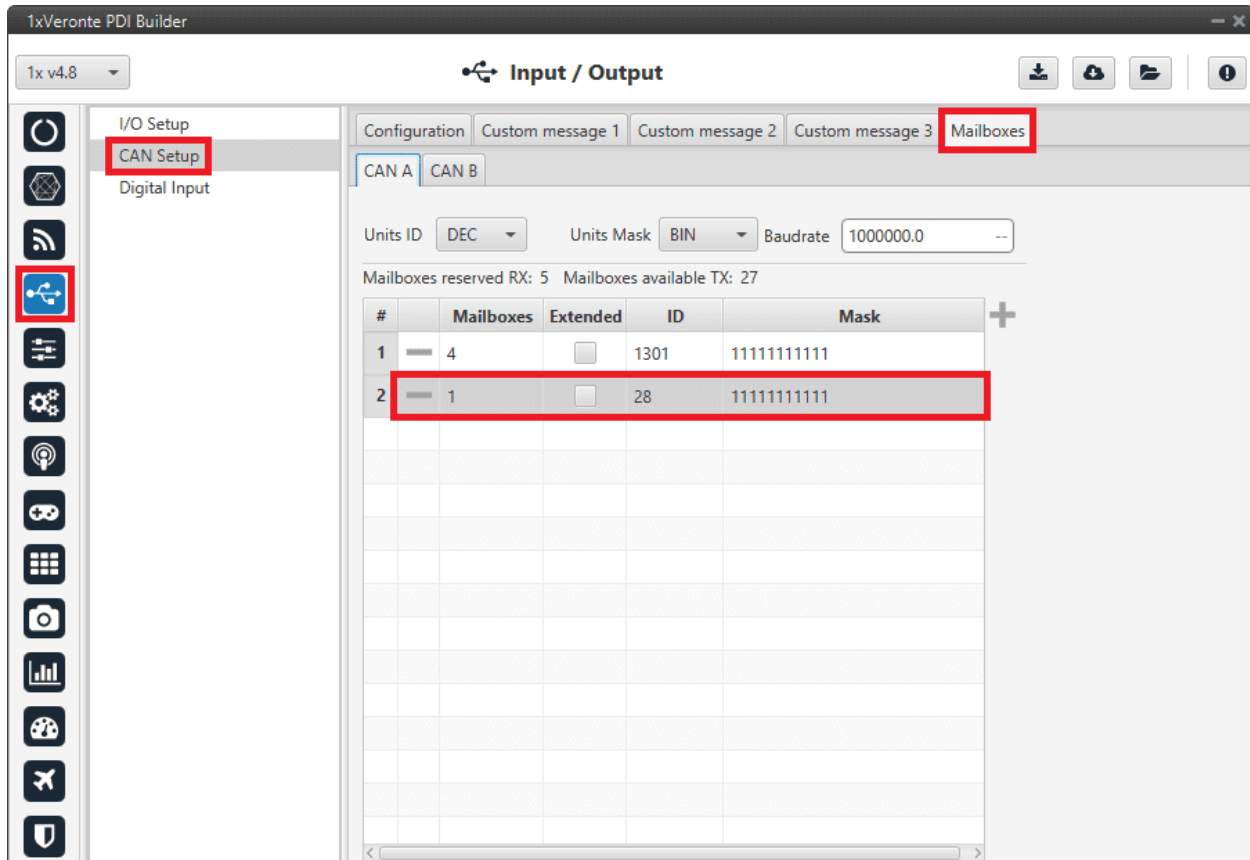


Fig. 9: Mailbox configuration

2. Go to Input/Output menu → CAN Setup section → **Configuration tab**.

Connect an **Input filter** with the **right CAN ID** to a **Custom message consumer**:

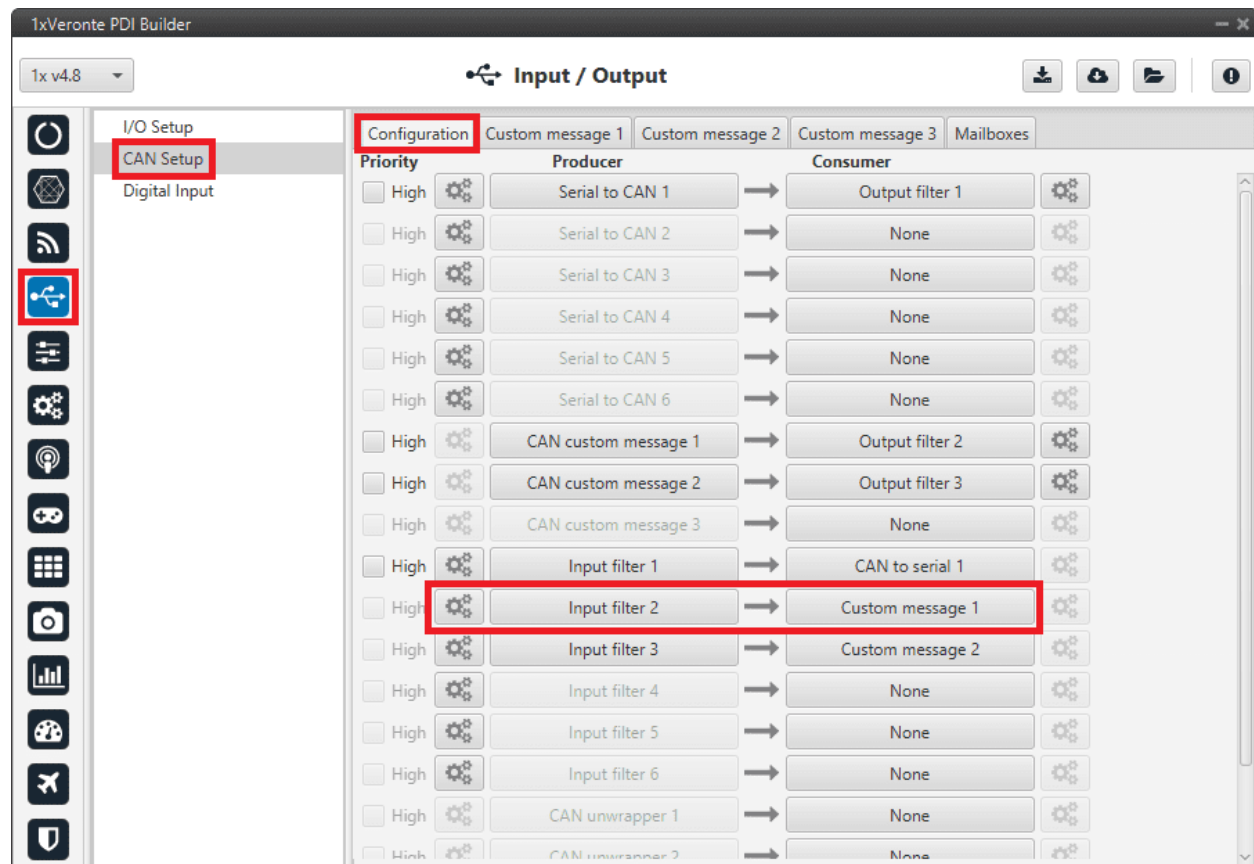


Fig. 10: Input filter

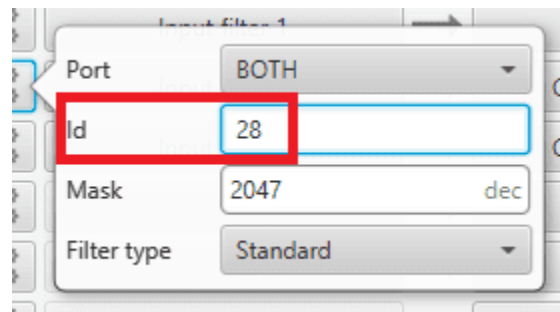


Fig. 11: Input filter configuration

- Go to Input/Output menu → CAN Setup section → **Custom message 1 tab** (as Custom Message 1 has been selected as consumer).

Configure the message reading as desired in the **RX tab** by setting the correct CAN ID.

The different options and parameters to be configured are explained in the *RX messages -> Custom messages section* of this manual.

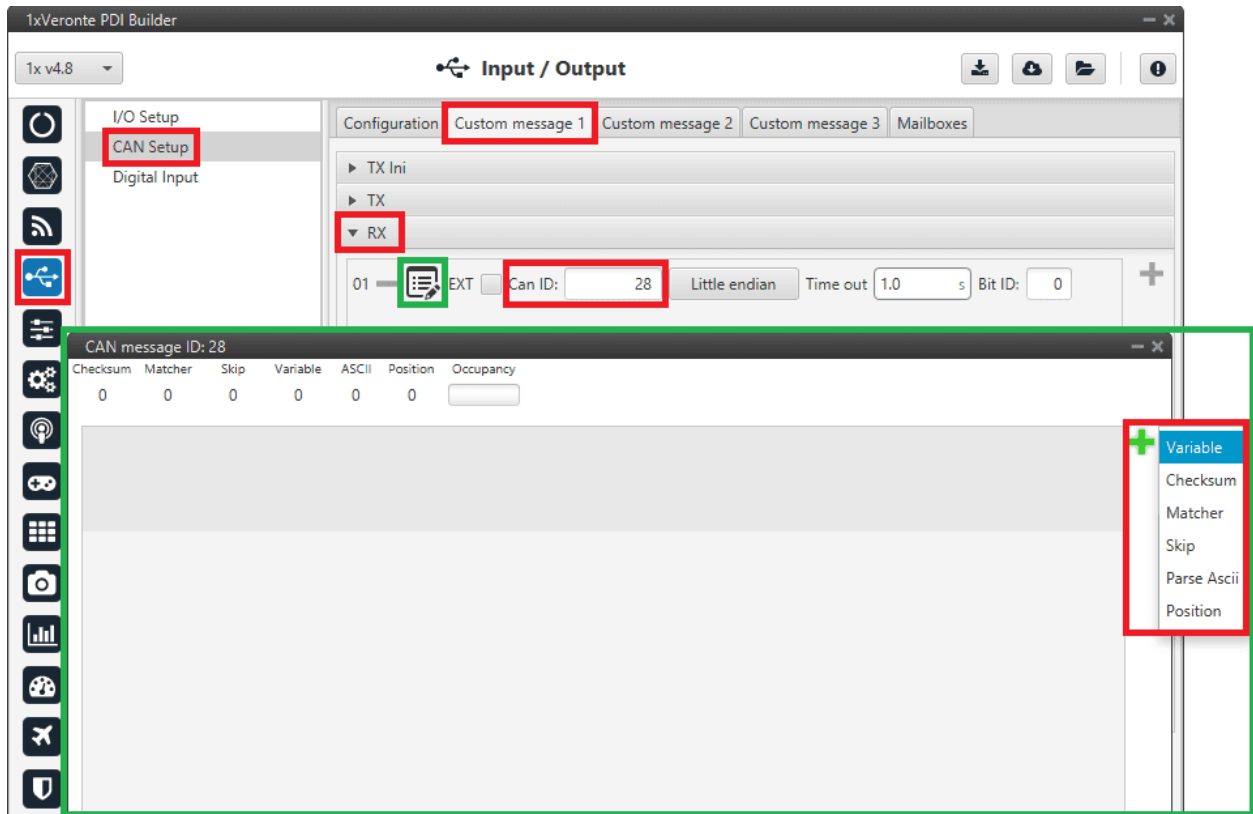


Fig. 12: Custom Message configuration

CAN messages transmission

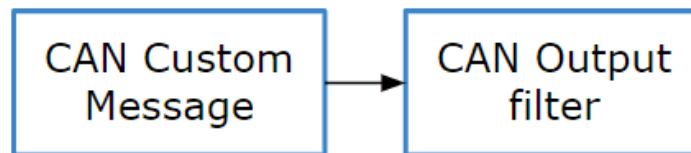


Fig. 13: CAN messages transmission diagram

1. Go to Input/Output menu → CAN Setup section → **Custom message 1** tab.

Select the fields to send in the **TX** or TX Ini section, as it is a Producer. More information on the configuration of CAN messages can be found in the *TX messages -> Custom messages* section of this manual.

For example, a CAN message set to ID 12:

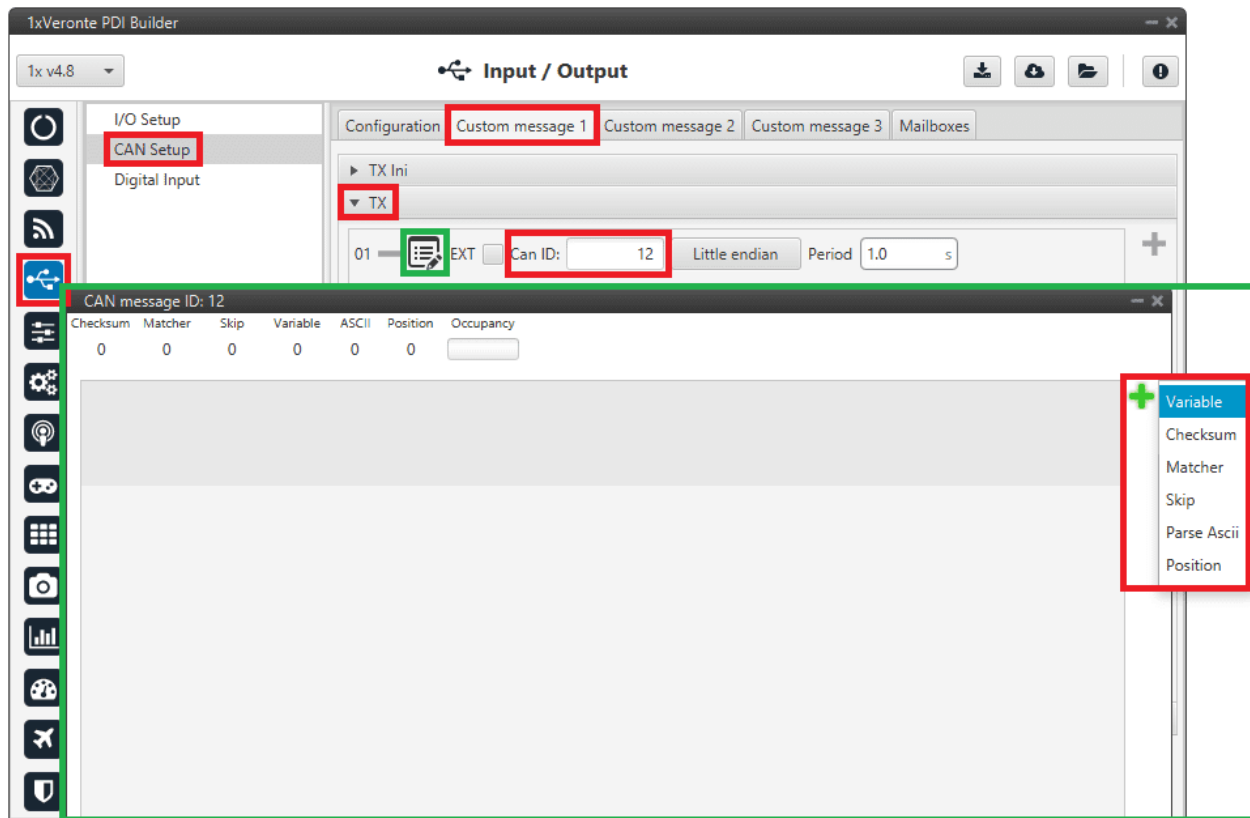


Fig. 14: Custom Message configuration

2. Go to Input/Output menu → CAN Setup section → **Configuration tab**.

Connect **CAN custom message 1** producer (as the message has been configured in the Custom Message 1 tab) to an **Output Filter** as follows:

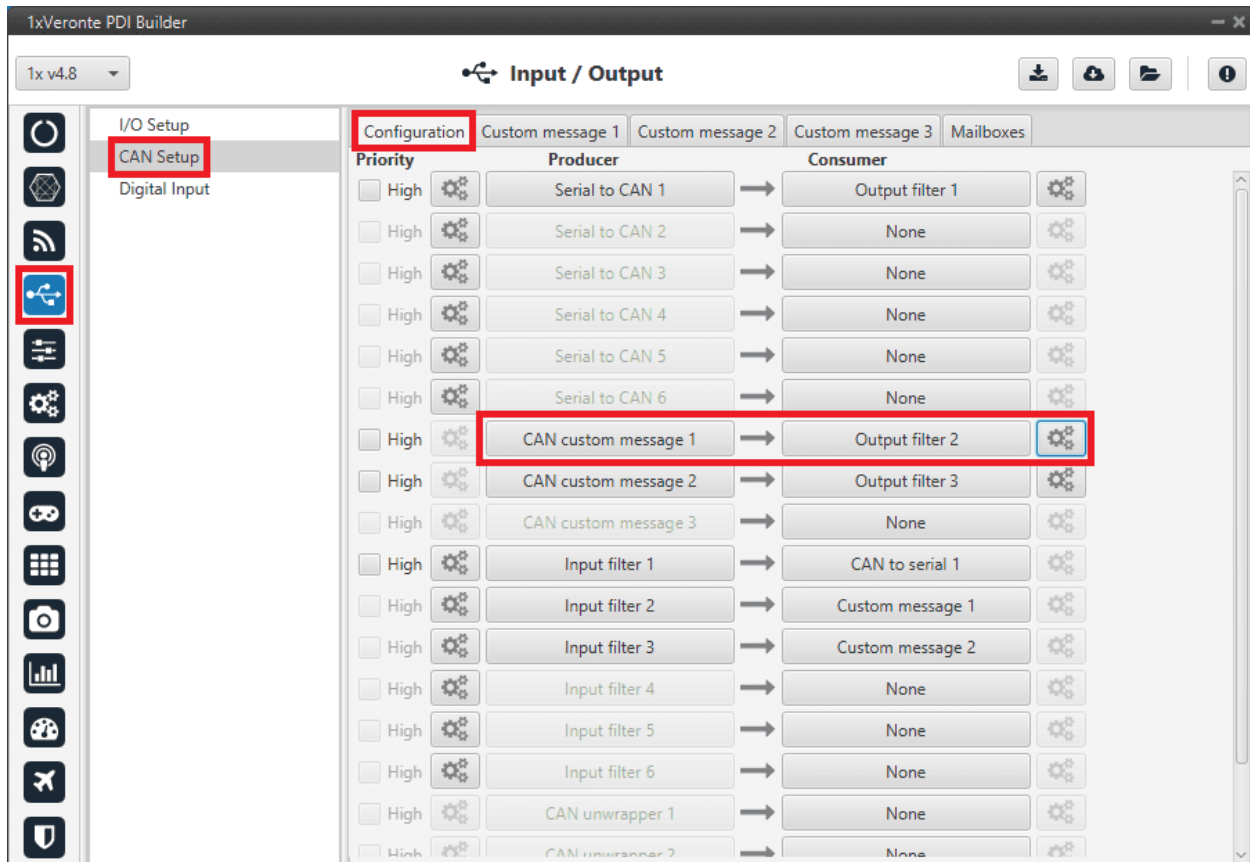


Fig. 15: Output filter

Warning: Remember that it is necessary to have **at least 1 free mailbox for TX messages**.

3.4 Data transmission between Veronte 1x Autopilots

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** section → **Data link to VApp** tab (see [Telemetry section](#), for more information about this).
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).

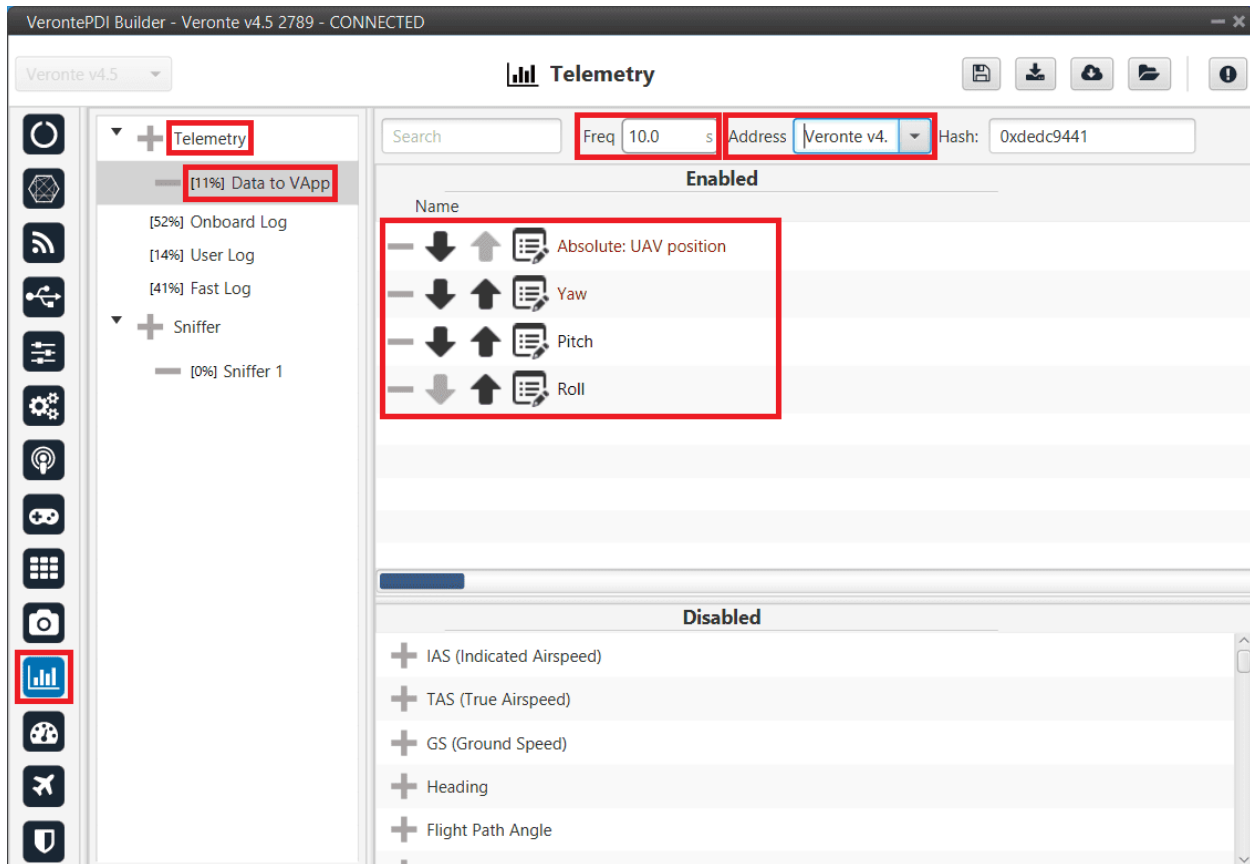


Fig. 16: 1x ground unit - Telemetry

For the **1x air unit**:

1. Go to Telemetry menu → **Sniffer** section (for more information about this, see [Sniffer section](#)).
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).

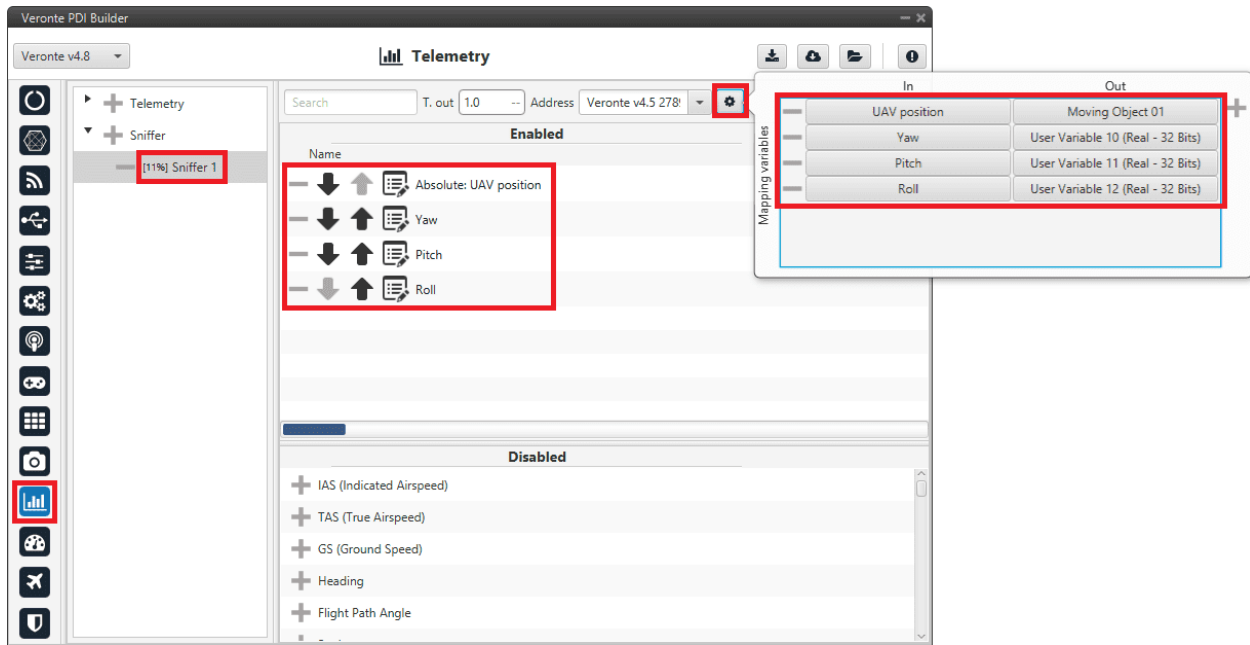


Fig. 17: 1x air unit - Sniffer

3.5 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*

3.5.1 Altimeters

3.5.1.1 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.

3.5.1.1.1 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 section [Hardware installation - Electrical](#) of the **1x user manual**).

Once connected to 1x autopilot, the value can be monitored in 1x PDI Builder by using the variables ADC1 to ADC5. For pin *ANALOG_1* the correspondent ADC variable in 1x PDI Builder is *ADC1*.

1. Go to Connections menu → **ADC 1 section**.

Click on 'Create new program':

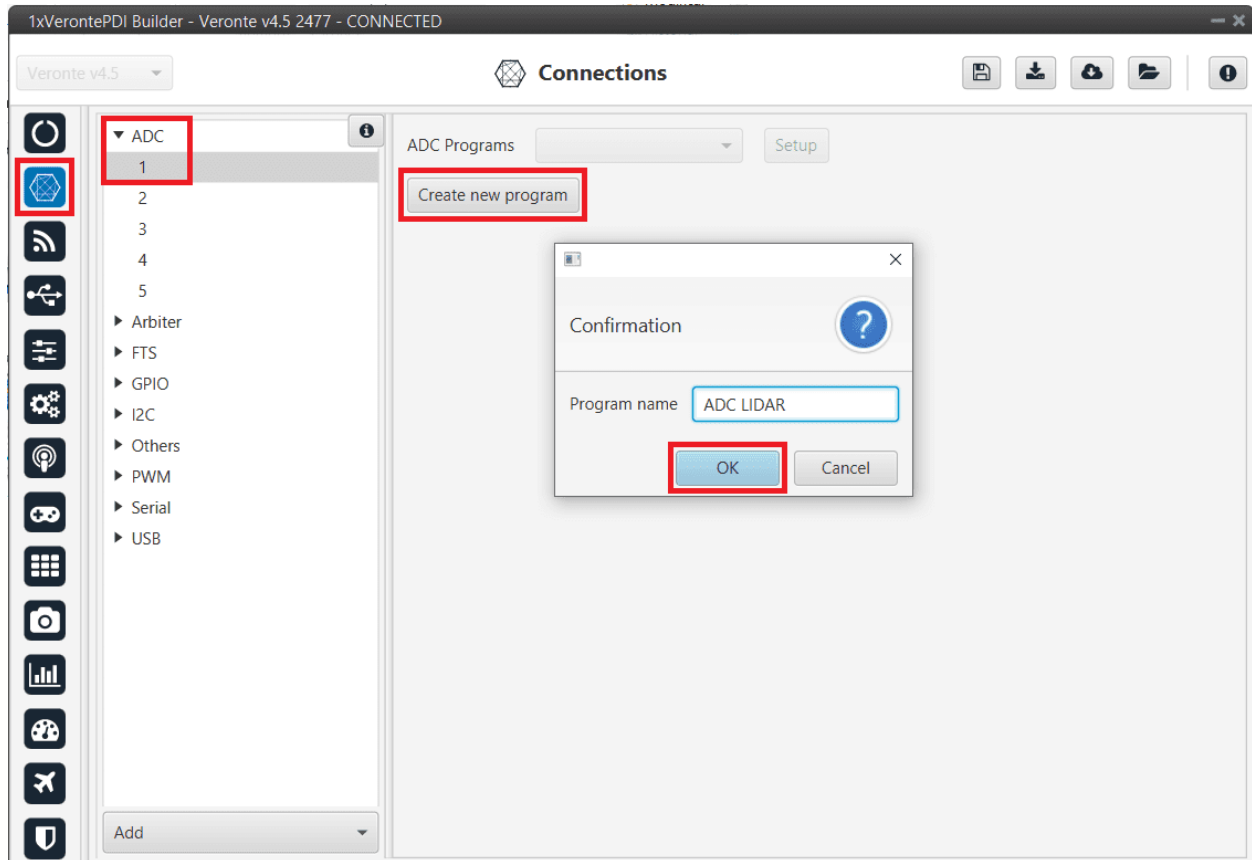


Fig. 18: Create ADC program

2. Go to **Block programs menu**.

Configure the following operation (for more information about blocks, see [Block Programs](#)):

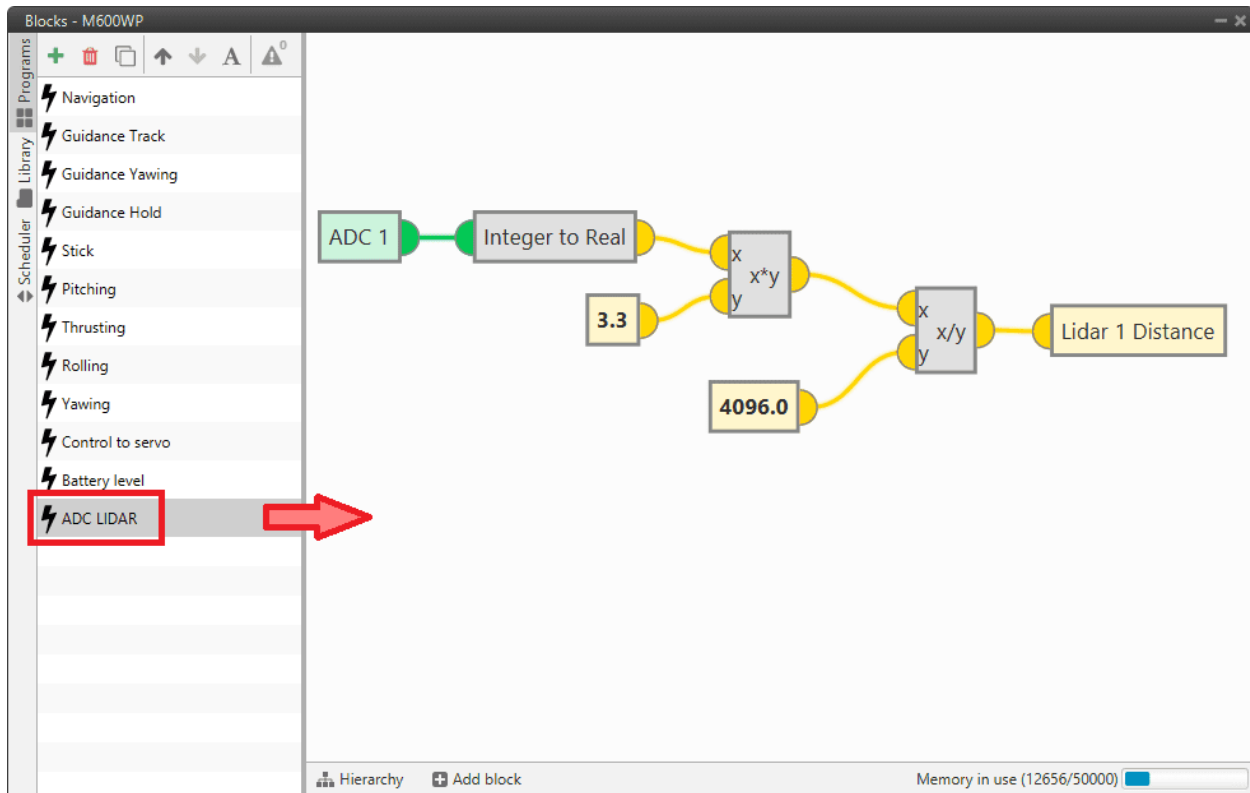


Fig. 19: Lidar operation

After implementing the operation the variable *Lidar 1 Distance* will represent the distance measured by the lidar.

Note: As 4x Autopilot can read up to 36 V per each ADC, the 3.3 value of the ADC program must be changed to 36 if applicable.

3.5.1.1.2 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 [Hardware installation - Electrical](#) section of the **1x user 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 section**.

- Enable Lidar 1
- 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 section](#) of this manual.

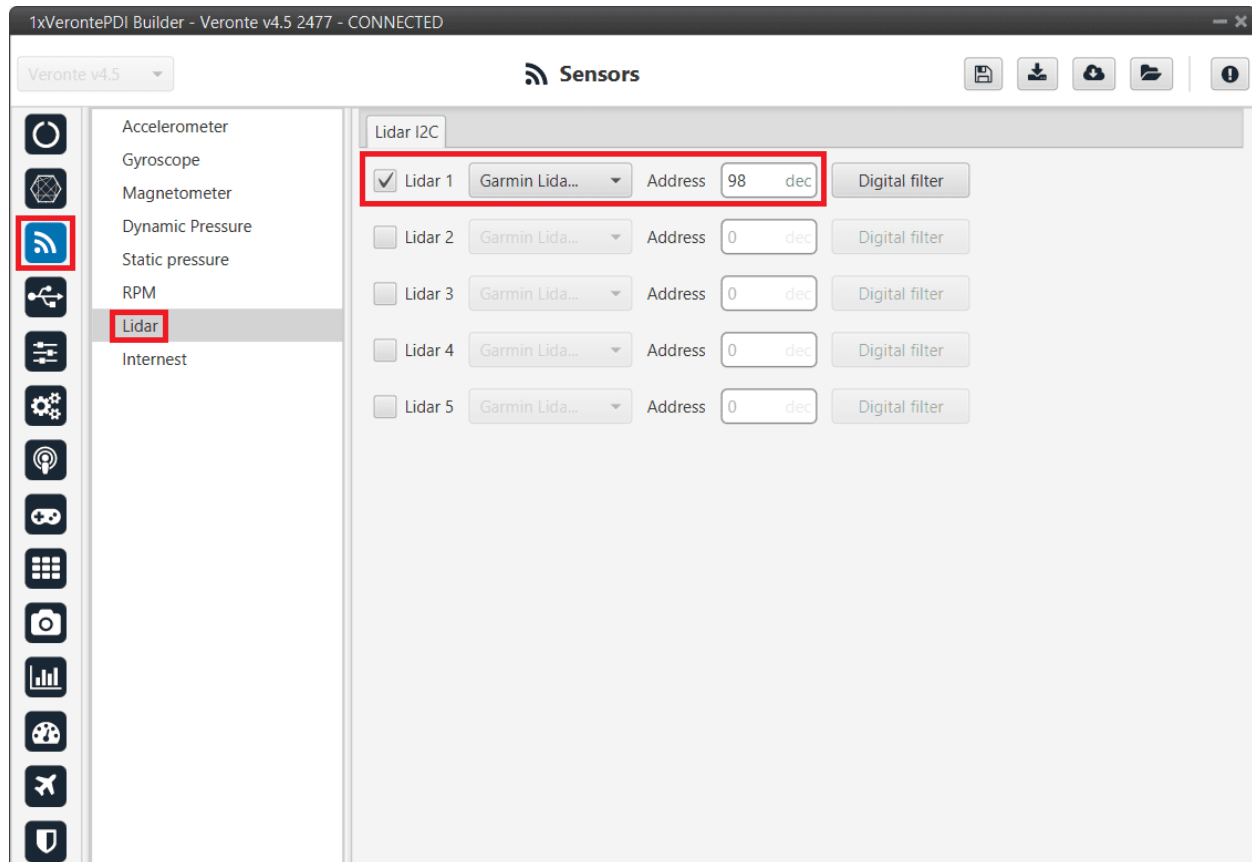


Fig. 20: I2C Lidar

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

3.5.1.1.3 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.

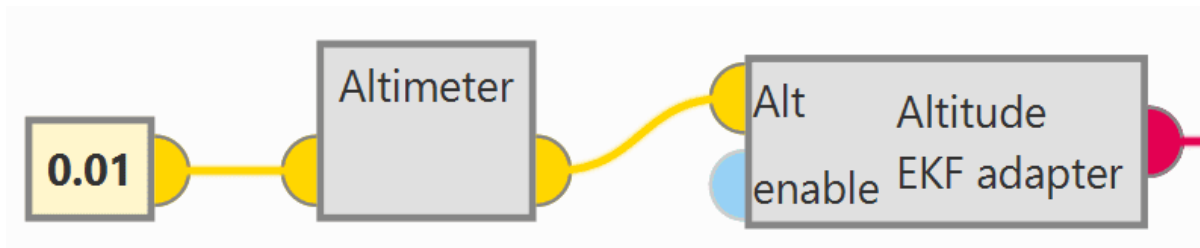


Fig. 21: Altimeter connection - Block Programs

The *Lidar Distance* variable where the lidar measurement is stored must be selected. In this example, *Lidar 1 Distance* has been used:

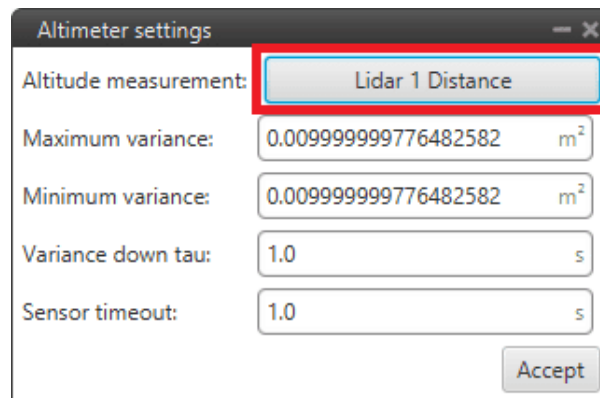


Fig. 22: Altimeter sensor configuration - Block Programs

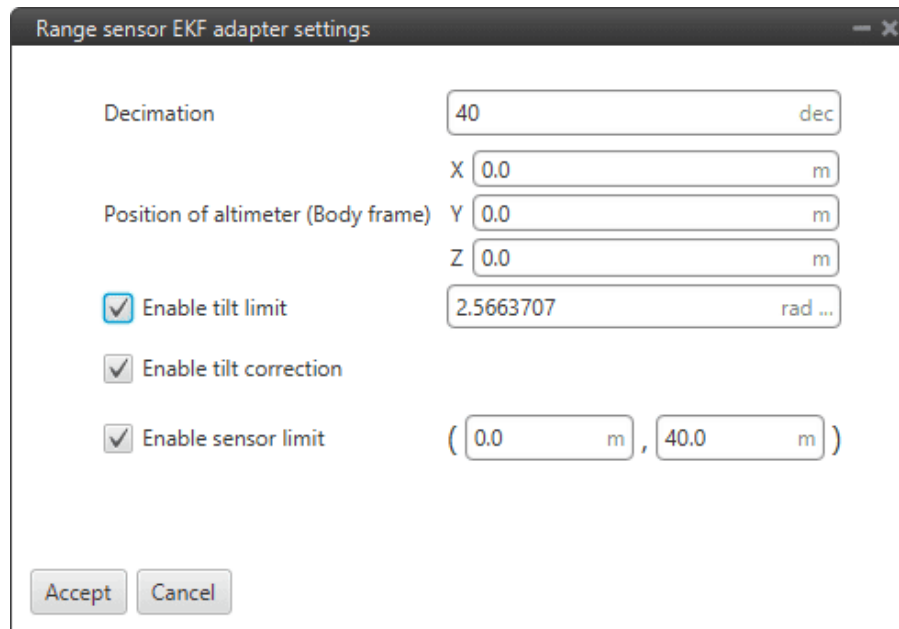


Fig. 23: Altitude EKF adapter configuration - Block Programs

For more information on these blocks, see [Altimeter sensor](#) and [Altitude EKF adapter](#) 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.

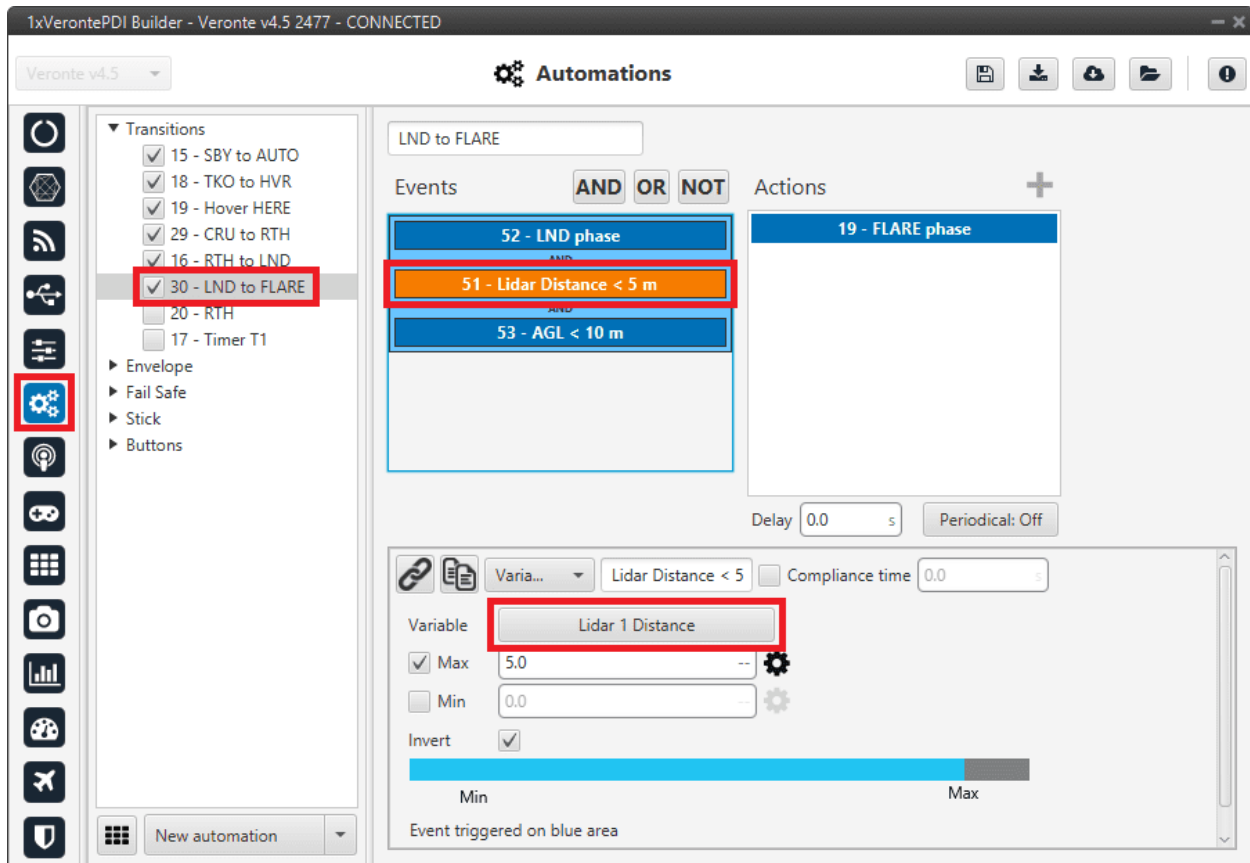


Fig. 24: Lidar automation example

For more information on automations, see [Automations section](#).

3.5.1.2 Radar

Radar altimeters are common devices on aircrafts.

3.5.1.2.1 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 section → **Configuration tab**.

Connect an Input Filter to **Custom message 1**, in this example *Input Filter 2*. and configure to read from **CAN A**:

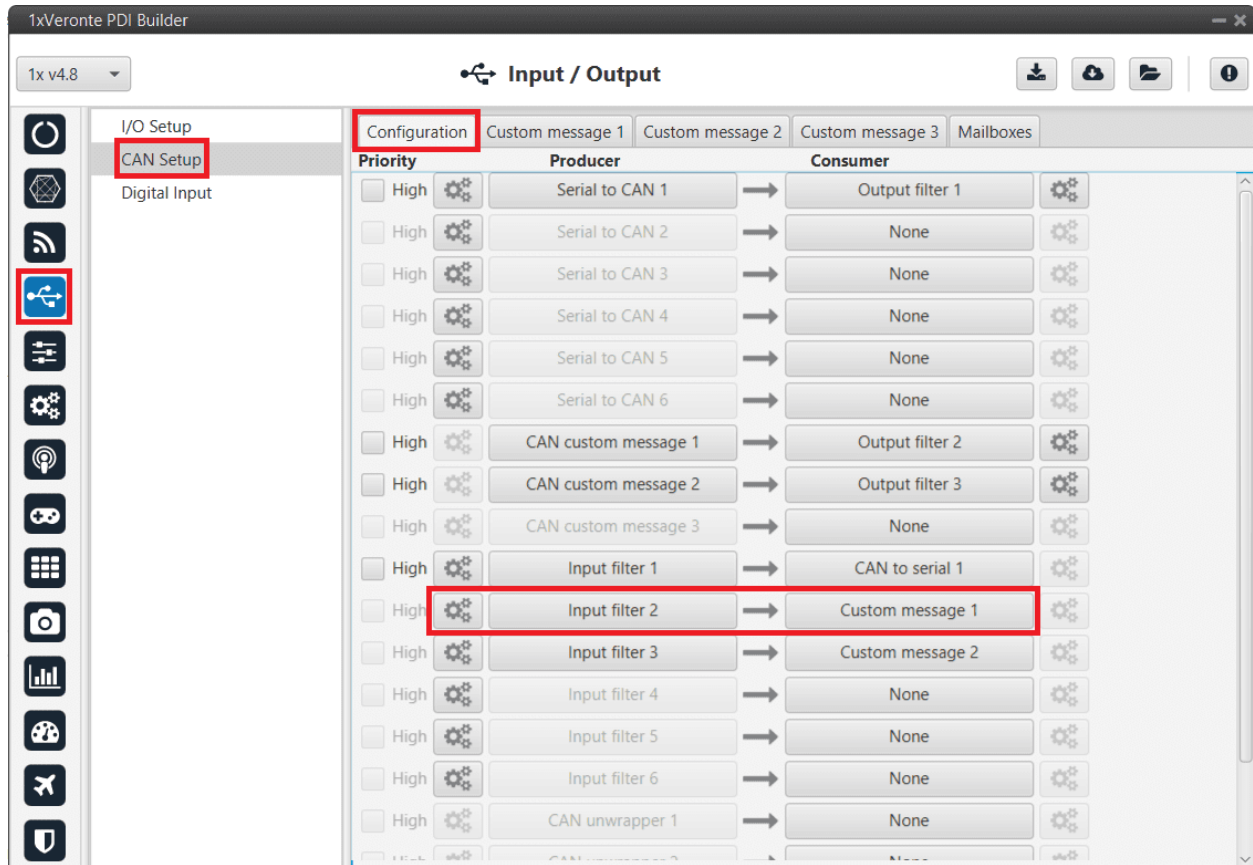


Fig. 25: CAN Setup - Input Filter

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**).

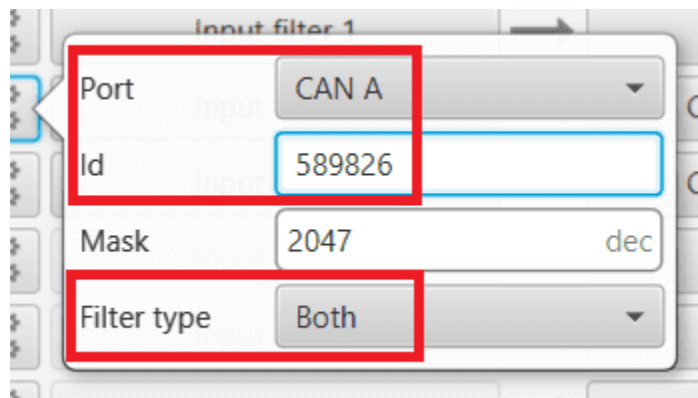


Fig. 26: CAN Setup - Input Filter configuration

- After specifying that **Custom message 1** will receive the data from **CAN A**, go to **Mailboxes tab**.

Configure a **CAN A mailbox** for **extended CAN ID message: 589826**:

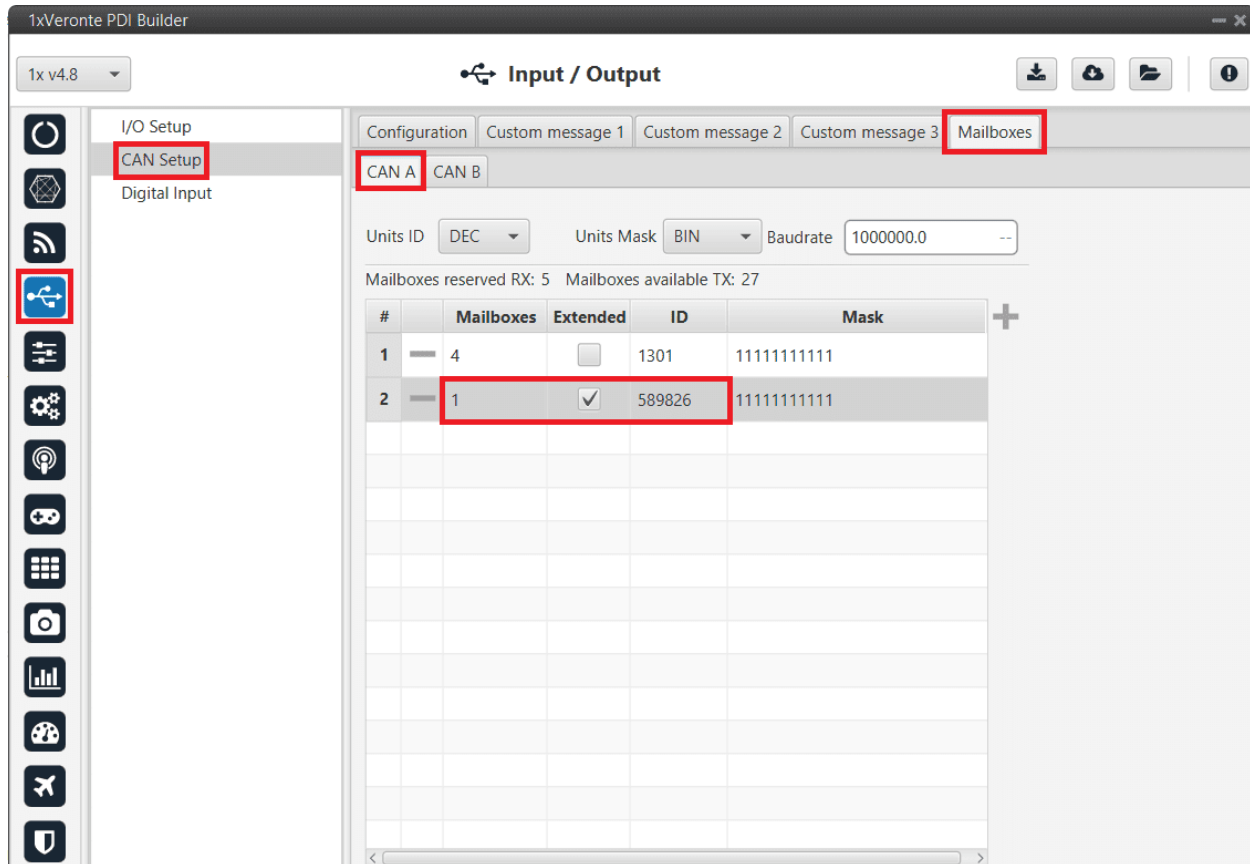


Fig. 27: Mailboxes configuration

- Go to UI menu → Variables section → **Reals Vars tab**.

Rename a User Variable that will be used to store the measurement read from the radar:

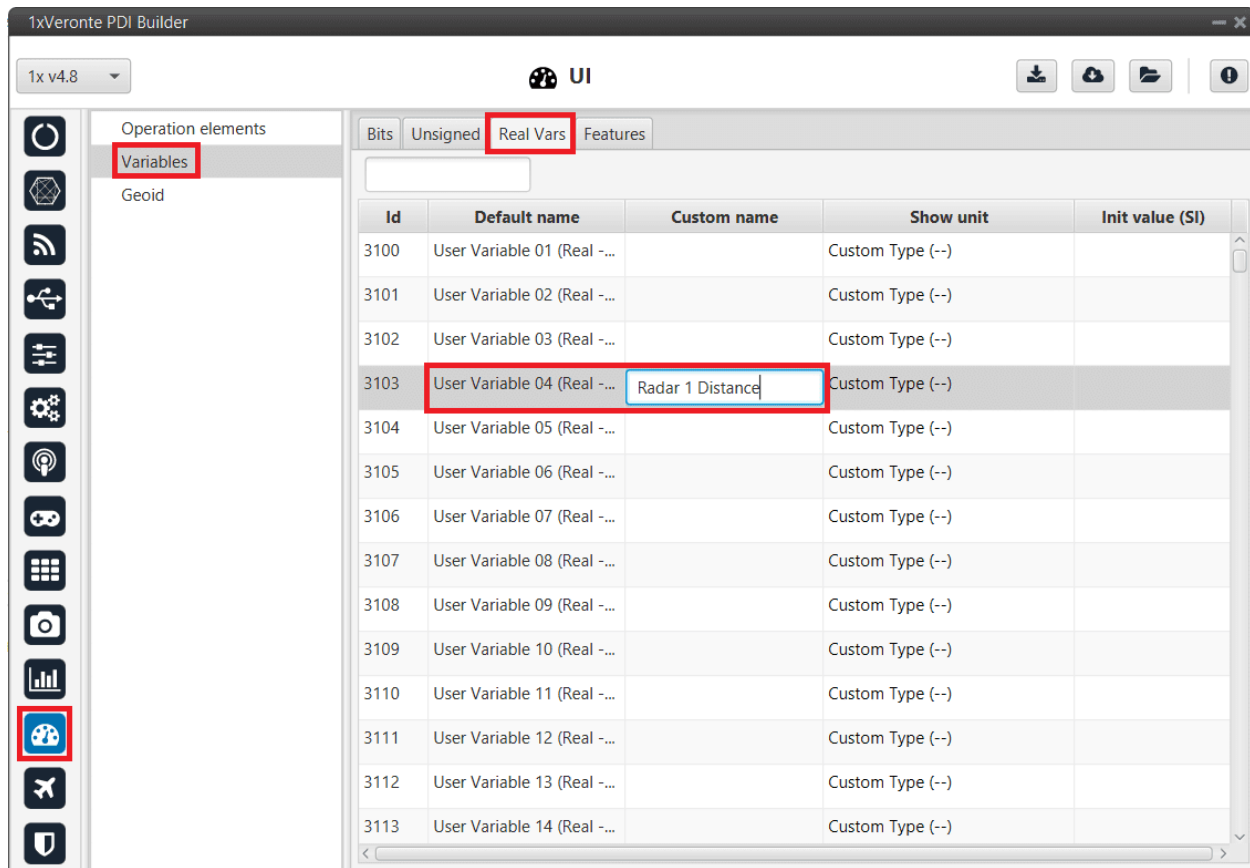


Fig. 28: User Variable renamed

4. Finally, go to Input/Output menu → CAN Setup section → **Custom message 1 tab**.
 - Add a new message in **RX** with **extended ID 589826** and **Big endian**:

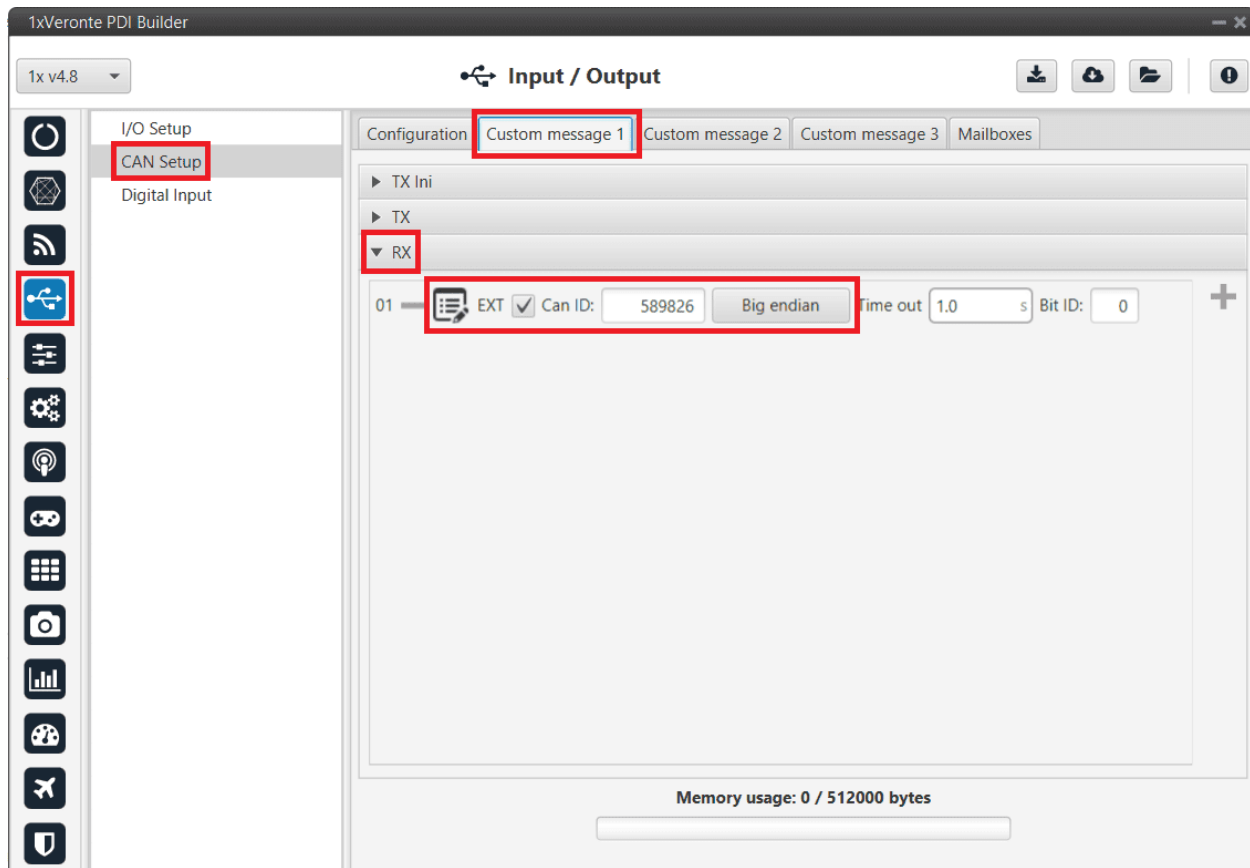


Fig. 29: Custom message 1

- Configure the reading of the message and store the received value in the user variable renamed above. And set the following values for the different parameters:

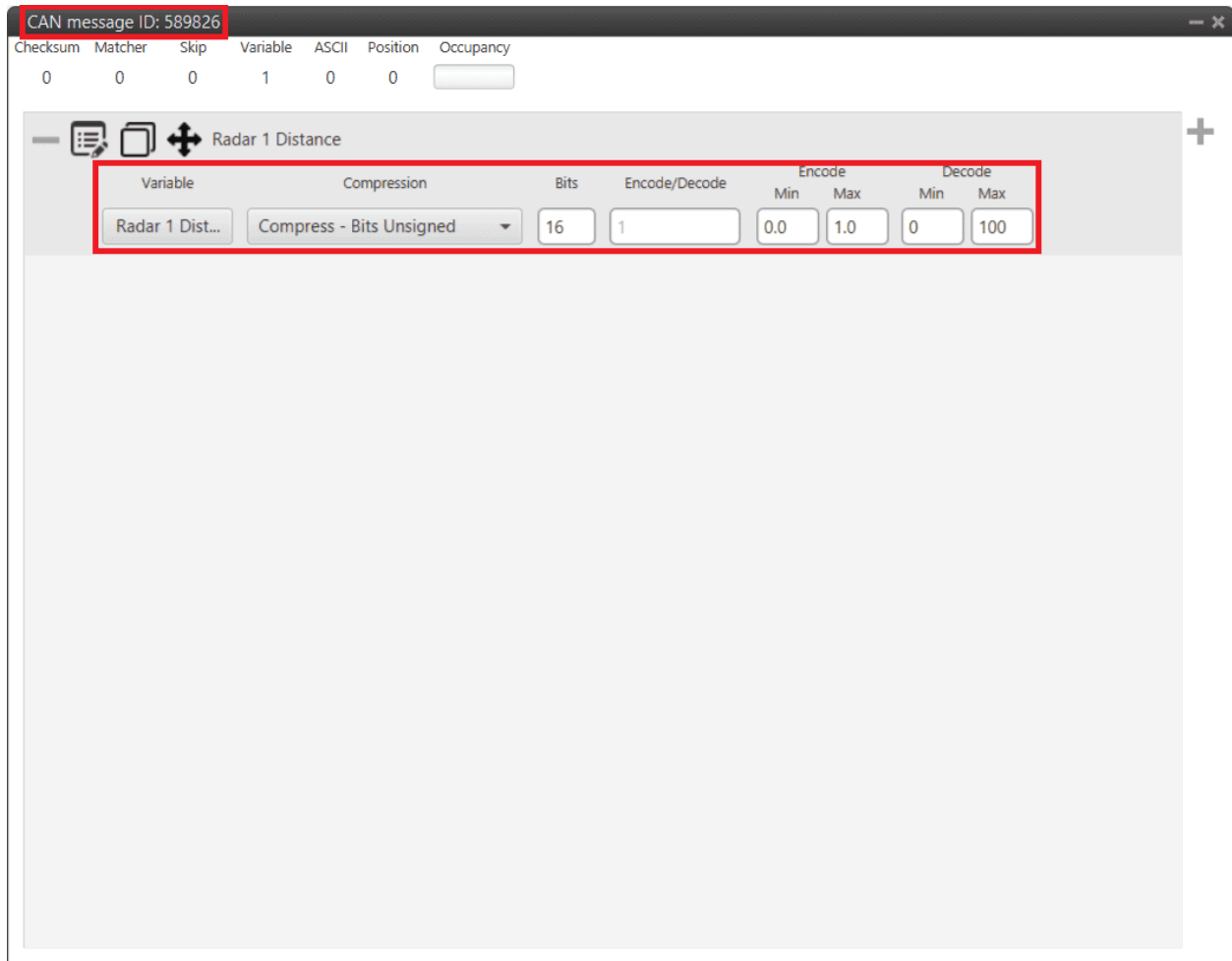


Fig. 30: Custom message 1 configuration

For more details on CAN configuration see [CAN Setup section](#).

3.5.1.2.2 Smartmicro CAN Radar

The following explanation corresponds to the integration of the **Smartmicro CAN Radar**. For more information on the Smartmicro Radar - Altimeter datasheet, the user can go to:

- [Micro Radar Altimeter Web](#)
- [Micro Radar Altimeter Data Sheet](#)

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 section → **Configuration tab**.

Set an Input Filter to read from **CAN A** in **Custom message 1**, in this example *Input Filter 3*.

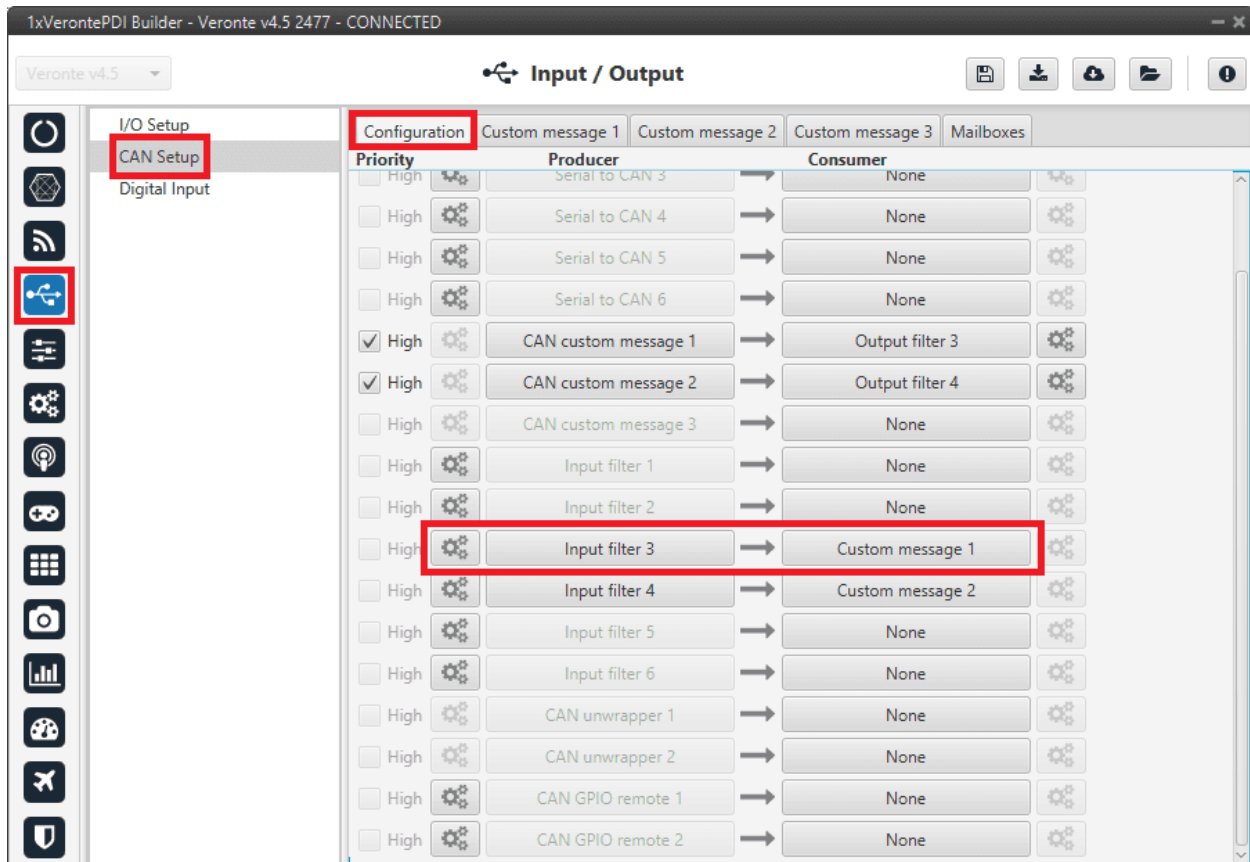


Fig. 31: CAN Setup - Input Filter

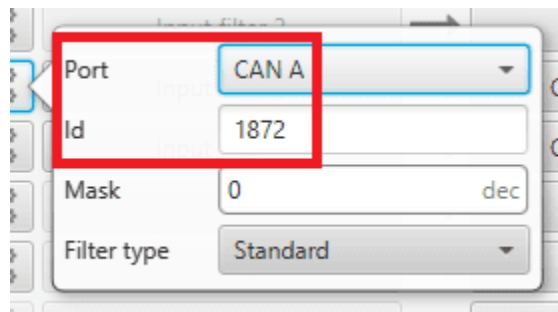


Fig. 32: CAN Setup - Input Filter configuration

- After specifying that **Custom message 1** will receive the data from **CAN A**, go to **Mailboxes tab**.

Configure the **CAN A** baudrate:

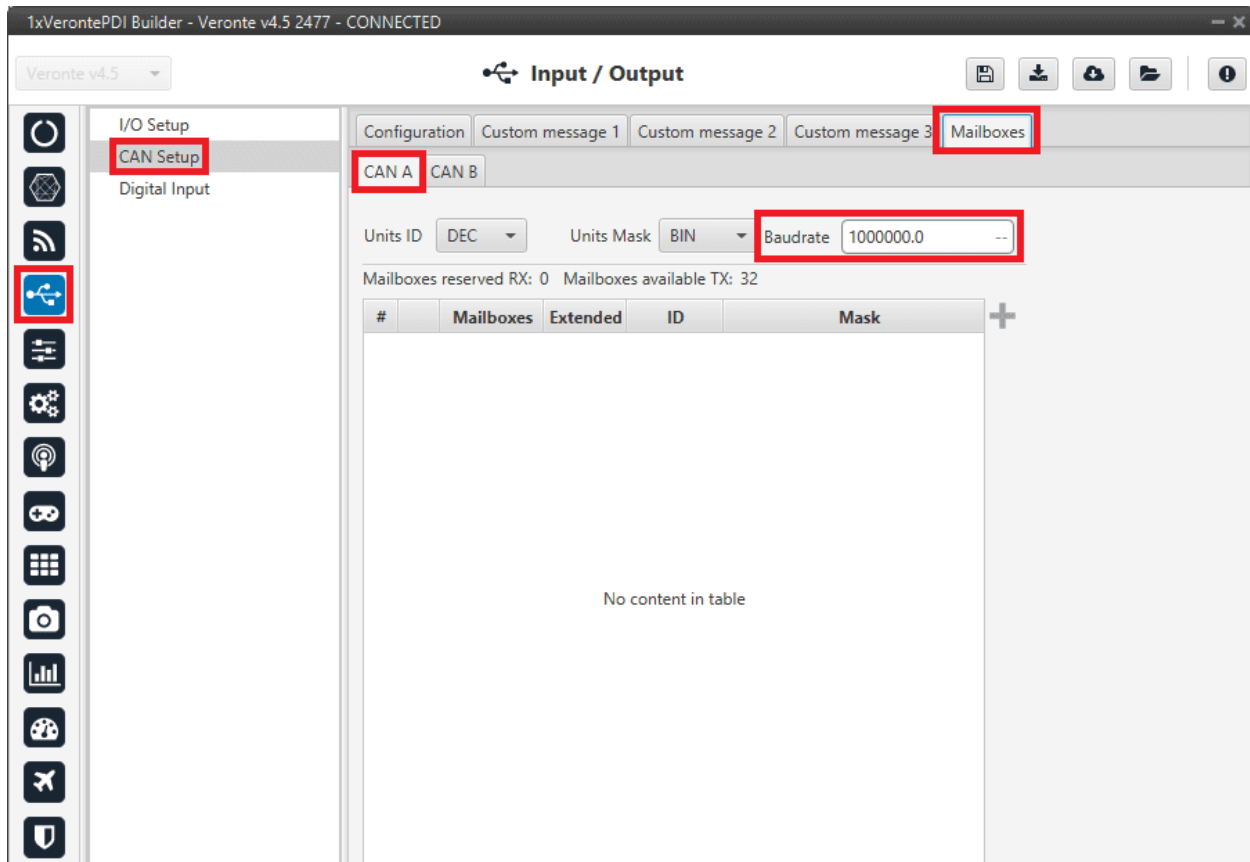


Fig. 33: Baudrate configuration

- Once the baudrate is set, configure **CAN A mailboxes** for **CAN ID message: 1872**.

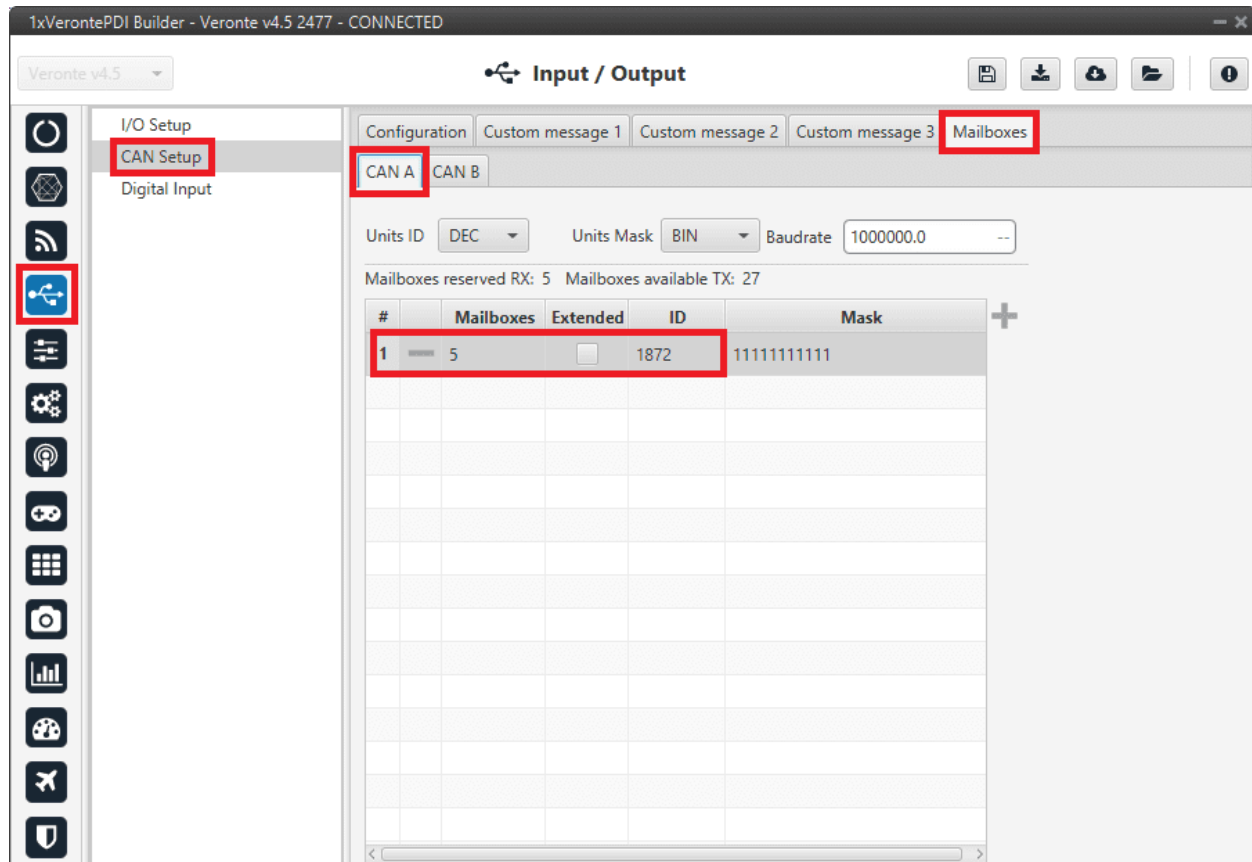


Fig. 34: Mailboxes configuration

- Go to **Custom message 1** tab.
 - Add a new message in **RX** with **ID 1872**.

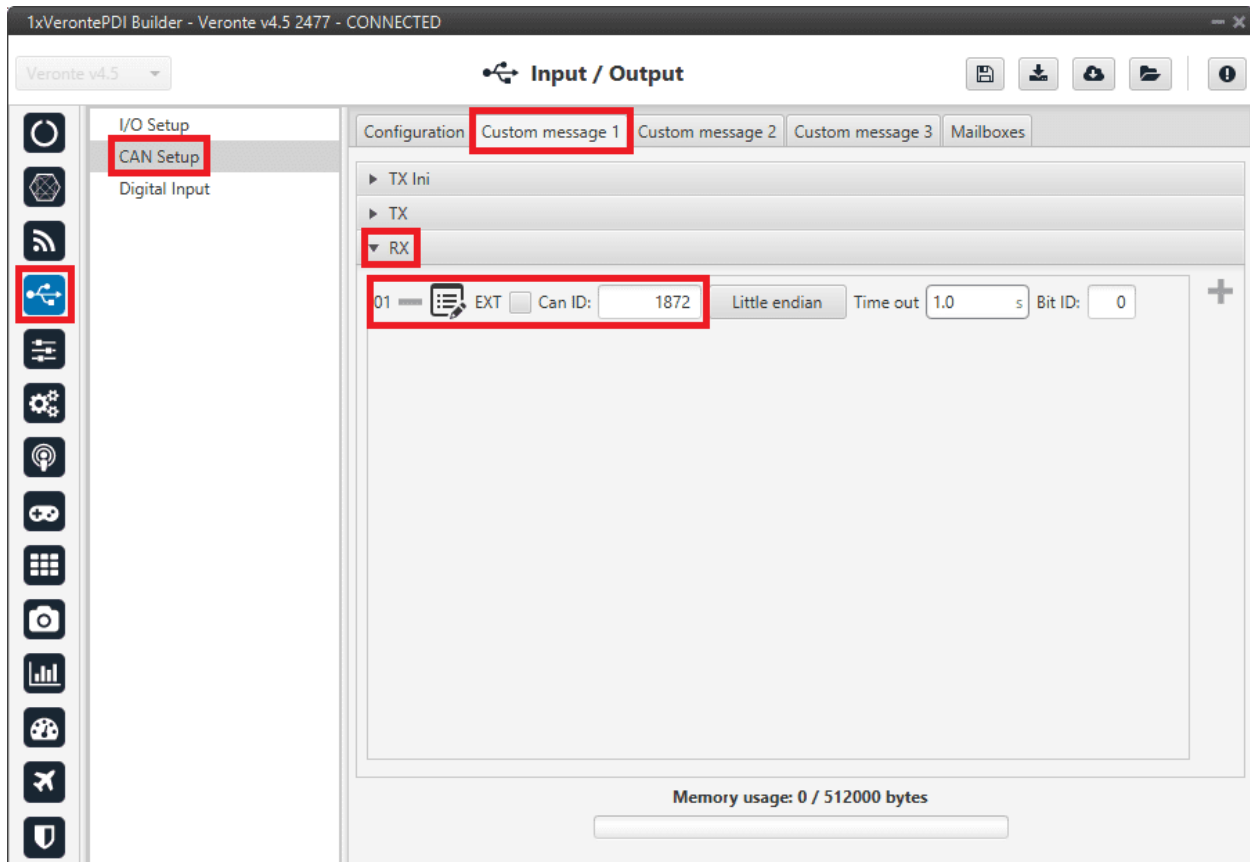


Fig. 35: Custom message 1

- Define the content of the incoming message:

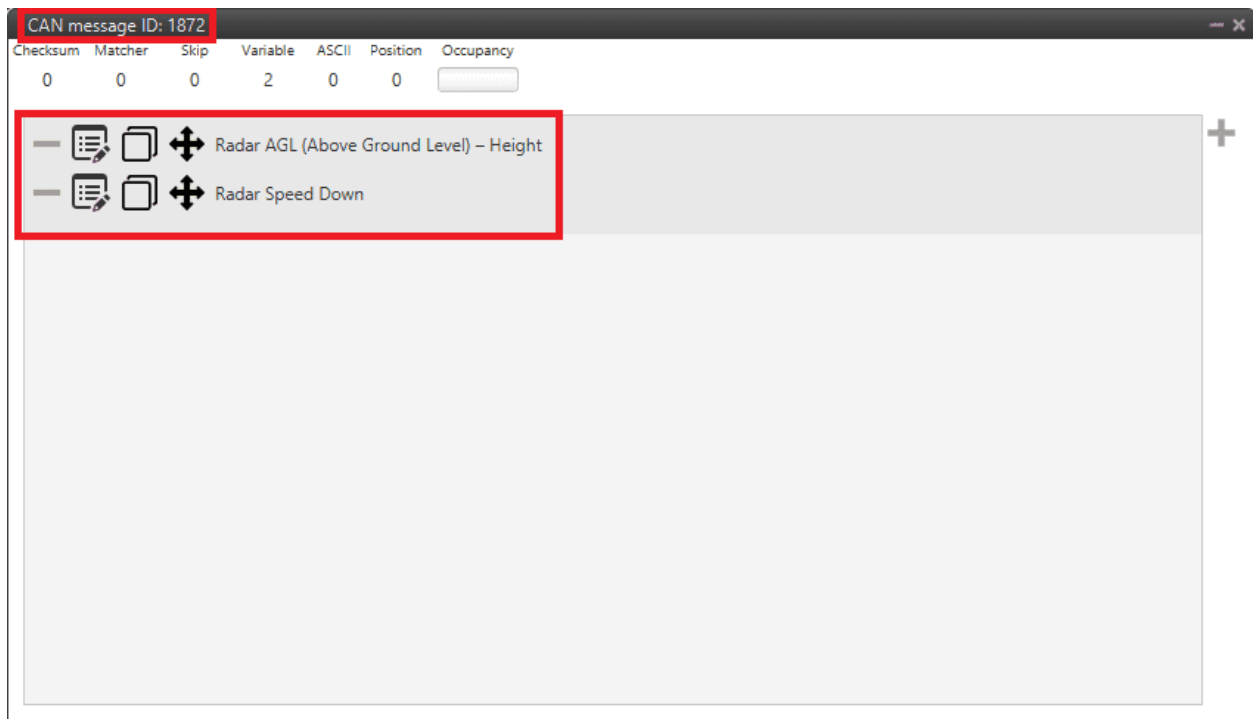


Fig. 36: Custom message 1 configuration

For more details on CAN configuration see [CAN Setup section](#).

Note: CAN ID messages and messages content will change for different Radar altimeters. Check the documentation of your device for further details.

3.5.2 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.

3.5.2.1 LM335 with Autopilot 4x

Once **LM335** sensor is wired and connected to **Autopilot 1x** or **4x** (according to [LM335 - 4x User Manual](#)), the value can be monitored in 1x PDI Builder by using the variables ADC1 to ADC5.

Note: For pin *ANALOG_1* the correspondent ADC variable in 1x PDI Builder is *ADC1*.

Read the following steps to configure a **Veronte Autopilot 1x**:

1. Go to Connections menu → **ADC 2 section** (This is only an example, the user must select the ADC pin where the signal is connected).

Click on 'Create new program':

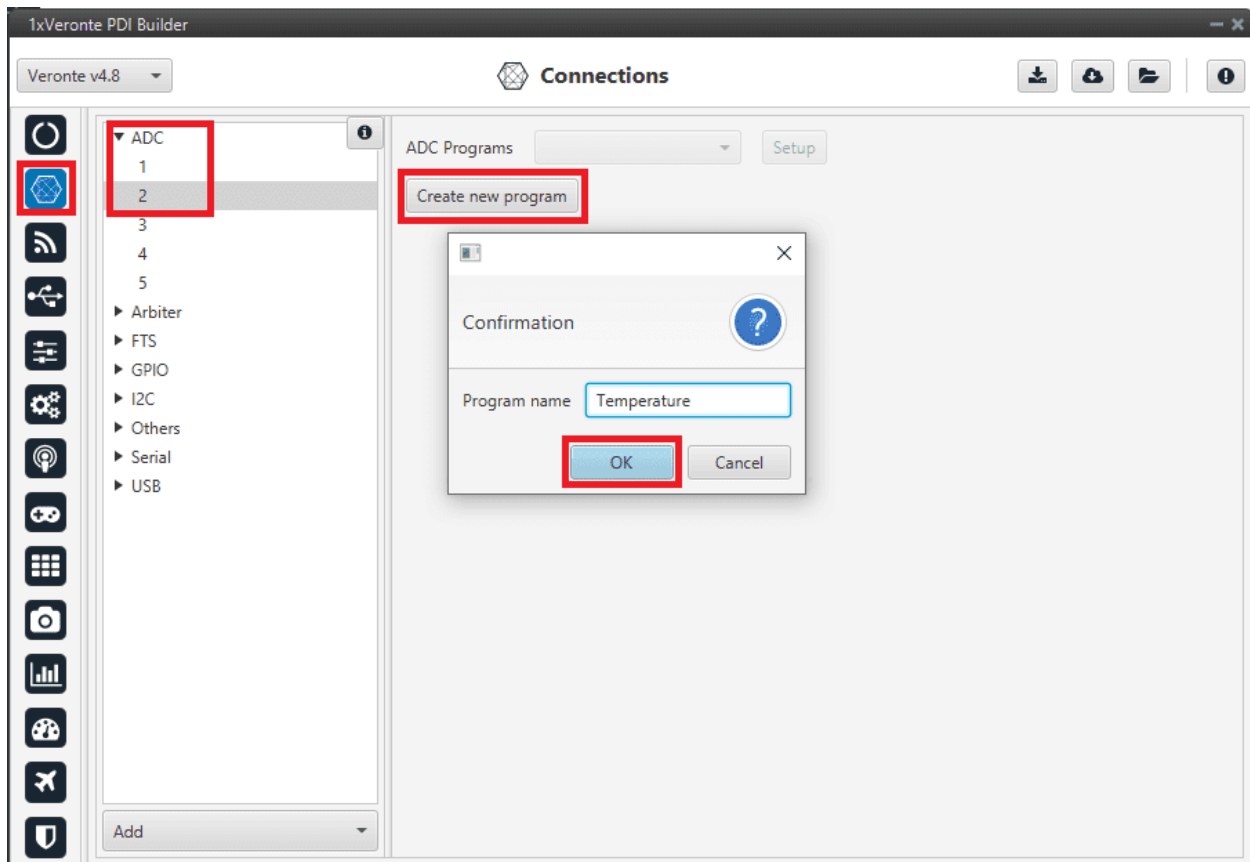
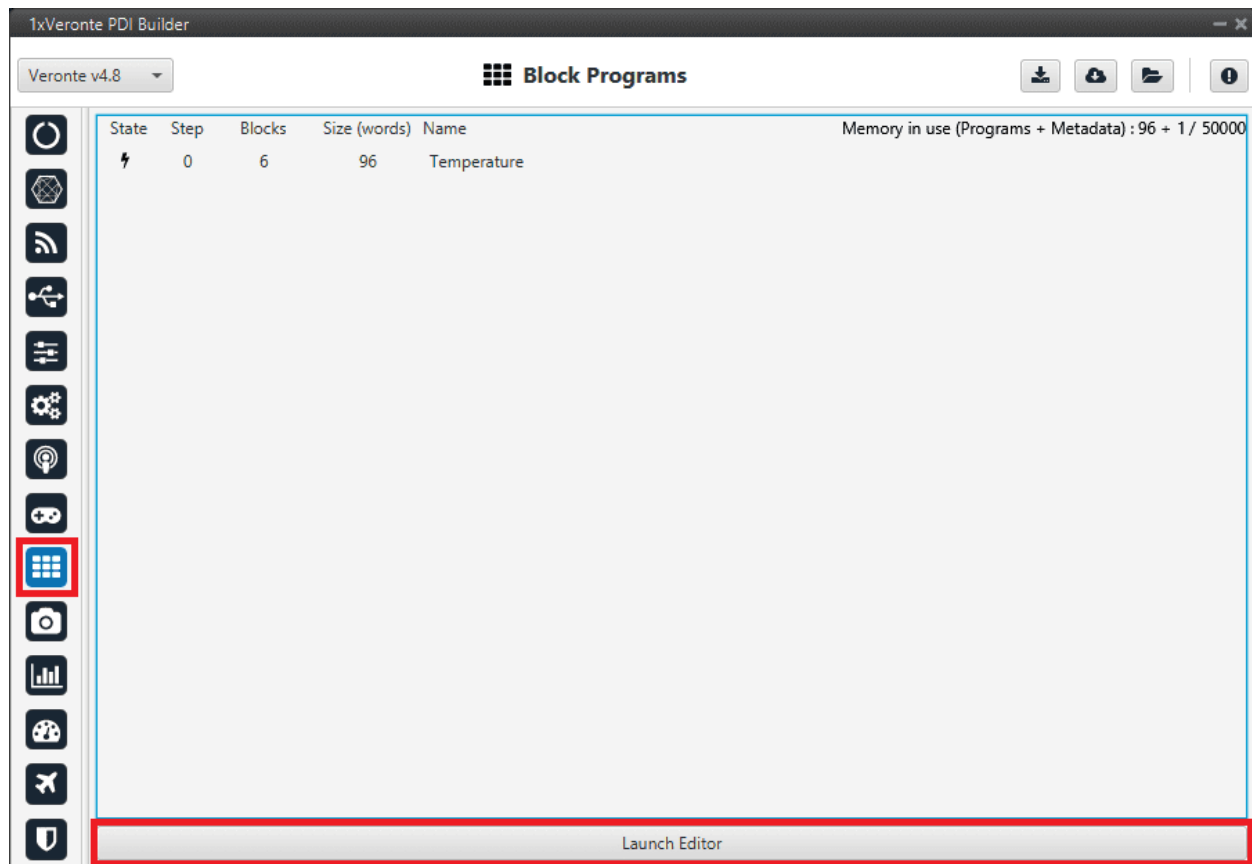


Fig. 37: Create ADC program

2. Go to **Block Programs** menu → **Launch Editor**.



Configure the following operation (for more information about blocks, read *Block Programs*):

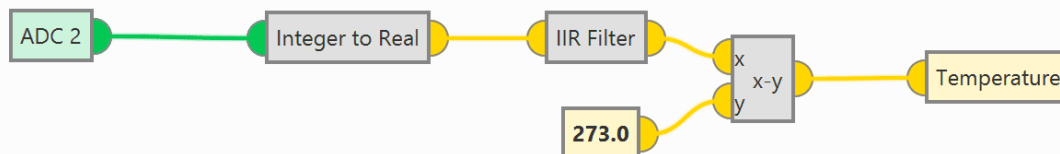


Fig. 38: LM335 sensor operation

Note:

- The **Temperature** variable is an **User Variable** which has been renamed.
- The equation to obtain temperature (in °C) from voltage is: $T = V_{out}100 - 273$. Nonetheless, in the blocks program, the input signal is not multiplied by 100, since **ADC** expresses the voltage in hundredths.

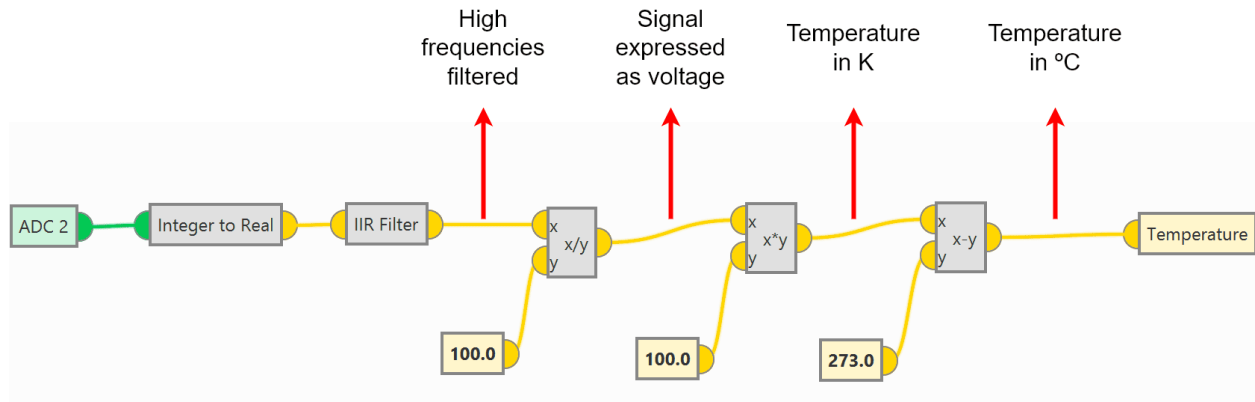


Fig. 39: LM335 sensor operation detailed

- 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.

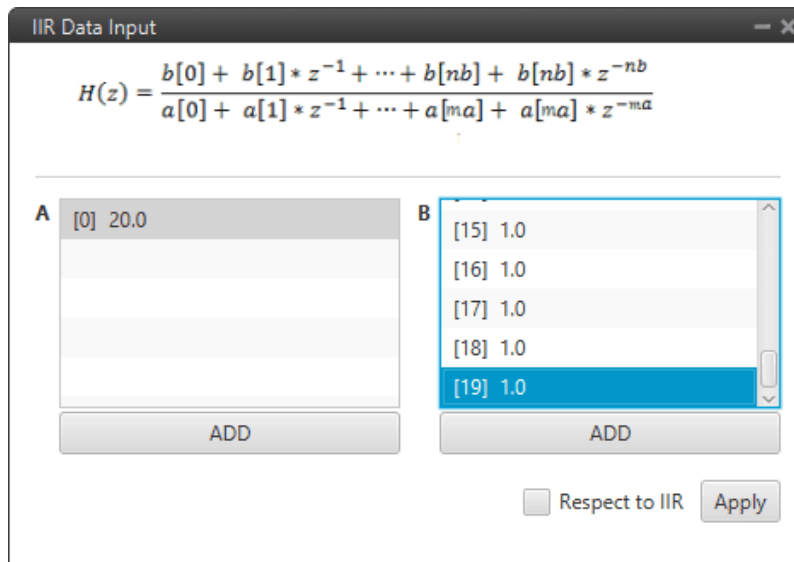


Fig. 40: IIR Filter block configuration

- Save the configuration 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. 41: Temperature value in Veronte Ops

3.5.2.2 Magnetometer Honeywell HMR2300

3.5.2.2.1 RS-232

Magnetometer Honeywell HMR2300 can be connected via **RS-232** (serial interface) in accordance with the manufacturer's specifications and following the [Hardware installation - Electrical](#) section of the **1x user manual**.

The following steps explain how to configure Veronte Autopilot 1x to integrate this external magnetometer:

1. Go to **Connections menu** → Serial section → **232 tab**.

Configure the serial communication settings:

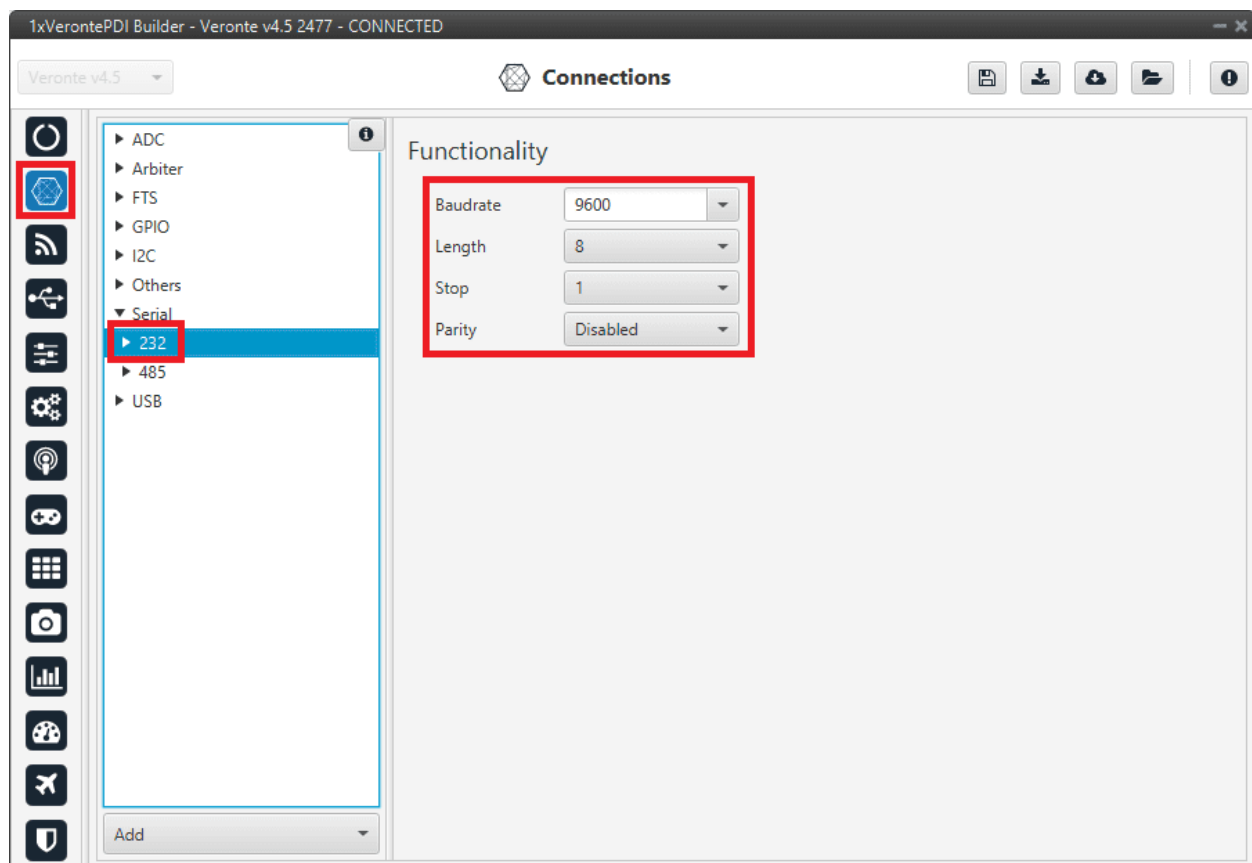


Fig. 42: RS-232 connection configuration

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**.

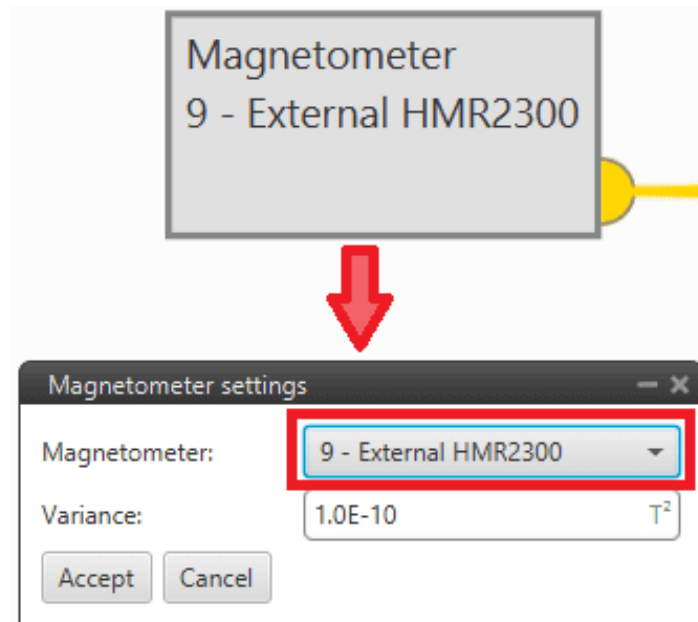


Fig. 43: Magnetometer sensor block configuration

For more information on this block, see *Magnetometer sensor* of **Block Programs** section.

3. Go to Sensors menu → Magnetometer section → **External HMR2300** tab.

Set the rotation matrix according to the sensor installation by clicking on *Edit Rotation Matrix*:

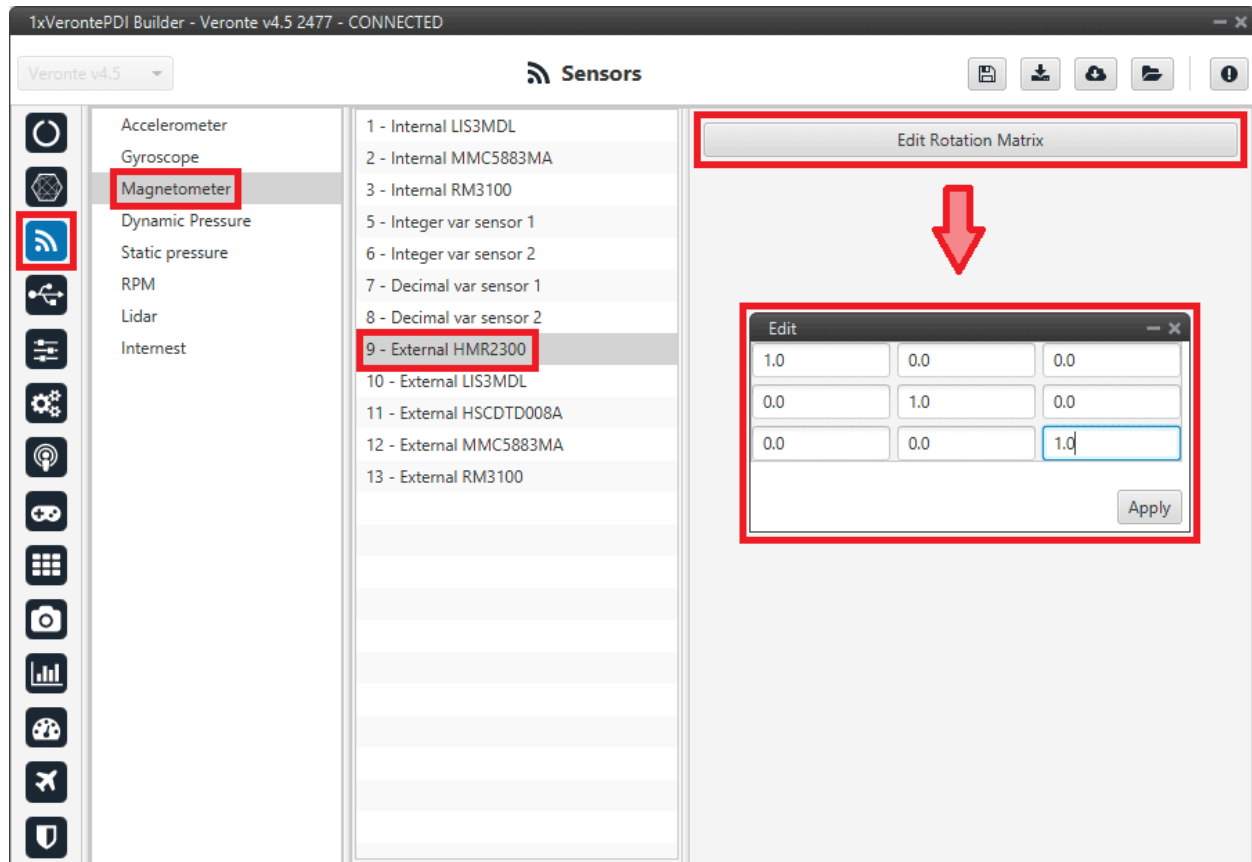


Fig. 44: External HMR2300 magnetometer - Rotation matrix

- Go to Input/Output menu → **I/O Setup** section.

Bidirectionally connect the **RS232 Producer** to the **External HMR2300 magnetometer Consumer**:

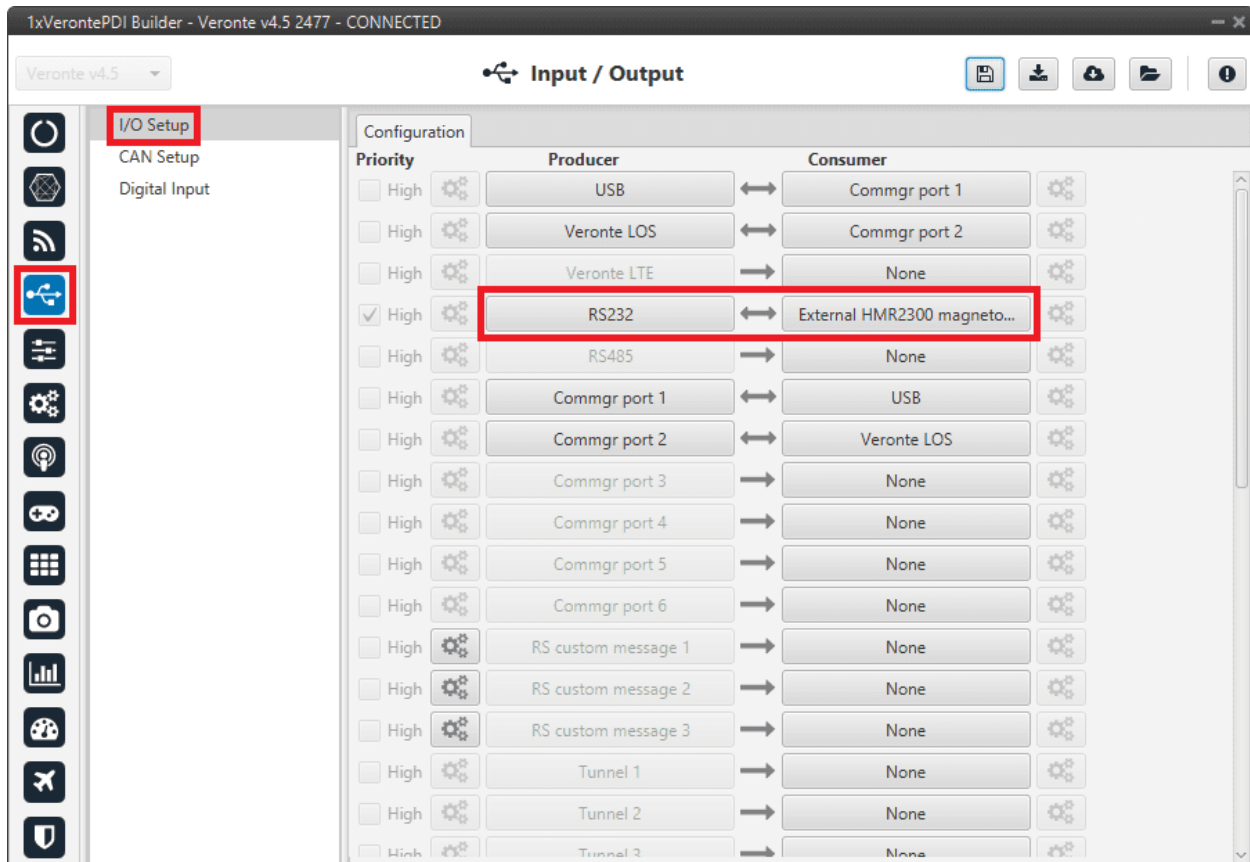


Fig. 45: RS232 ↔ External HMR2300 magnetometer

Then, the **External HMR2300 magnetometer Producer** should be automatically connected to the **RS232 Consumer**:

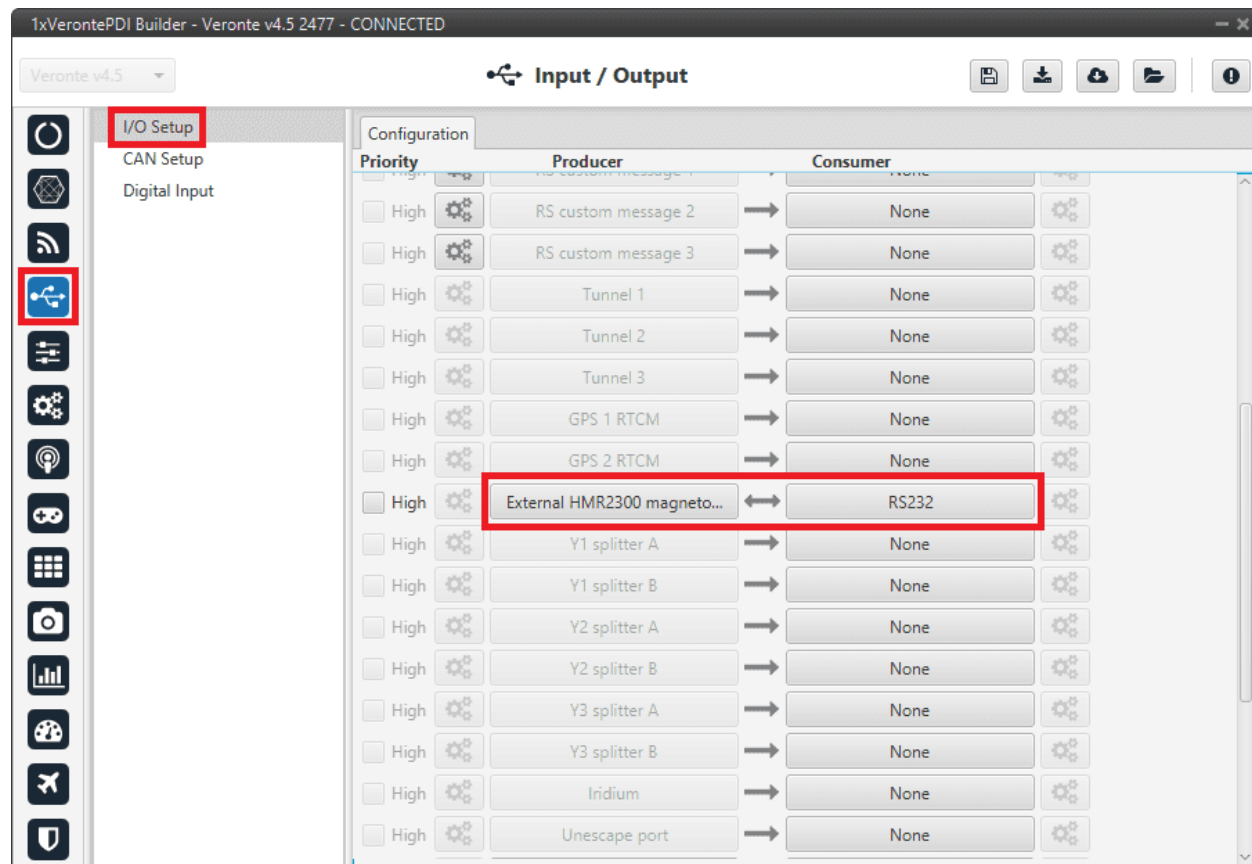


Fig. 46: External HMR2300 magnetometer ↔ RS232

For more information on the Magnetometer Honeywell HMR2300, check out the datasheet: [Smart Digital Magnetometer HMR2300](#).

3.5.2.2.2 RS-485

Magnetometer Honeywell HMR2300 can be connected via **RS-485** (serial interface) in accordance with the manufacturer's specifications and following the [Hardware installation - Electrical](#) section of the **1x user manual**.

Follow the next steps to establish a correct 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 converter).
2. The configuration to establish communication must be:
 - In **binary** mode
 - With **continuous forwarding**
 - **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 Honeywell HMR2300 magnetometer.

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 connection, the **485 connection 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](#).

3.5.2.3 MEX as Magnetometer Honeywell HMR2300

MEX can be used as an external magnetometer Honeywell HMR2300 connected to Veronte Autopilot 1x via serial interface, **RS-232** or **RS-485**. For this serial connection, check the [Hardware installation - Electrical](#) section of the **1x user manual**.

The following steps explain how to configure Veronte Autopilot 1x to integrate it as an external magnetometer:

1. Go to **Connections menu** → Serial section → **232/485 tab**.

Configure the serial communication settings:

Note: If the user connects the MEX via RS-485, configure the RS-485 port instead of RS-232.

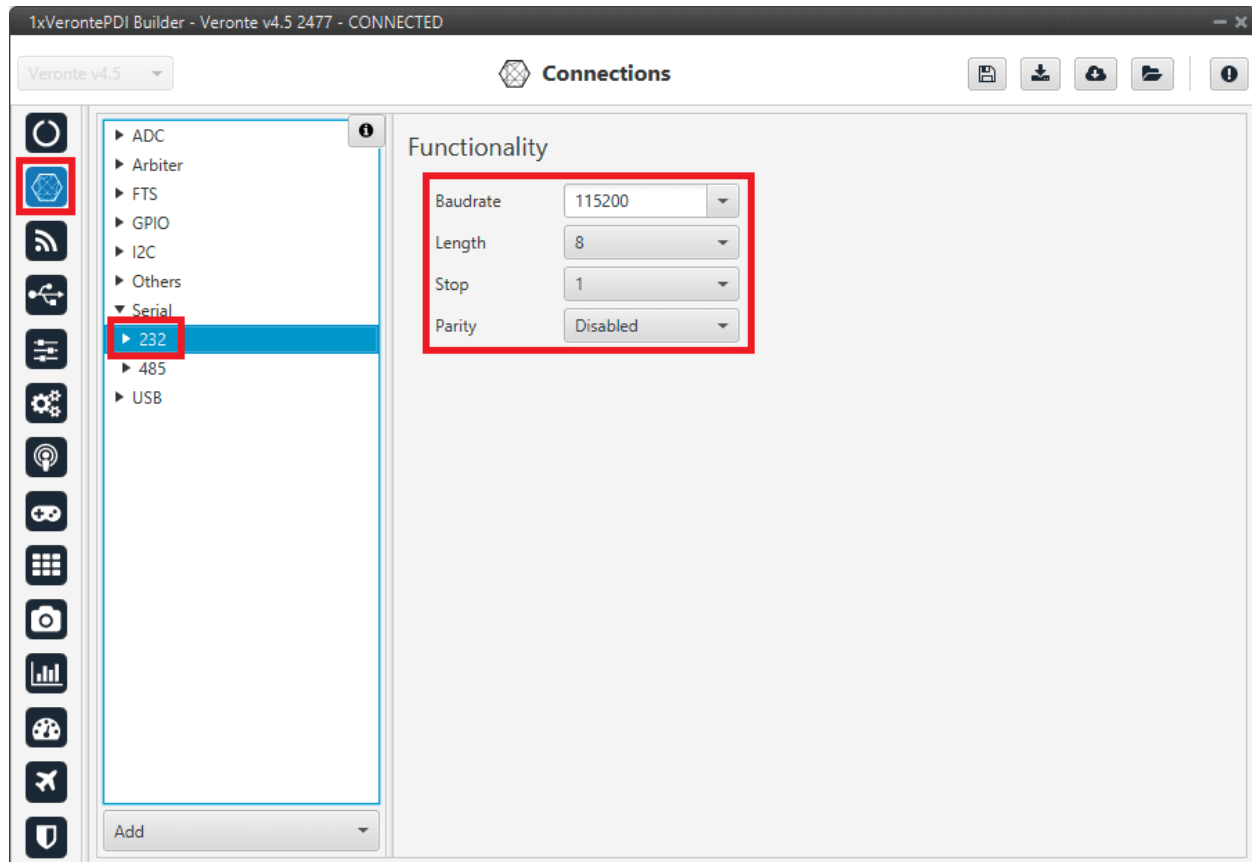


Fig. 47: RS-232 connection configuration

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**.

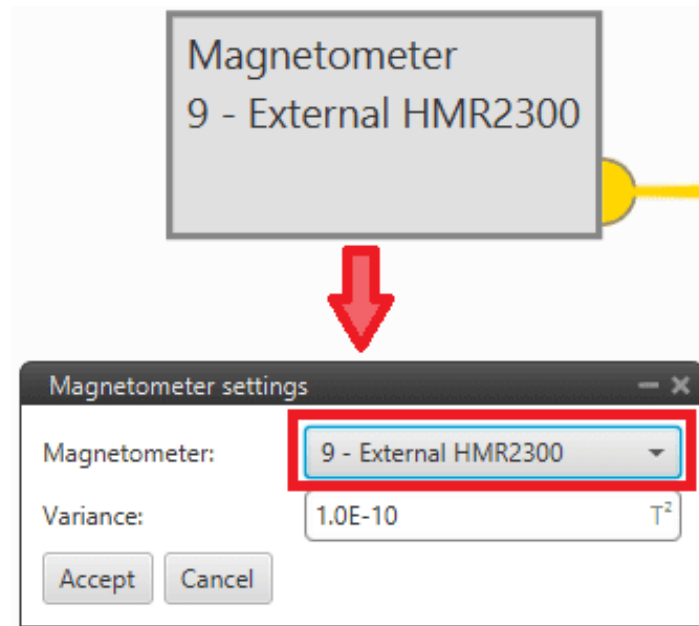


Fig. 48: Magnetometer sensor block configuration

For more information on this block, see *Magnetometer sensor* of **Block Programs** section.

3. Go to Sensors menu → Magnetometer section → **External HMR2300 tab**.

Set the rotation matrix according to the sensor installation by clicking on *Edit Rotation Matrix*:

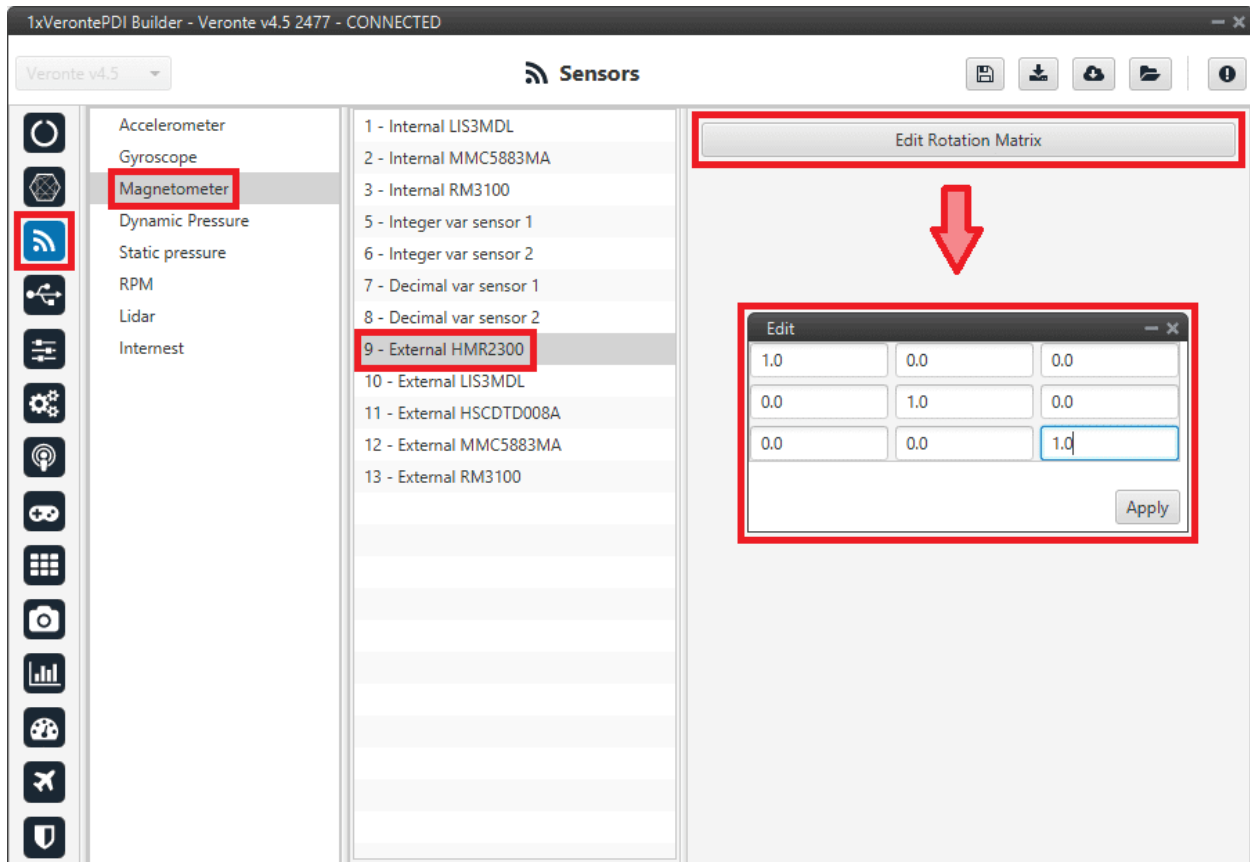


Fig. 49: External HMR2300 magnetometer - Rotation matrix

- Go to Input/Output menu → **I/O Setup section.**

Bidirectionally 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.

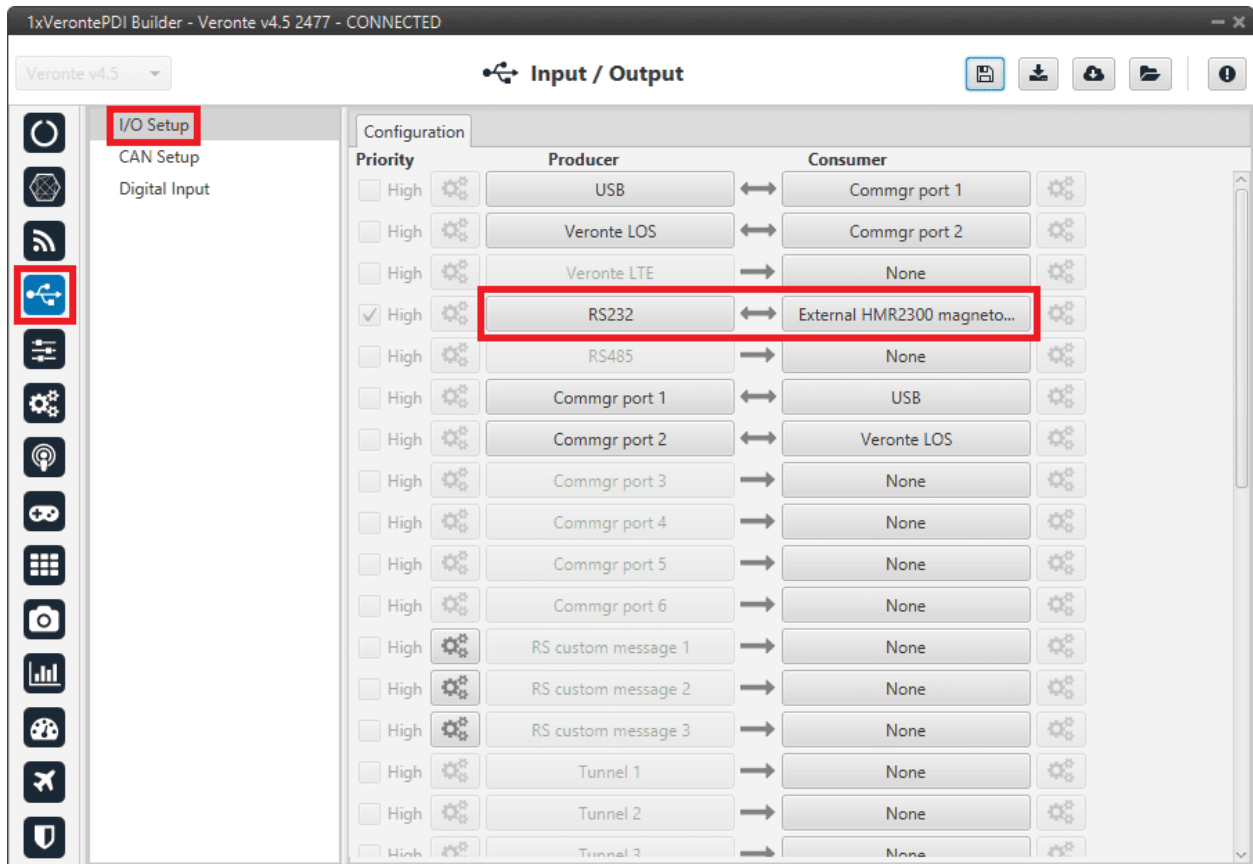


Fig. 50: RS232 ↔ External HMR2300 magnetometer

Then, the **External HMR2300 magnetometer Producer** should be automatically connected to the **RS232 Consumer**:

Note: If the user connects the MEX via RS-485, the RS485 Consumer shall be connected instead of the RS232 Consumer.

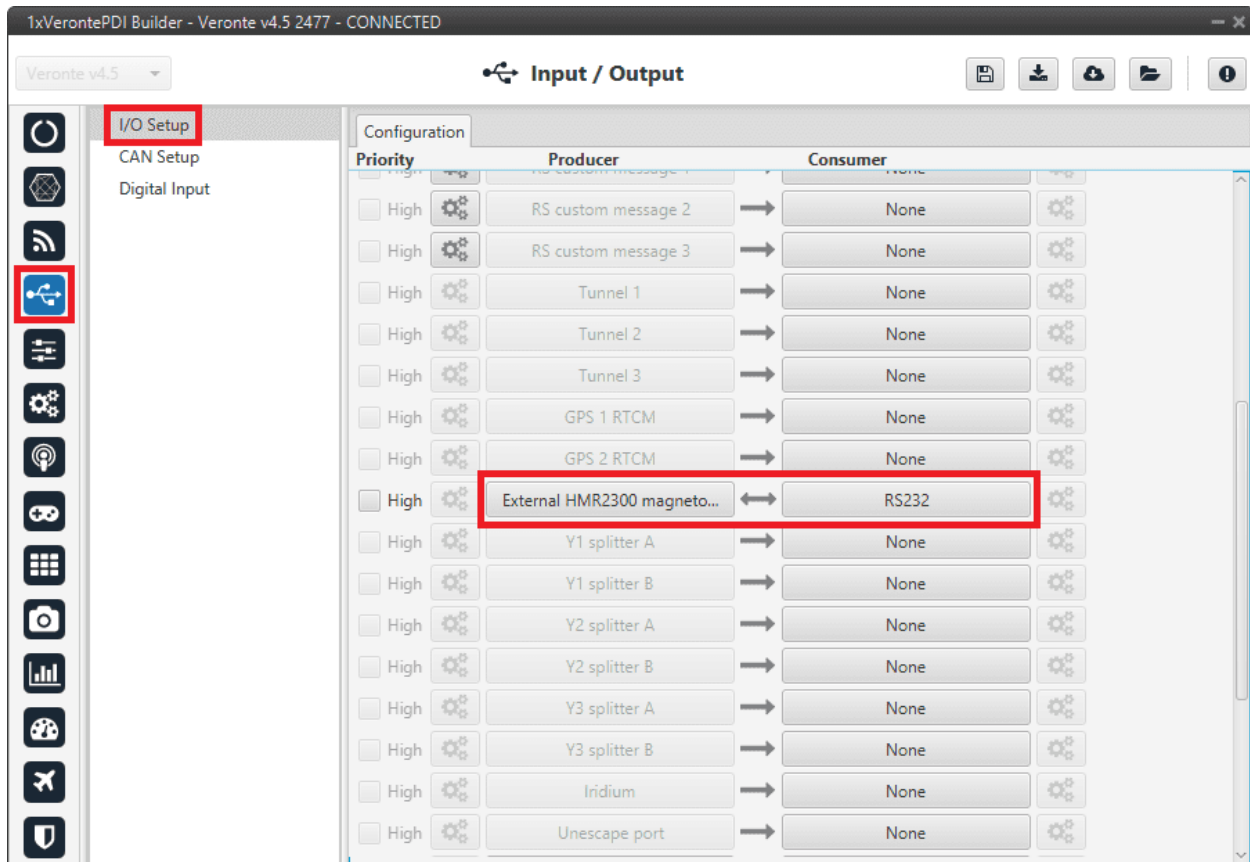


Fig. 51: External HMR2300 magnetometer ↔ RS232

3.5.2.4 OAT Sensor

Once OAT sensor is connected to 1x autopilot, the value can be monitored in 1x PDI Builder by using the variables ADC1 to ADC5.

Note: For pin *ANALOG_1* the correspondent ADC variable in 1x PDI Builder is *ADC1*.

Read the following steps to configure it:

1. Go to Connections menu → **ADC 1 section**.

Click on 'Create new program':

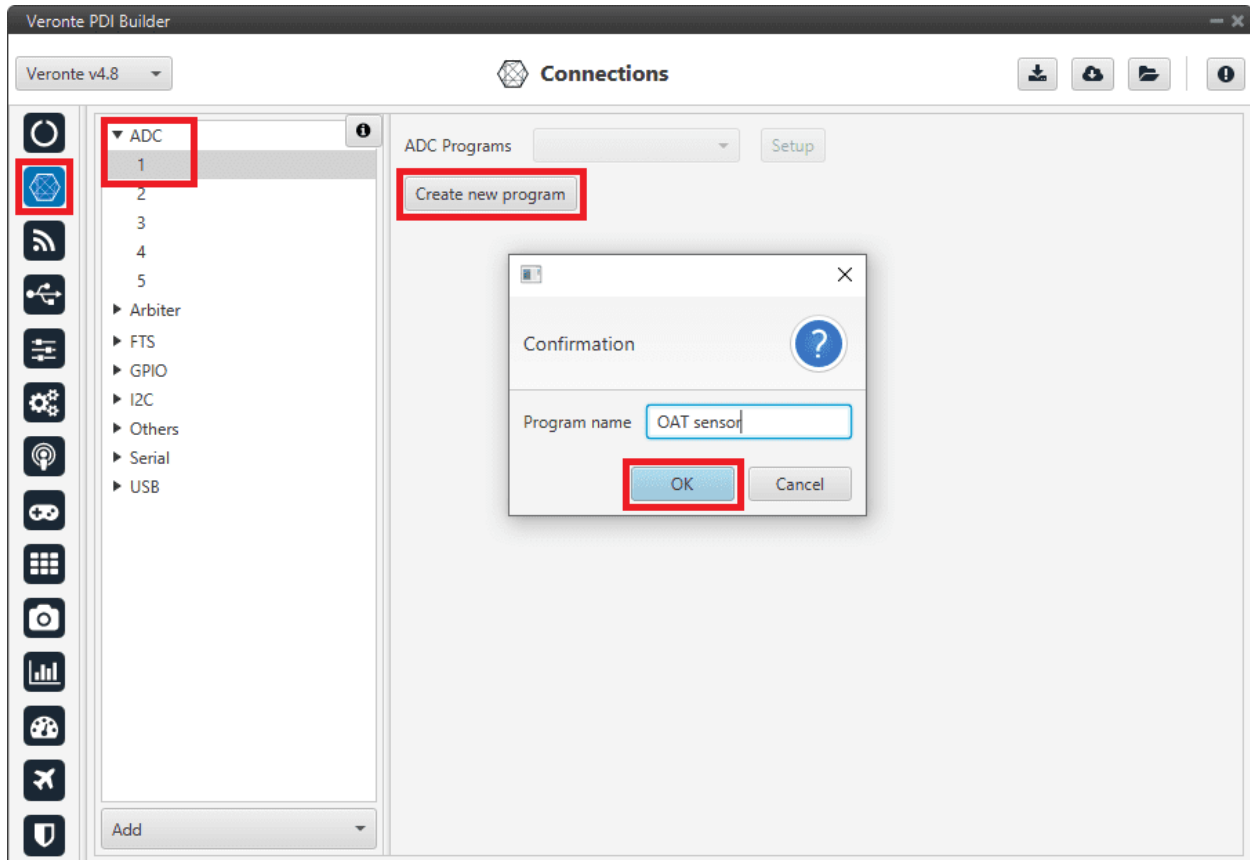


Fig. 52: Create ADC program

2. Go to **Block programs** menu.

Configure the following operation (for more information about blocks, see [Block Programs](#)):

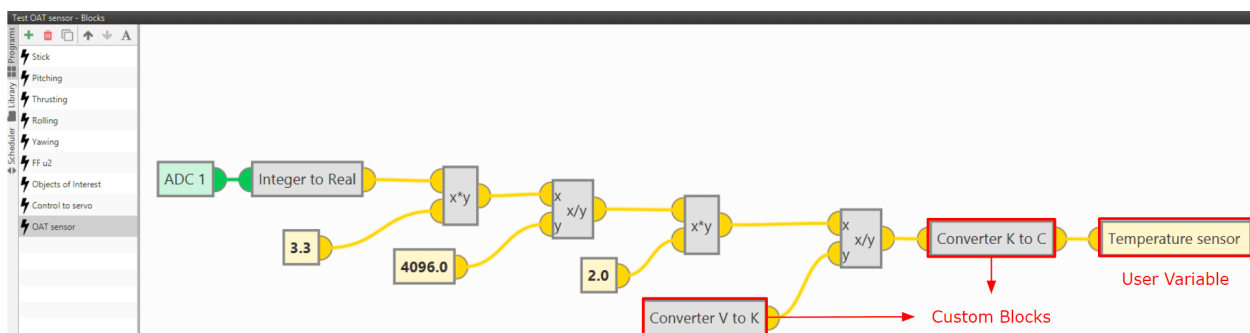


Fig. 53: OAT sensor operation

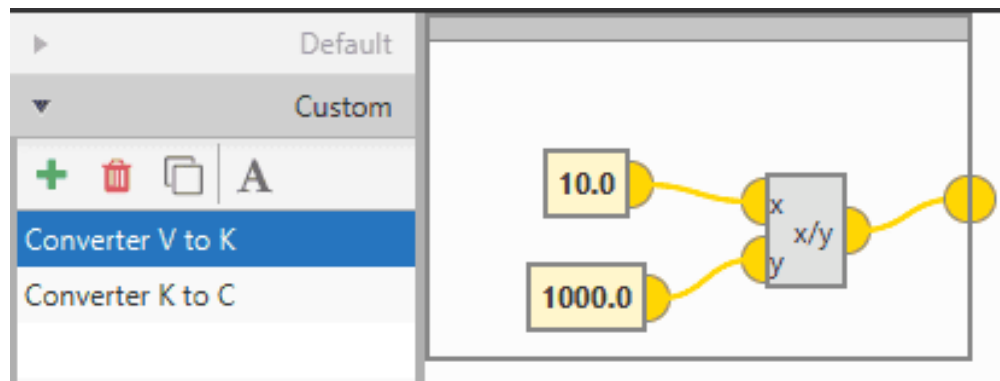


Fig. 54: Custom block: Converter from V to K

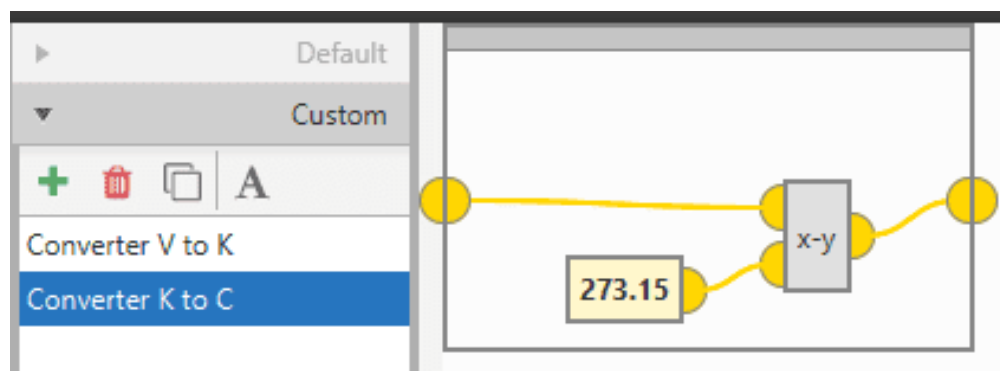


Fig. 55: Custom block: Converter from K to °C

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 4x autopilot can read up to 36 V per each ADC, the 3.3 value of the ADC program must be changed to 36 if applicable.

3.5.2.5 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 **Connections menu** → Serial section → **232 tab**.

Configure the serial communication settings:

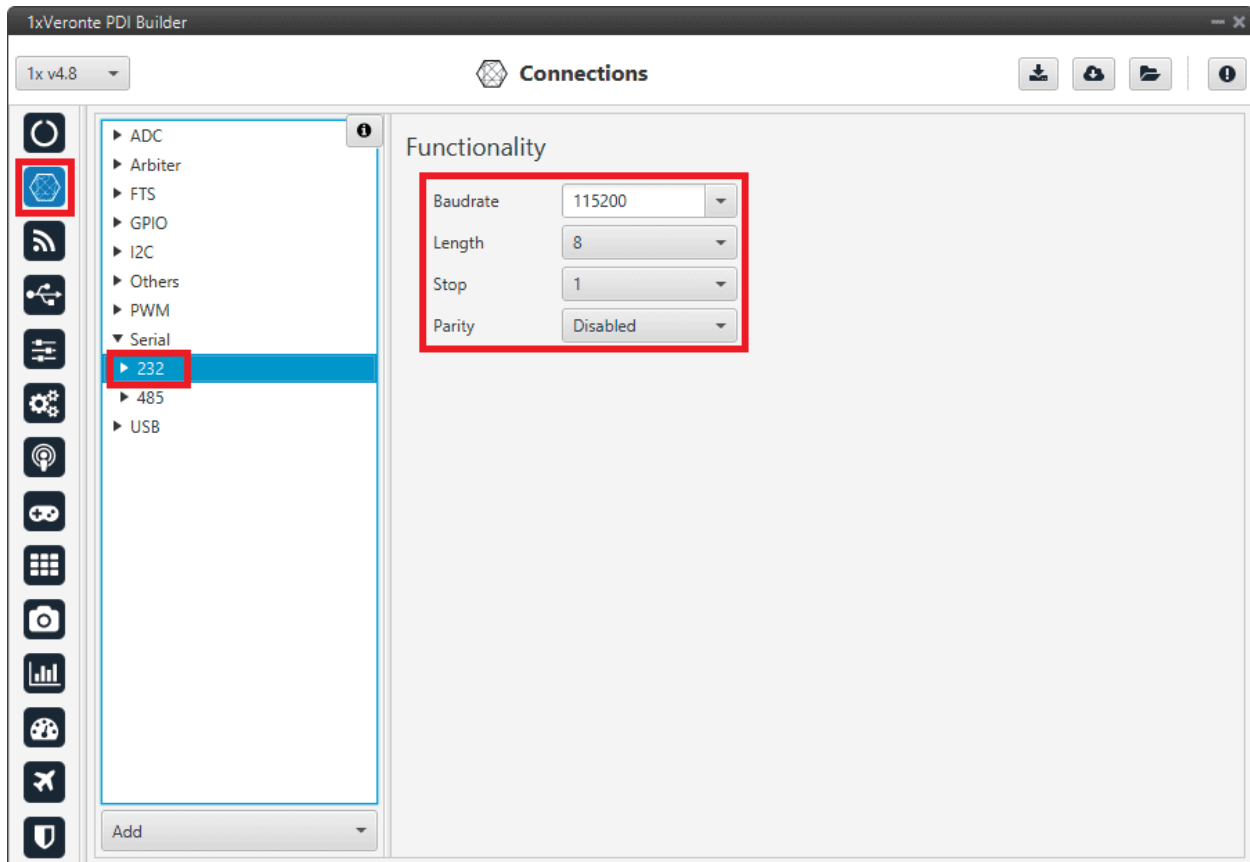


Fig. 56: RS-232 connection configuration

2. Go to Input/Output menu → **I/O Setup** section.

Connect the **RS232 Producer** to the **Vectornav VN-300 Consumer**:

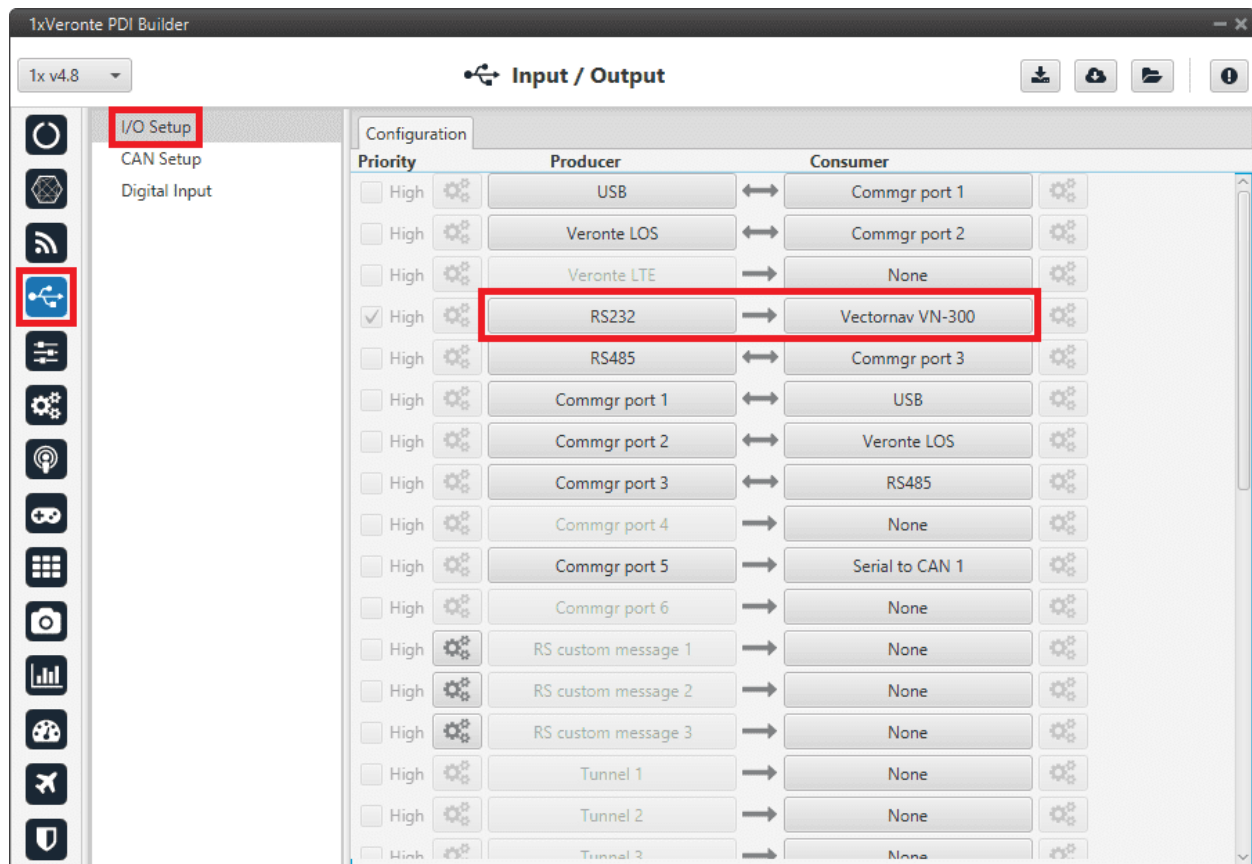


Fig. 57: RS232 → Vectornav VN-300

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 navigation are implemented.

- Configure the *Navigation block* selecting **Vectornav VN-300**.

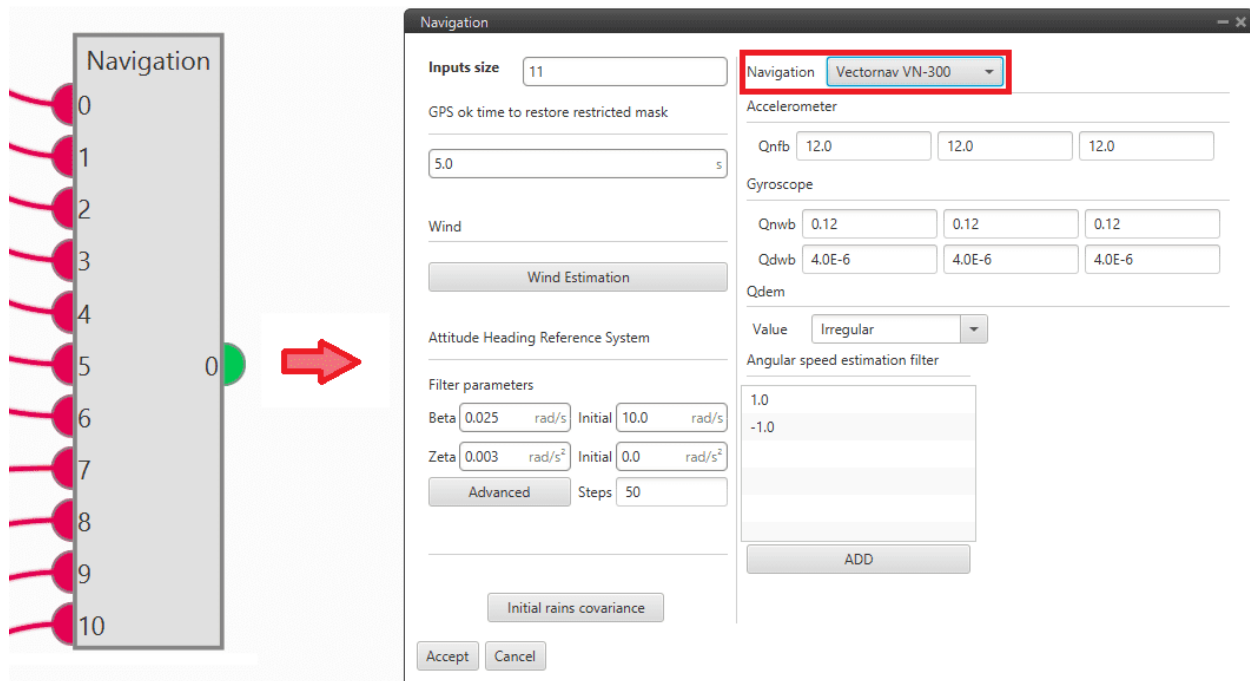


Fig. 58: Navigation sensor block configuration

For more information on this block, see [Navigation](#) of **Block Programs** section.

3.5.2.5.1 Vectornav VN-300 configuration

It is also required a configuration on the **Vectornav 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 1x Autopilot 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**.

3.5.3 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 1x Autopilot:

- Veronte Autopilot 1x v4.5: To consult its internal radio, click [here](#).
- Veronte Autopilot 1x v4.8: To consult its internal radio, click [here](#).

3.5.3.1 Digi internal radio

3.5.3.1.1 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 the [user manual](#) to use it.

Configuration in 1x PDI Builder

2. Go to Input/Output menu → **I/O Setup section**.

Since the configuration of this menu 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 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 1x Autopilot 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**.

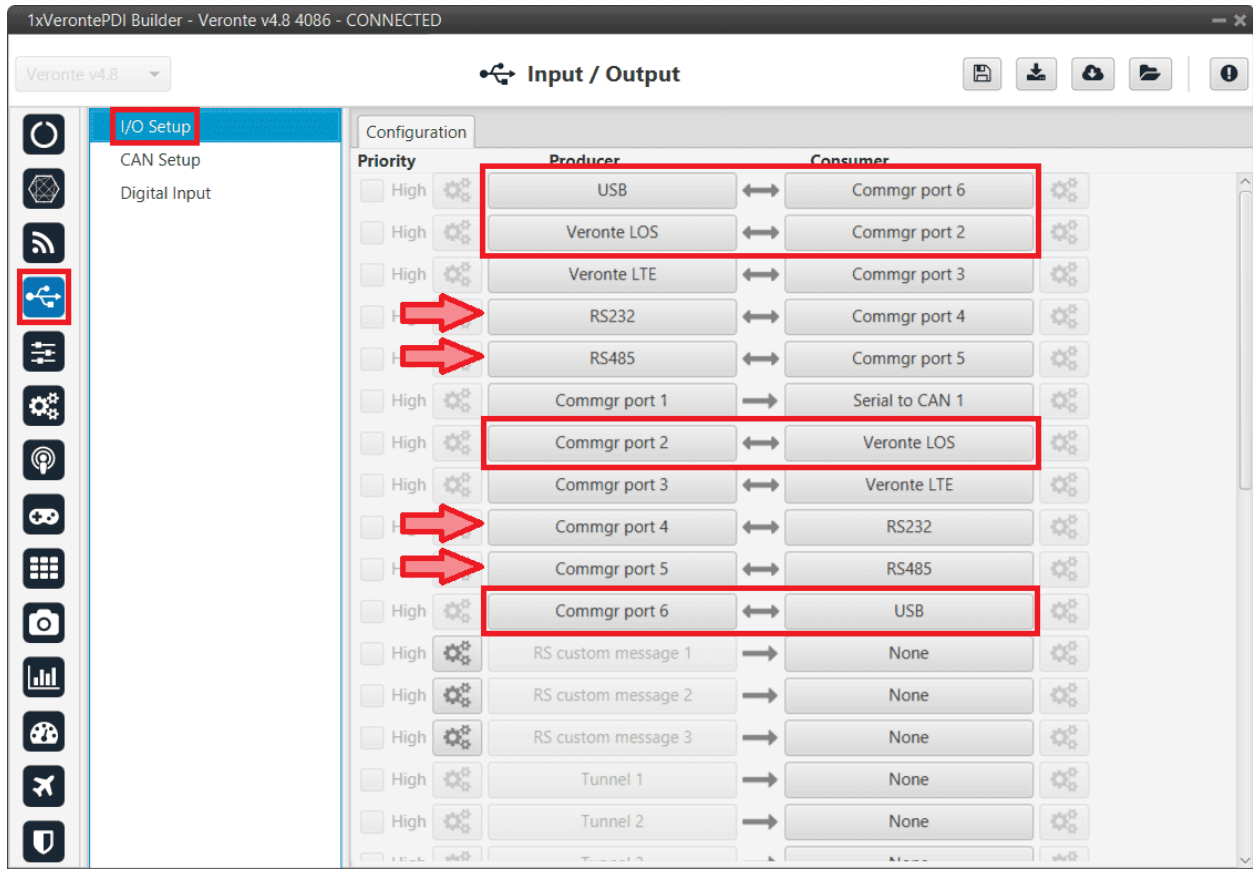


Fig. 59: Example of configuration of USB and Veronte LOS ports

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 .

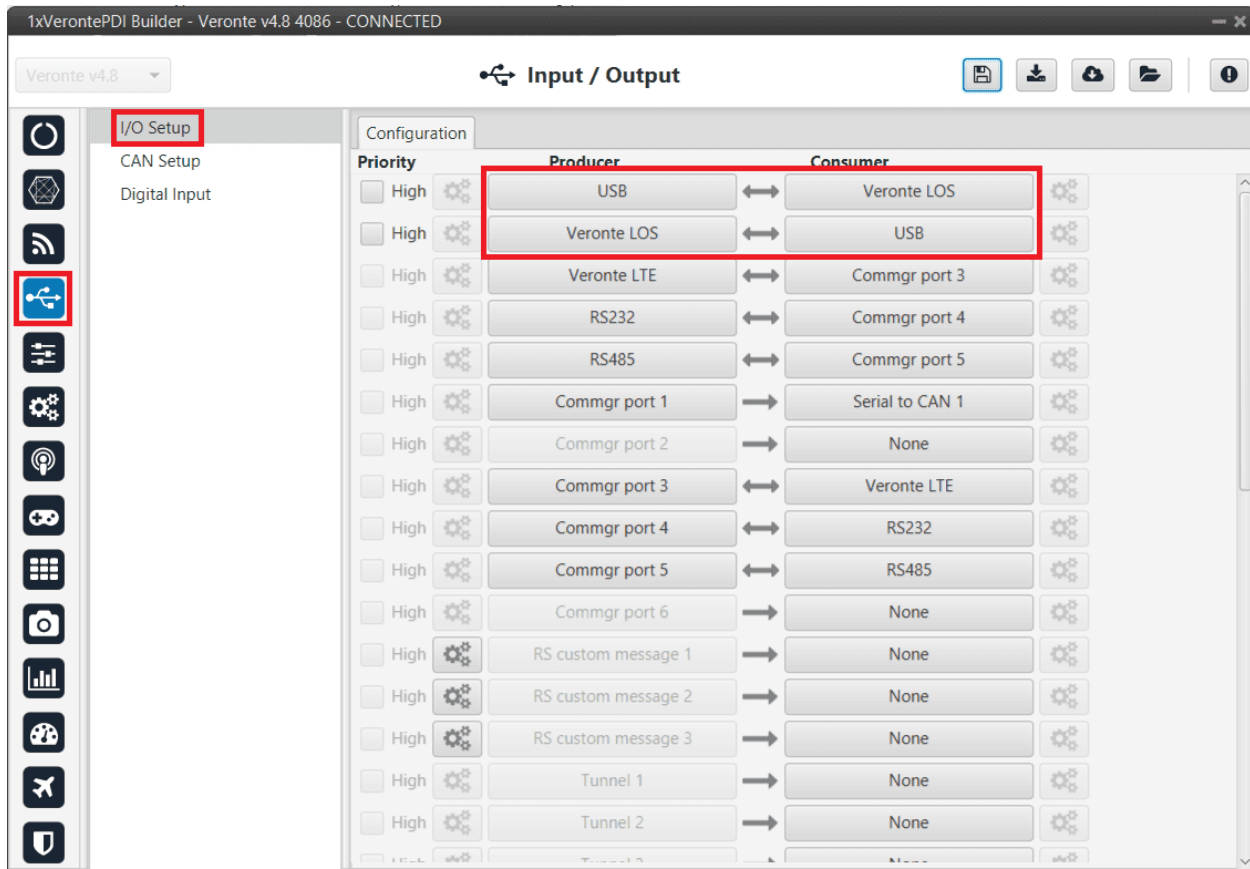


Fig. 60: USB ↔ Veronte LOS

- Go to Communications menu → **Veronte LOS** section.

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.

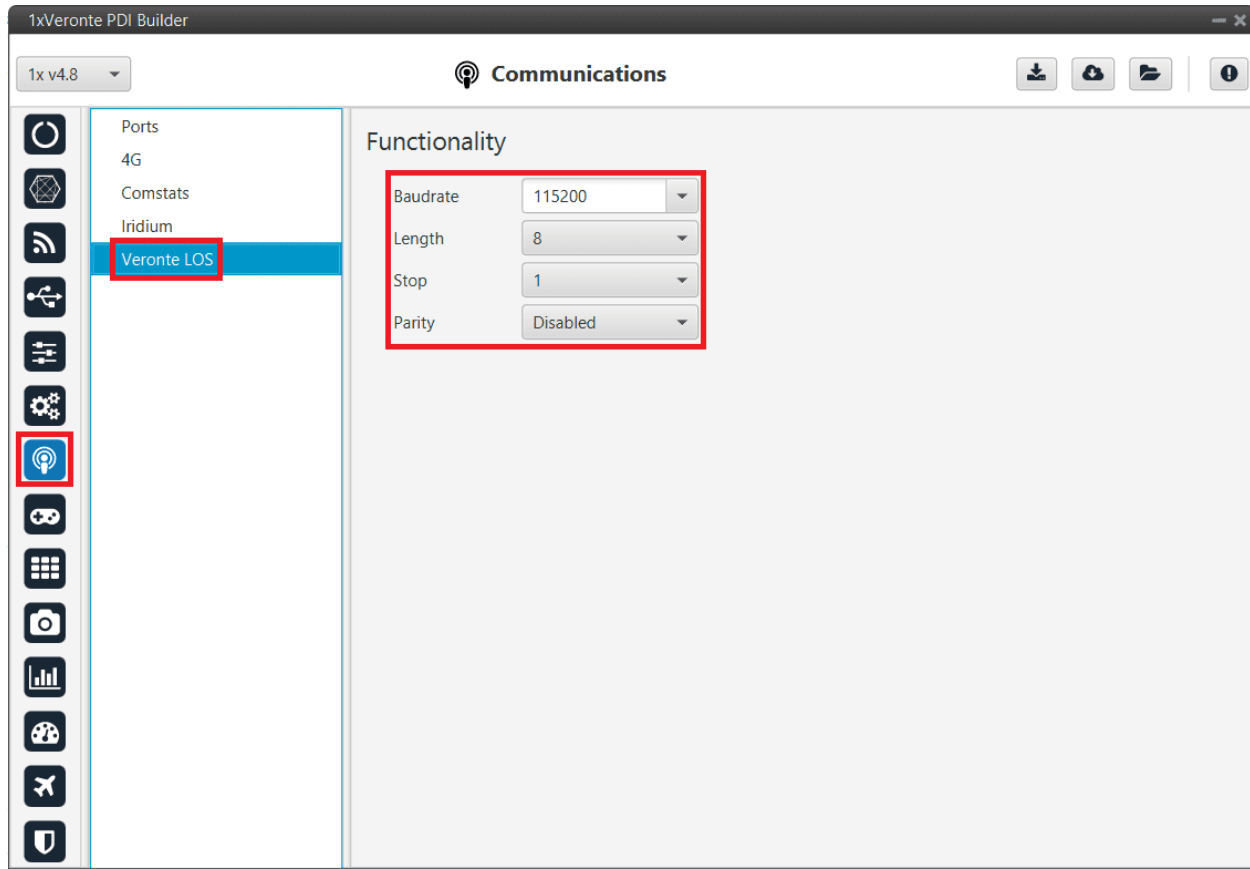



Fig. 61: Veronte LOS baudrate

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:

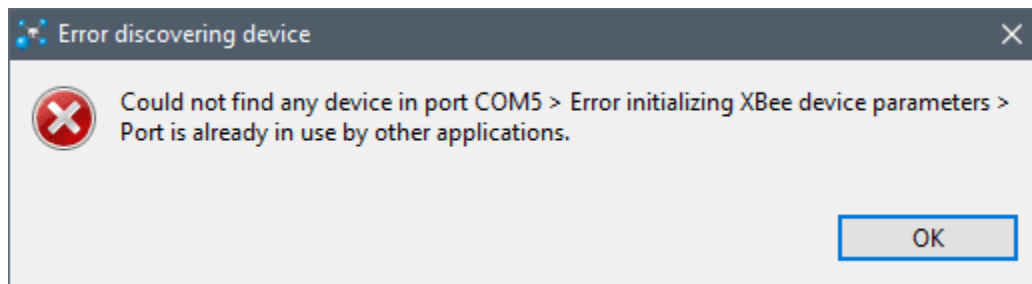


Fig. 62: XCTU error message

Important: Remember that to completely close the application the user must close it from the windows system tray.



Fig. 63: Close Veronte Link

Configuration in Digi radio software

7. Download and install **XCTU** (Digi radio software).
8. Build a configuration for 'air' or 'bc' 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 address 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

▼ Addressing

Source and destination addressing settings

SH Serial Number High	13A200
SL Serial Number Low	41F2B738
MY 16-bit Source Address	FFFF
DH Destination Address High	13A200
DL Destination Address Low	41EF1560
RR XBee Retries	0
TO Transmit Options	0 Bitfield
NP Maximum Packet Payload Length	66

Fig. 64: Air configuration example

▼ Addressing

Source and destination addressing settings

SH Serial Number High	13A200
SL Serial Number Low	41EF1560
MY 16-bit Source Address	FFFF
DH Destination Address High	0
DL Destination Address Low	FFFF
RR XBee Retries	0
TO Transmit Options	0 Bitfield
NP Maximum Packet Payload Length	66

Fig. 65: Ground configuration example

- 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

- Go to Input/Output menu → **I/O Setup section**.

Finally, after configuring the Digi radio in its software, restore the annotated **USB** and **Veronte LOS** configuration (step 2).

If after the whole process described above for setting up the radio or later during operation the **communication between the Digi radio and the Veronte Autopilot 1x is lost**, please check the [Communication lost with intenal Digi radio - > Troubleshooting section](#) of this manual.

3.5.3.1.2 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 [VCP manual](#) -> [Message structure](#).

3.5.3.2 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 the [user manual](#) to use it.

Configuration in 1x PDI Builder

2. Go to Input/Output menu → **I/O Setup section**.

Since the configuration of this menu 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.

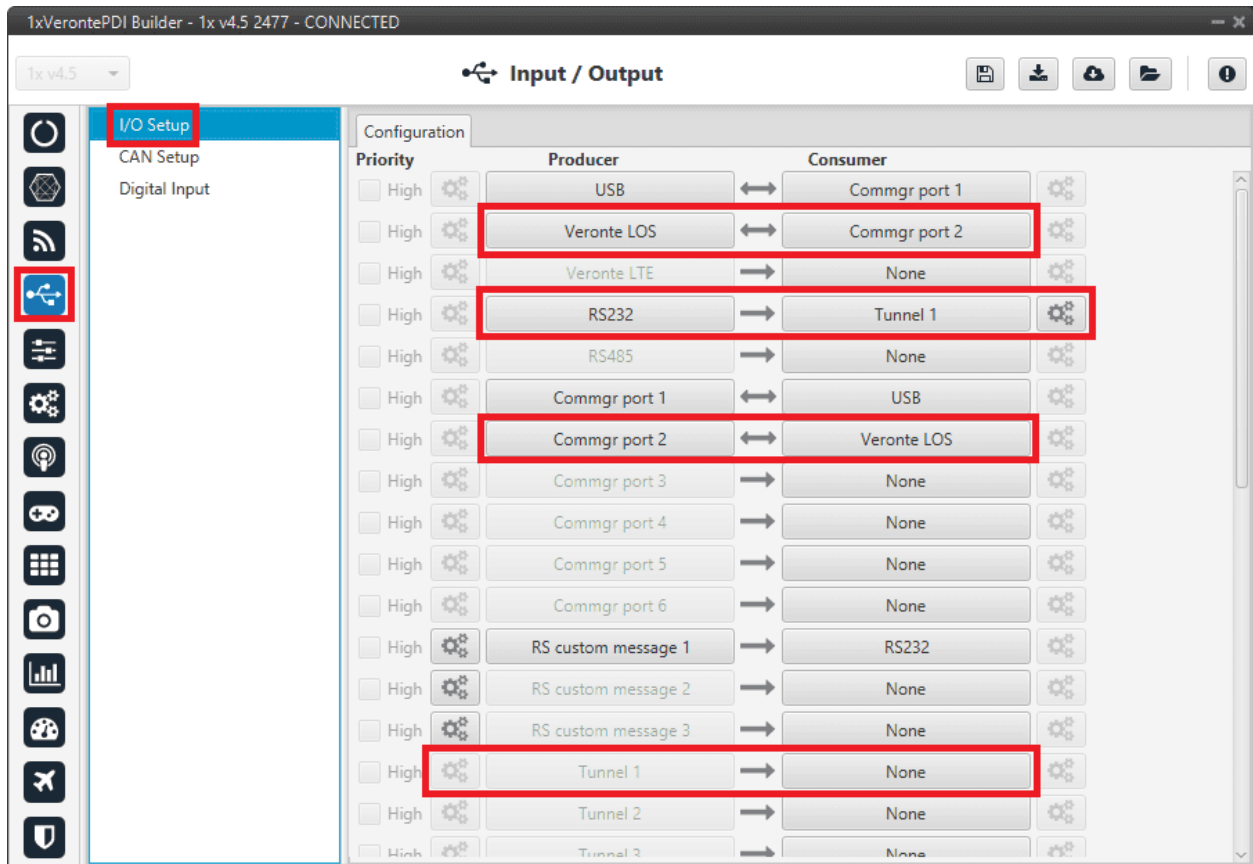


Fig. 66: Example of configuration of Veronte LOS port

3. Change the port which **Veronte LOS producer** is connected to and select a **Tunnel** as **consumer**, in this example *Tunnel 1* has been selected.

Veronte LOS and **Tunnel 1** must have **bidirectional communication** \longleftrightarrow .

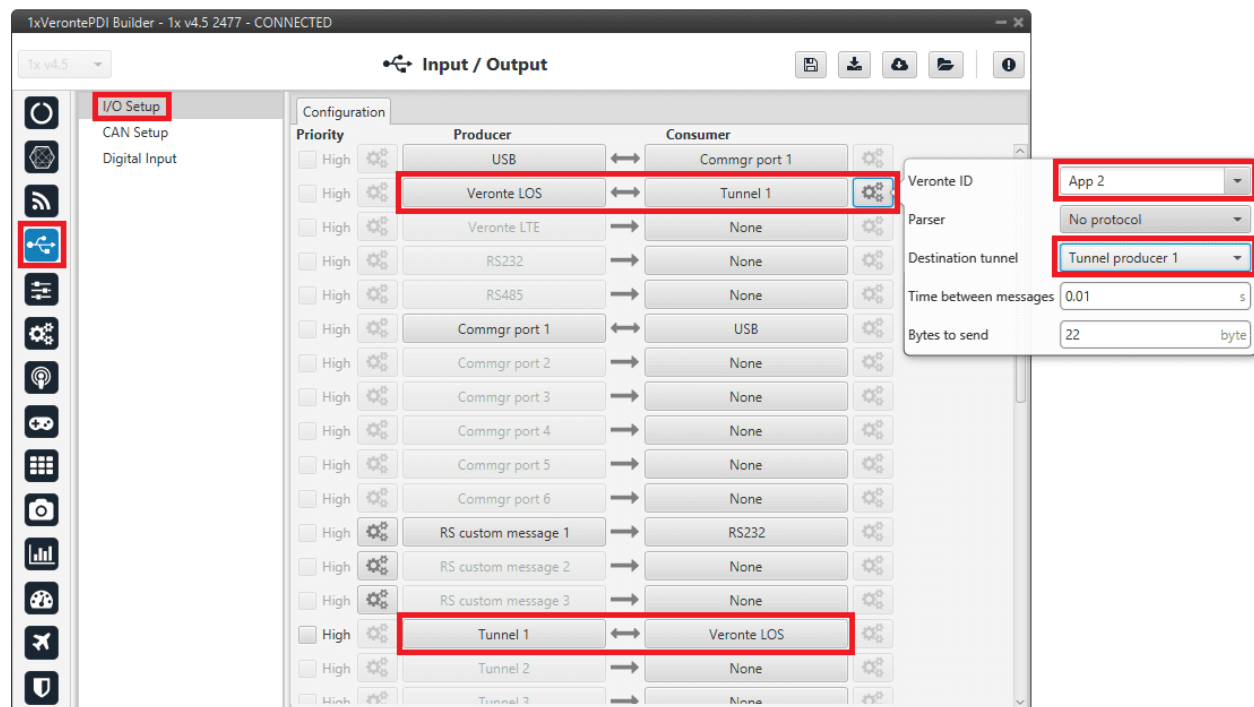



Fig. 67: Veronte LOS ↔ Tunnel 1

4. Click on  to apply the changes to the **Autopilot 1x**.

Configuration in 1x PDI Calibration

5. In **1x PDI Calibration**, go to **Terminal** menu.


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** section.

Finally, after configuring the Microhard radio in **1x PDI Calibration**, restore the annotated **Veronte LOS** and **Tunnel 1** configuration (step 2).

7. Click on  to apply the changes to the **Autopilot 1x**.

3.5.3.3 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 **Connections** menu → Serial section → **232** tab.

Check that these parameters are the same as the parameter values previously set in the external radio.

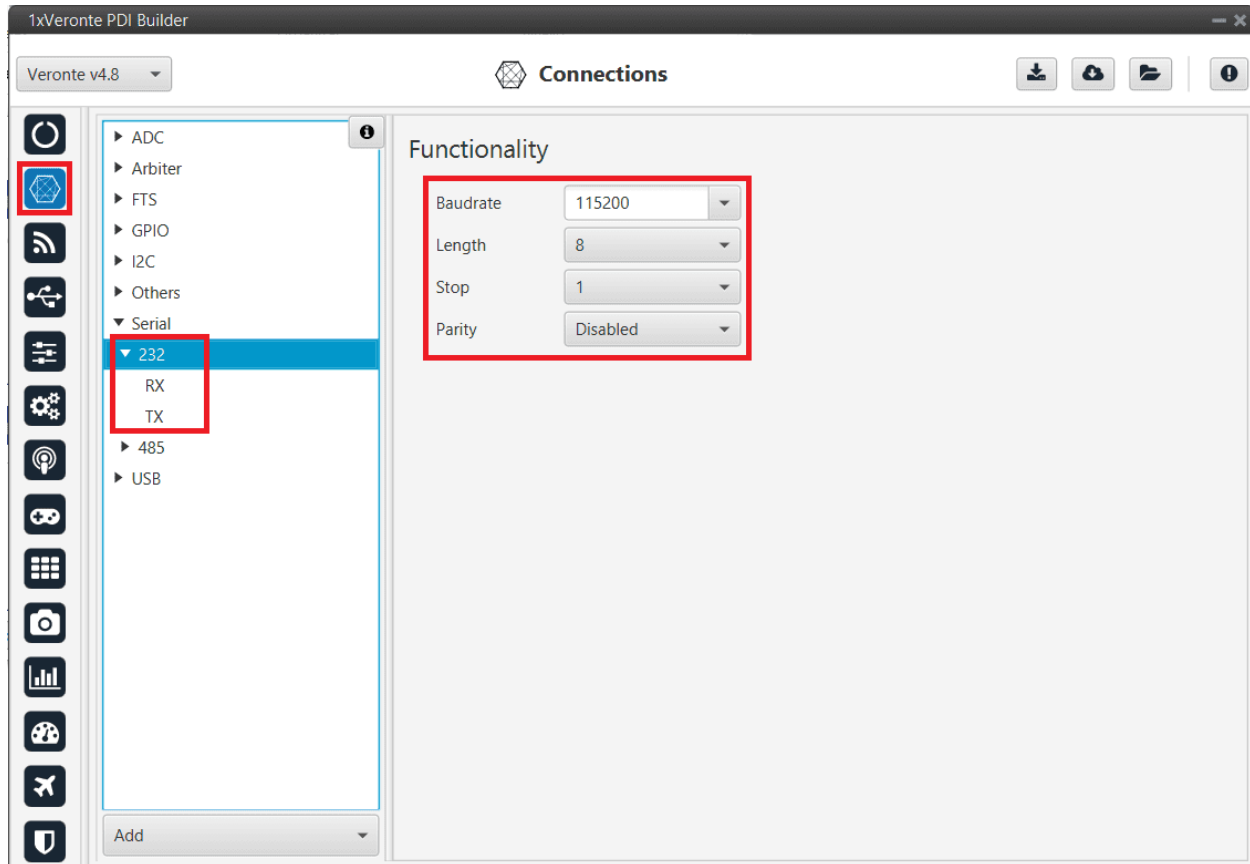


Fig. 68: RS-232 connection configuration

2. Go to Input/Output menu → **I/O Setup** section.

RS-232 has to be configured as a **bidirectional** commgr port.

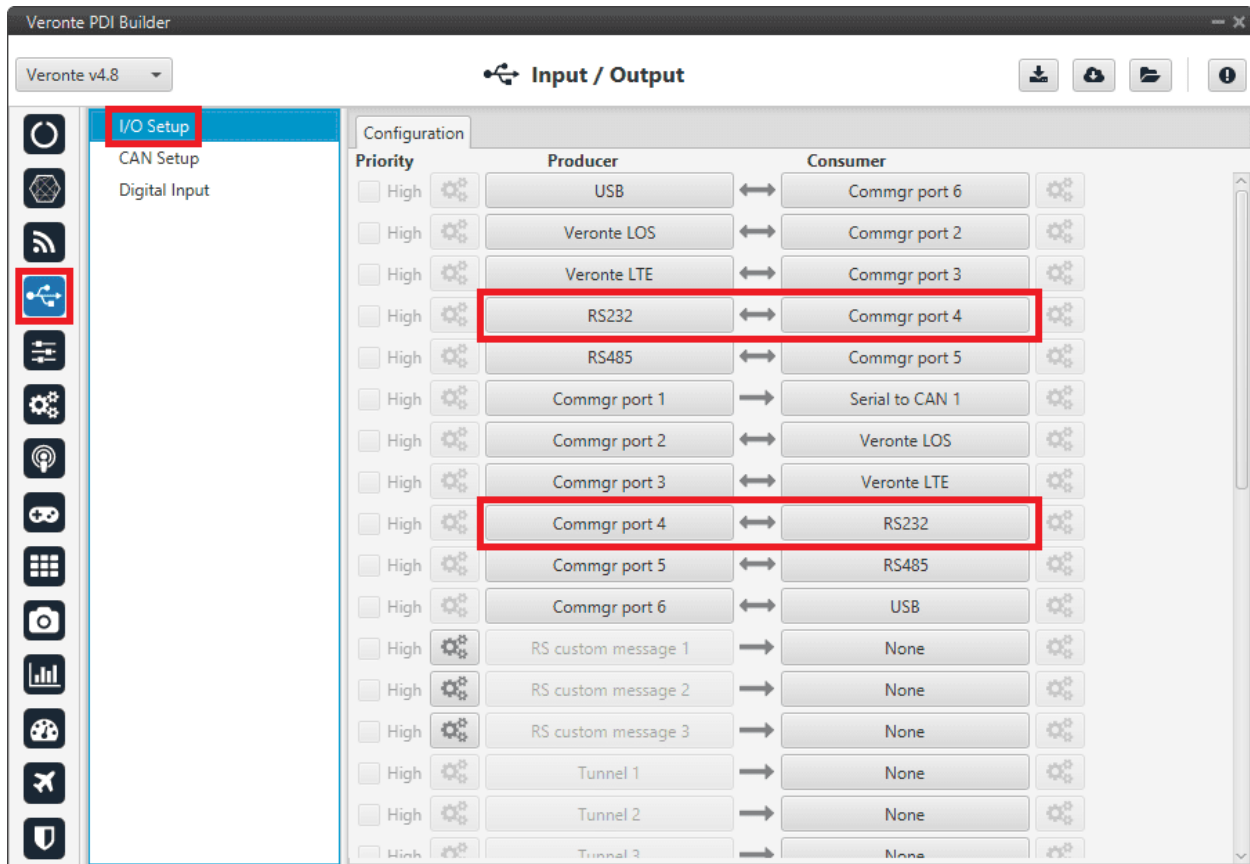


Fig. 69: RS-232 I/O configuration

Note: These settings have to be made in both 1x autopilot units (GND and AIR).

3.5.4 Servos

The user can configure any actuator compatible with the communication interfaces.

3.5.4.1 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 [Hardware installation - Electrical](#) section of the **1x user manual** to connect it to the 1x autopilot.
2. Go to Connections menu → **PWM section**.
 - Select and configure the PWM pins where the servos are connected. Set the frequency according to the manufacturer's specifications.

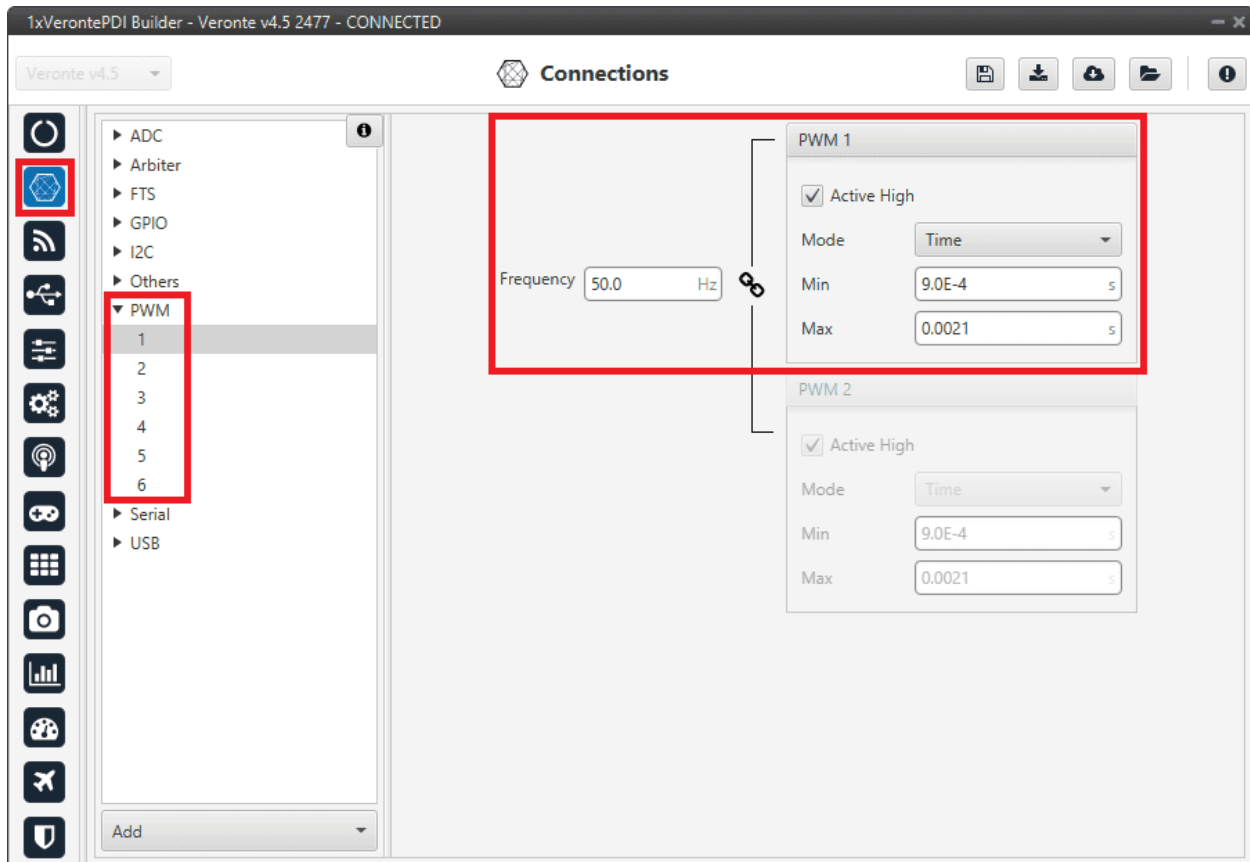


Fig. 70: PWM connection configuration

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 they are configured as GPIO. For this, refer to [PWM 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*:

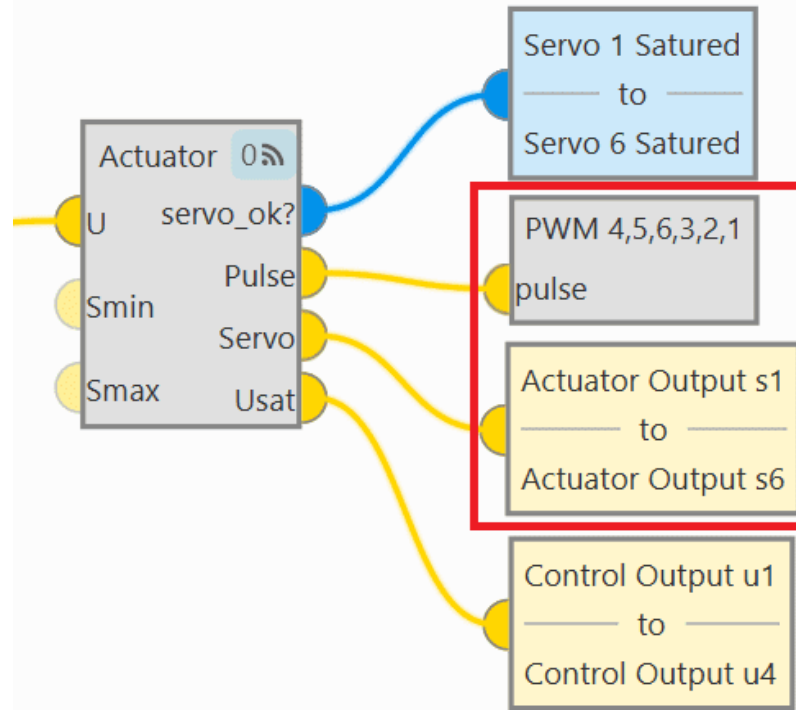


Fig. 71: Block programs connection

- 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 will relate to the first actuator output, which does not necessarily mean that *PWM 1* is assigned to *Actuator Output s1*.

In this example, the PWMs are assigned to the actuator outputs as shown in the following figure:

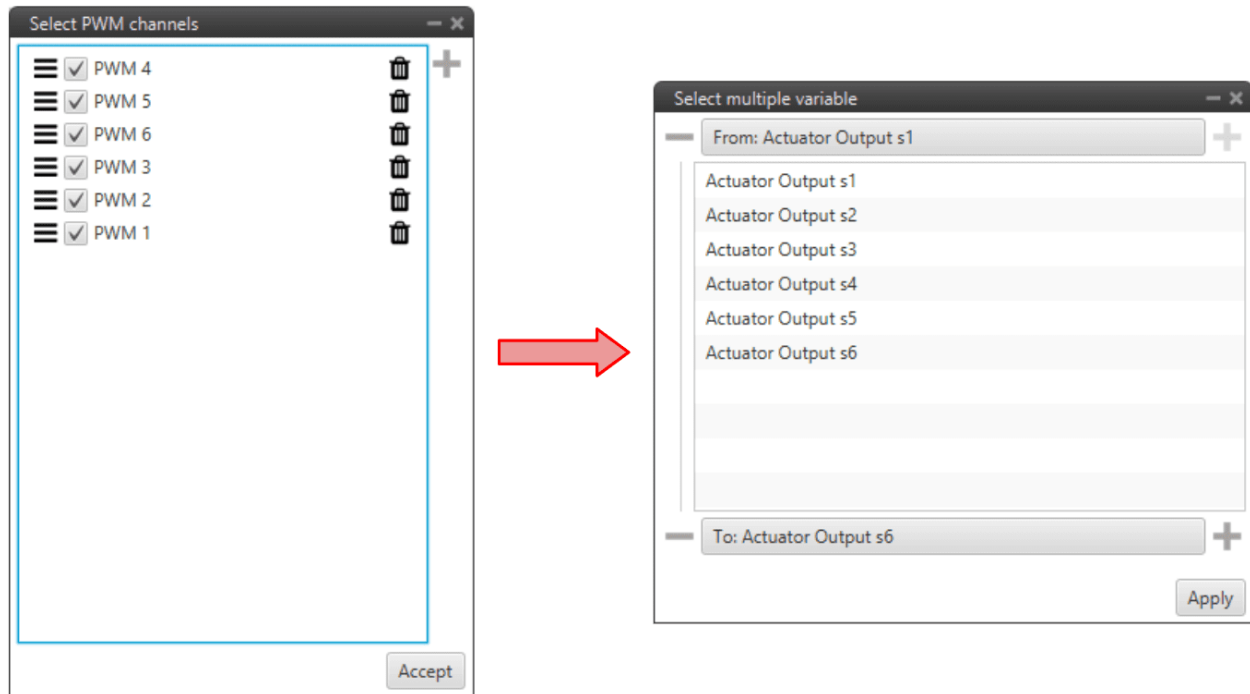


Fig. 72: Block programs configuration

Note: For instance, *PWM 6* is assigned to *Actuator Output s3*.

For more information on *Actuator* and *PWM* blocks, see [Actuator block](#) and [PWM block](#) of **Block Programs** section.

3.5.4.2 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.

3.5.4.2.1 Volz DA26 - RS485

Firstly, the following wiring connection is recommended for a **RS485 connection between Volz DA26 servos and Veronte Autopilot 1x**:

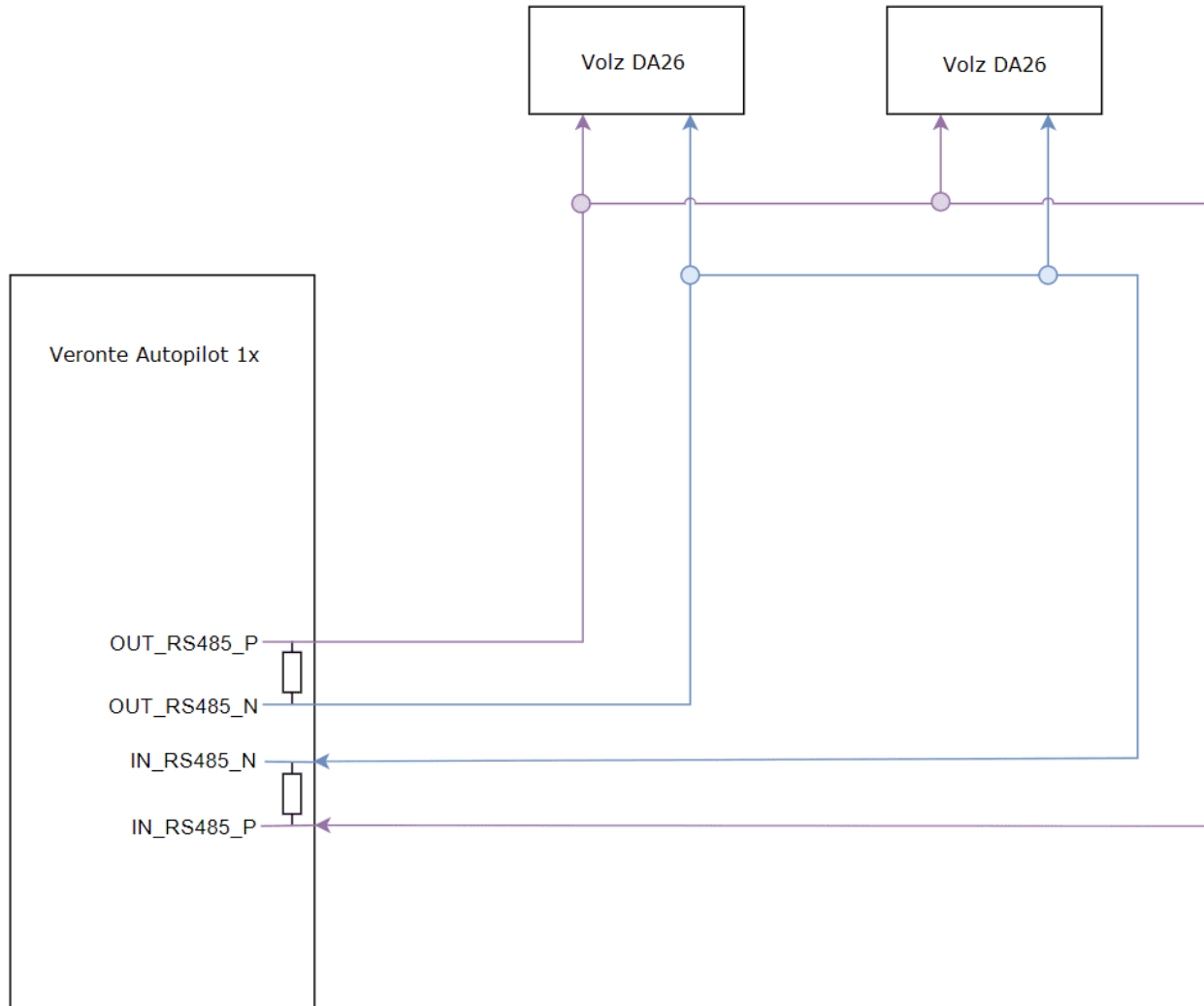


Fig. 73: Volz DA26 - Veronte Autopilot 1x wiring connection

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 section**.

Bidirectionally connect the **RS485** port to a **RS custom message**, in this example *RS custom message 1* is used:

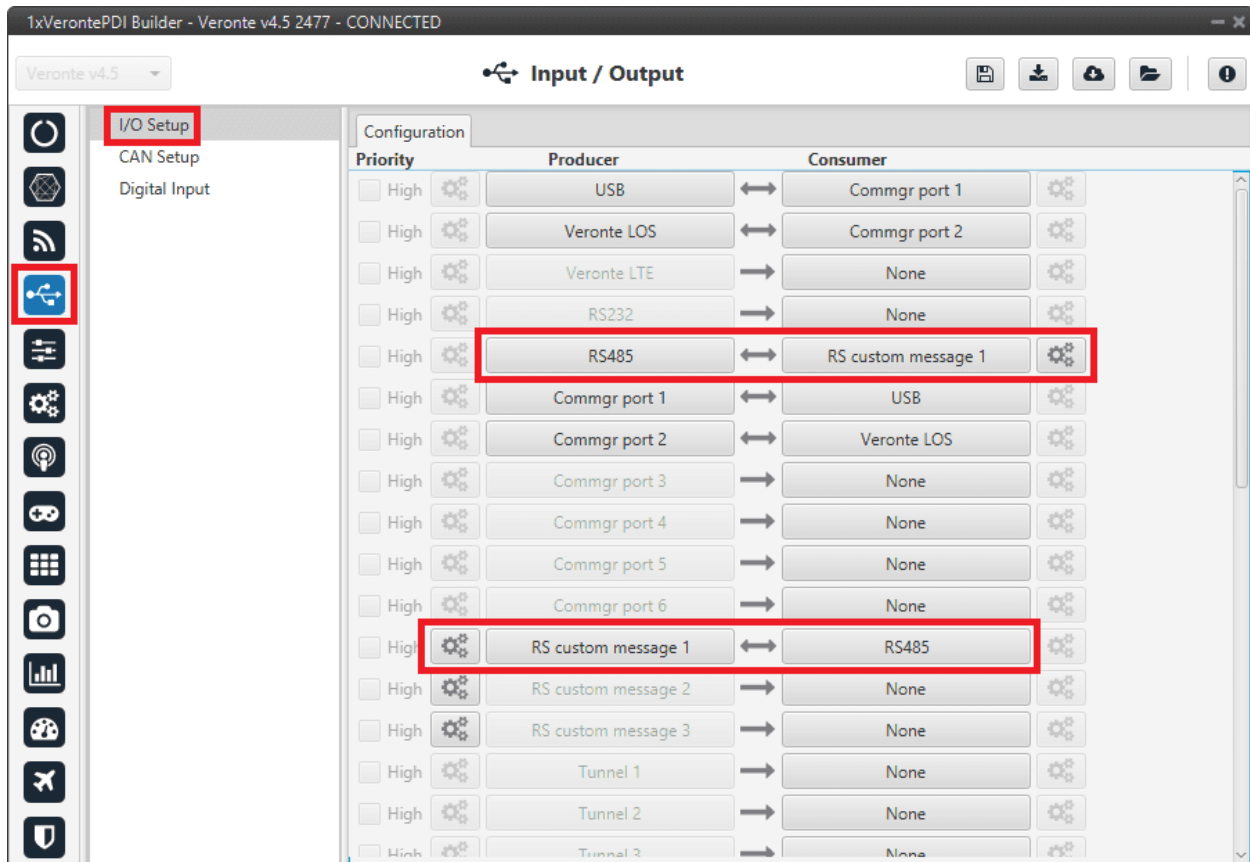


Fig. 74: RS485 ↔ RS custom message 1

2. Configure the *RS custom message 1* producer 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.

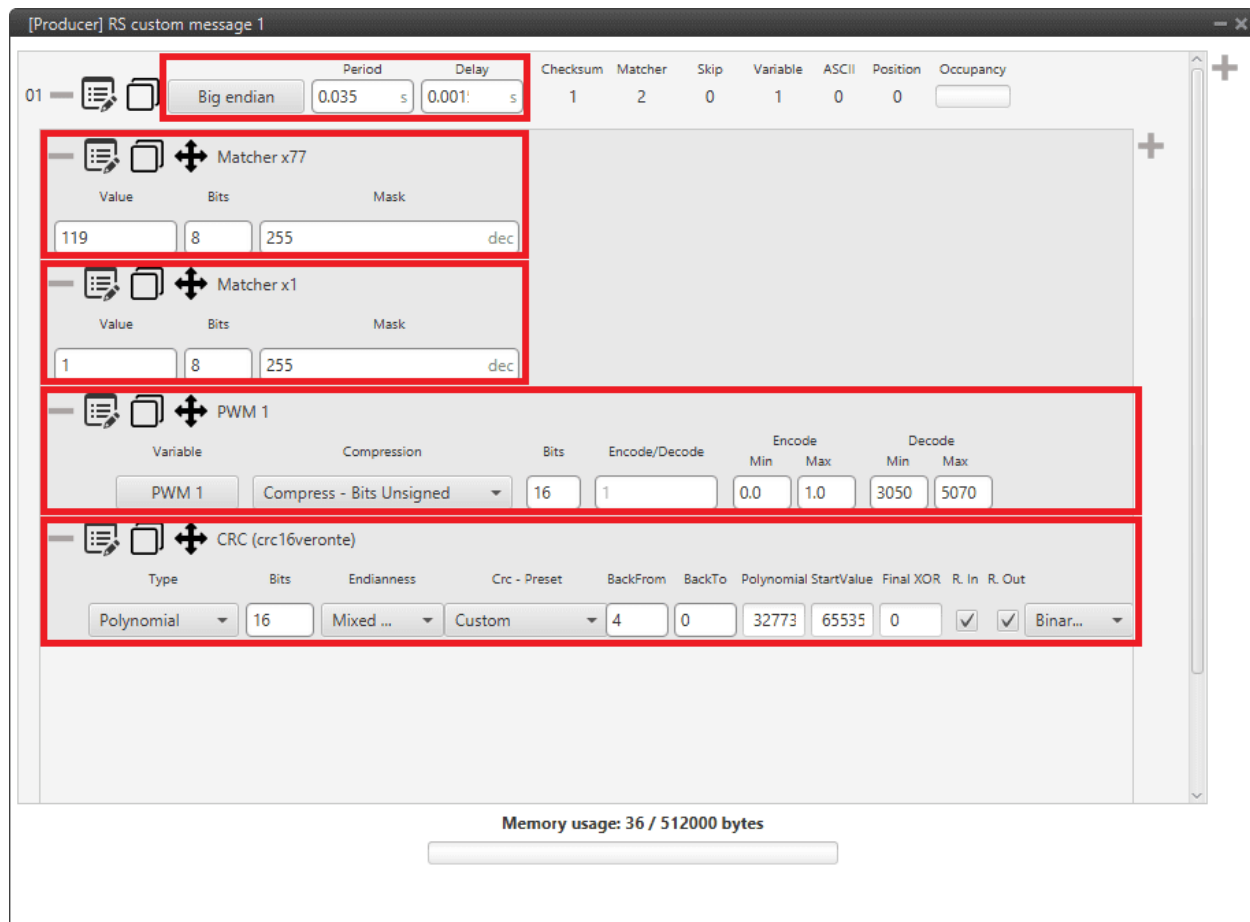


Fig. 75: RS custom message 1 - Manufacturer's communication protocol

- **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:** Mixed 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 - I/O section* of this manual.

3.5.5 Stick

3.5.5.1 PPM Stick

3.5.5.1.1 General case: GND unit sends commands directly to the air unit

Follow the steps below to perform a correct stick configuration on the ground and air units.

Ground unit

1. Go to Input/Output menu → **Digital Input section**.

Make sure that the following parameters have been configured:

- Producer: **CAP 1**
 - Enabled
 - Select the pin to which the transmitter is connected (normally *EQEP A (i.e., GPIO 17)*)
 - Edge detection: First rising edge
- Consumer: **PPM 1**

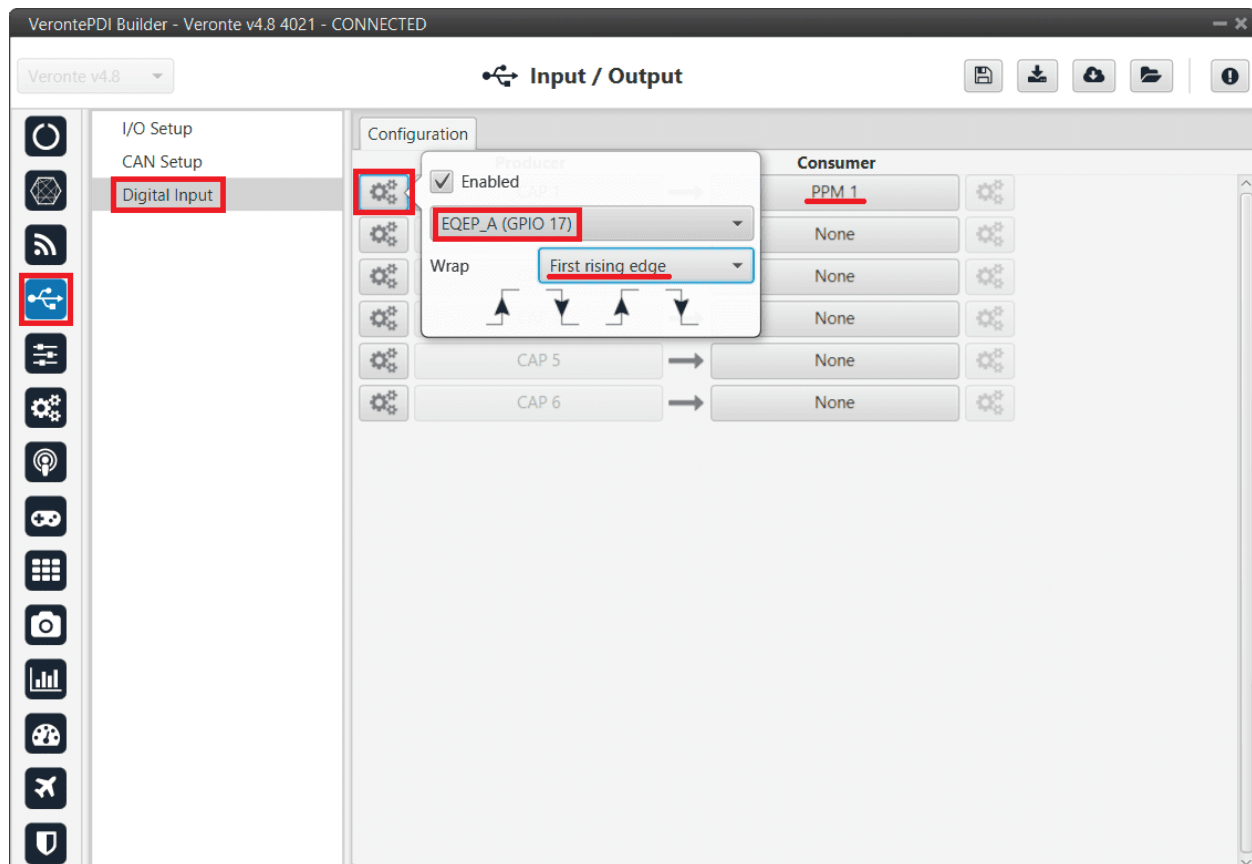


Fig. 76: Stick - Digital Input configuration

- Go to Connections menu → **GPIO section**.

Verify that the pin to which the transmitter is connected, in this case *GPIO 17 (i.e., EQEP A)*, is set as input.

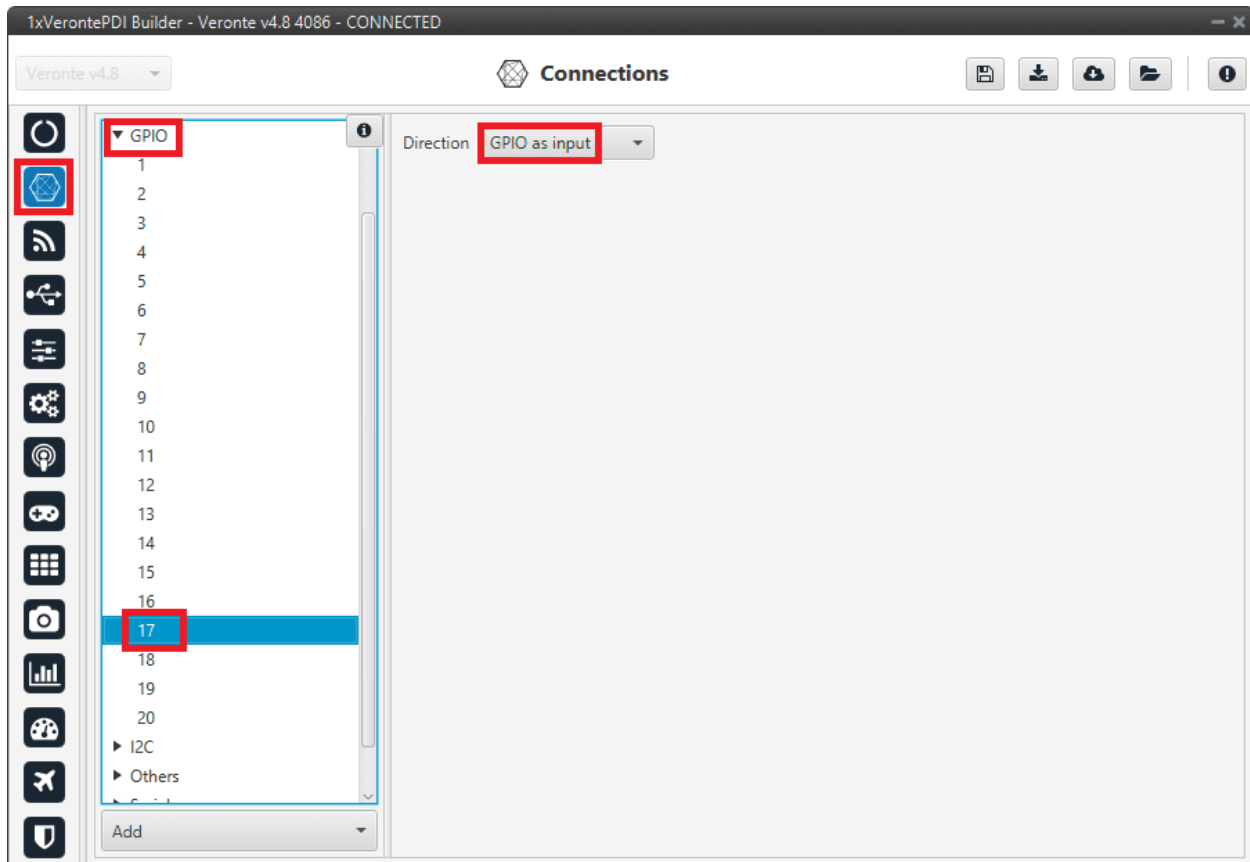


Fig. 77: Stick - GPIO/EQEP configuration

3. Go to Stick menu → Transmitter 1 section → **PPM tab**.

Select the brand of transmitter that applies.

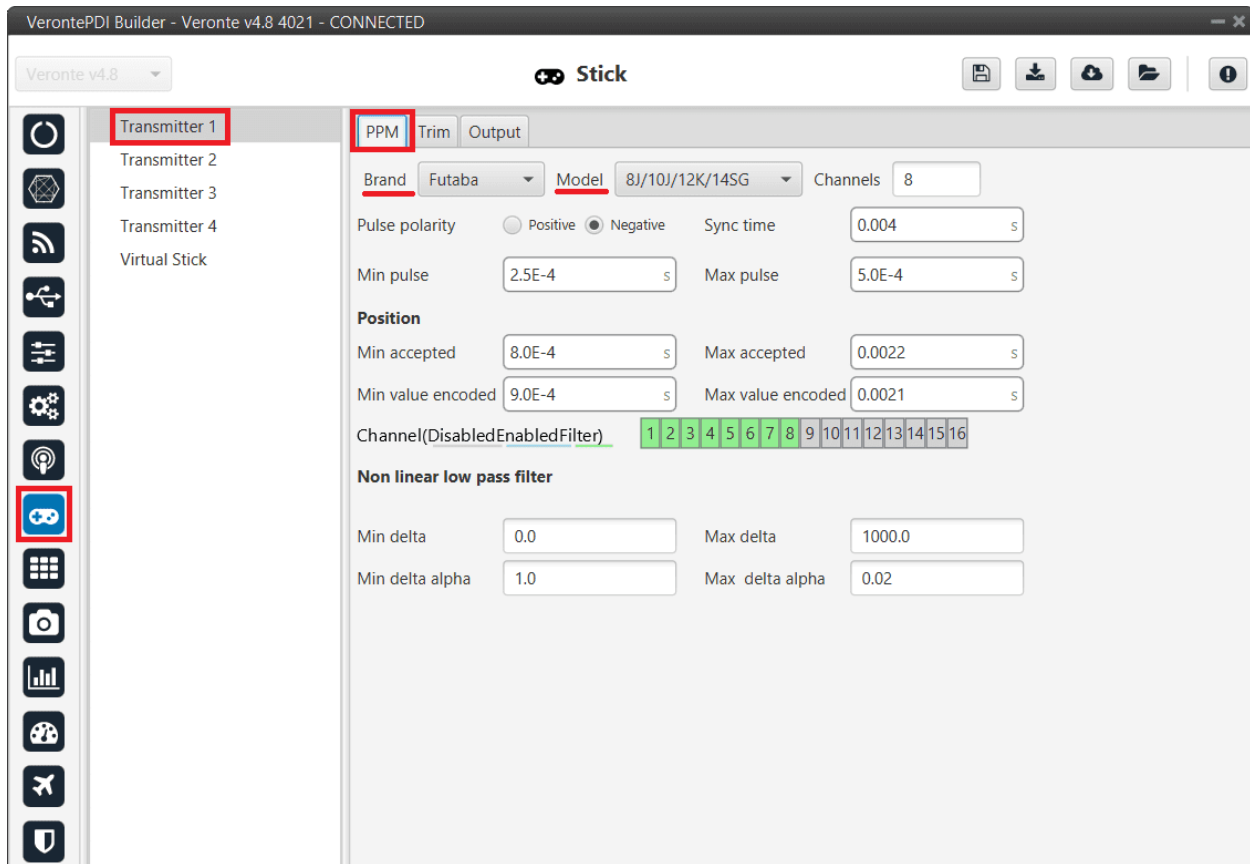


Fig. 78: Stick - PPM configuration

4. Go to Stick menu → Transmitter 1 section → **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 *Output tab* of the Stick section.

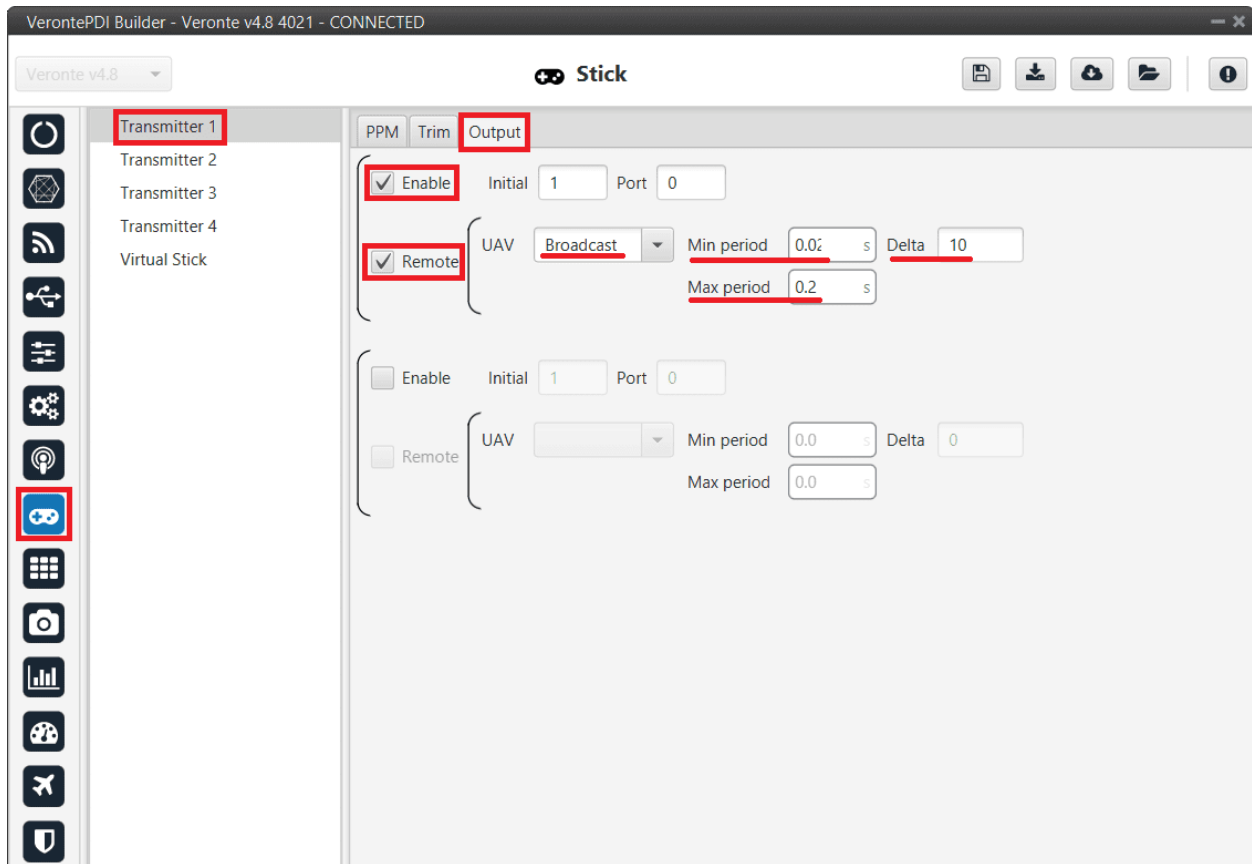


Fig. 79: Stick - Output configuration

If all these settings are correct, users can check that ‘Stick PPM 1 not detected’ variable of the **GND unit** will be true.

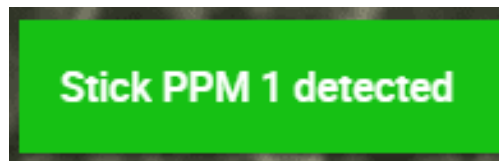


Fig. 80: Stick PPM 1 not detected variable - True

Air unit

1. Go to Stick menu → Transmitter 1 section → **PPM tab**.
Select the brand of transmitter that applies (make the same configuration as the ground unit).
2. Go to Stick menu → Transmitter 1 section → **Output tab**.
Just click on **Enable**.

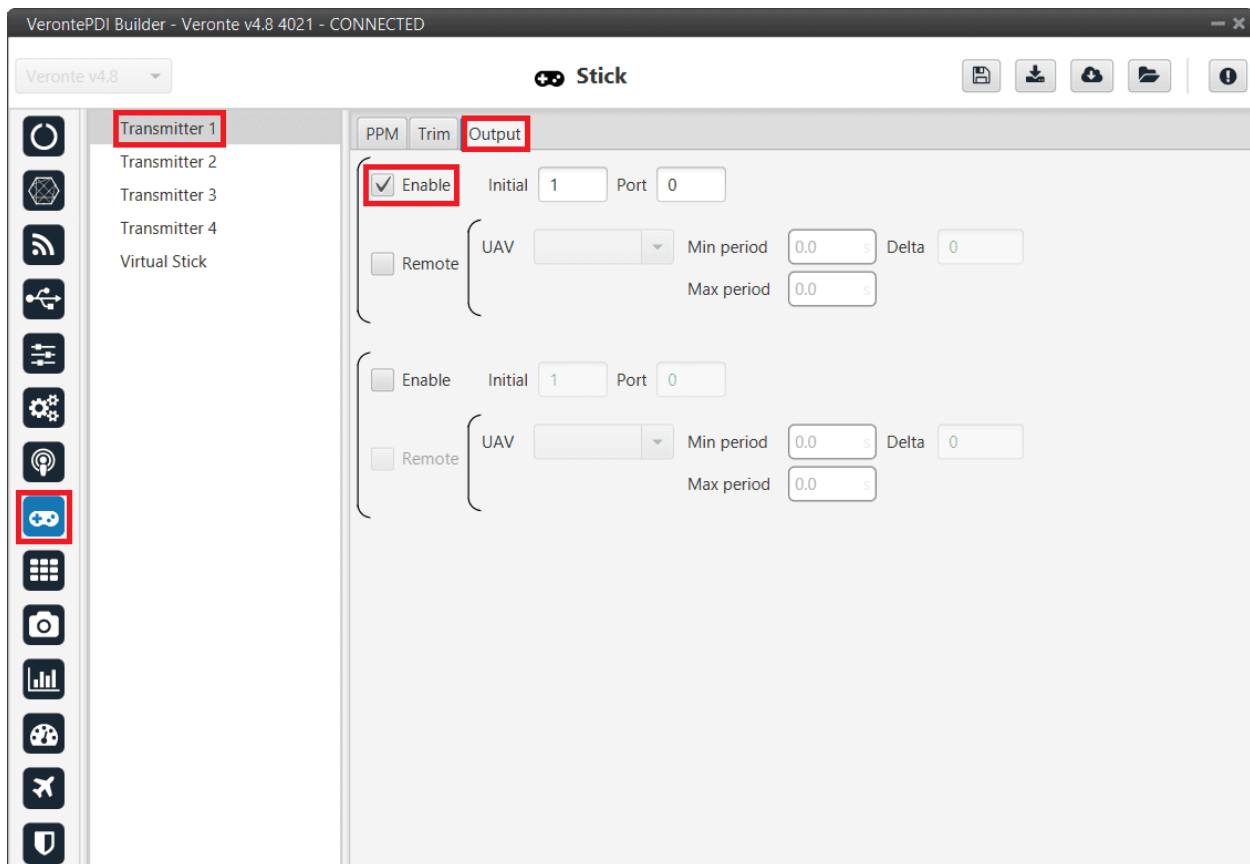


Fig. 81: Stick - Output configuration

- Go to Block Programs menu → **Stick program** → Double click on the **Stick block** → **Edit sources**.

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**.

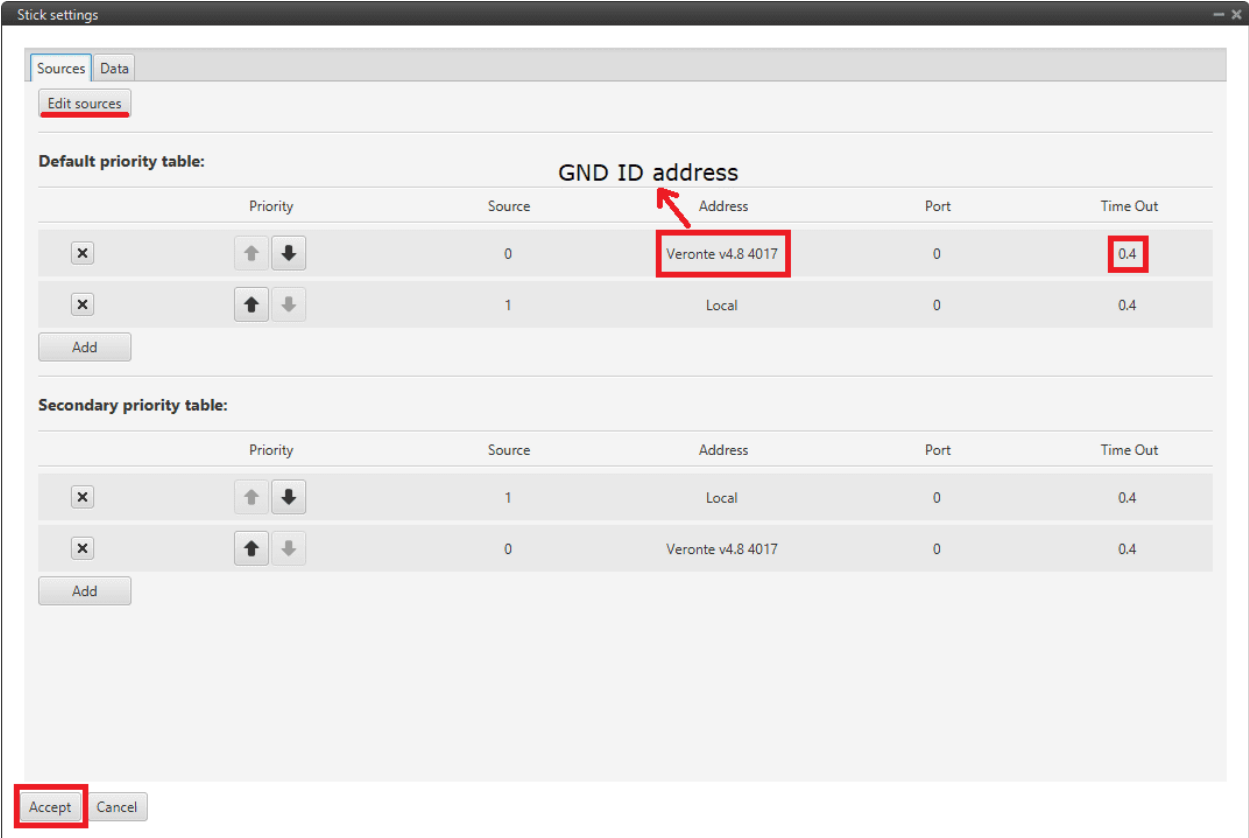


Fig. 82: Stick block configuration

Then, if all is correct, users can check that ‘Stick not detected’ variable of the **AIR unit** will be true.



Fig. 83: Stick not detected variable - True

And that means that the communication between the GND and the AIR unit is correctly configured.

3.5.5.1.2 Simulation case (HIL)

In this case, the user is only using one 1x autopilot.

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.

3.5.5.1.3 On-board PPM receiver case

In that case, follow the next steps:

Ground unit

1. Go to Stick menu → Transmitter 1 section → **PPM tab**.

Select the brand of transmitter that applies.

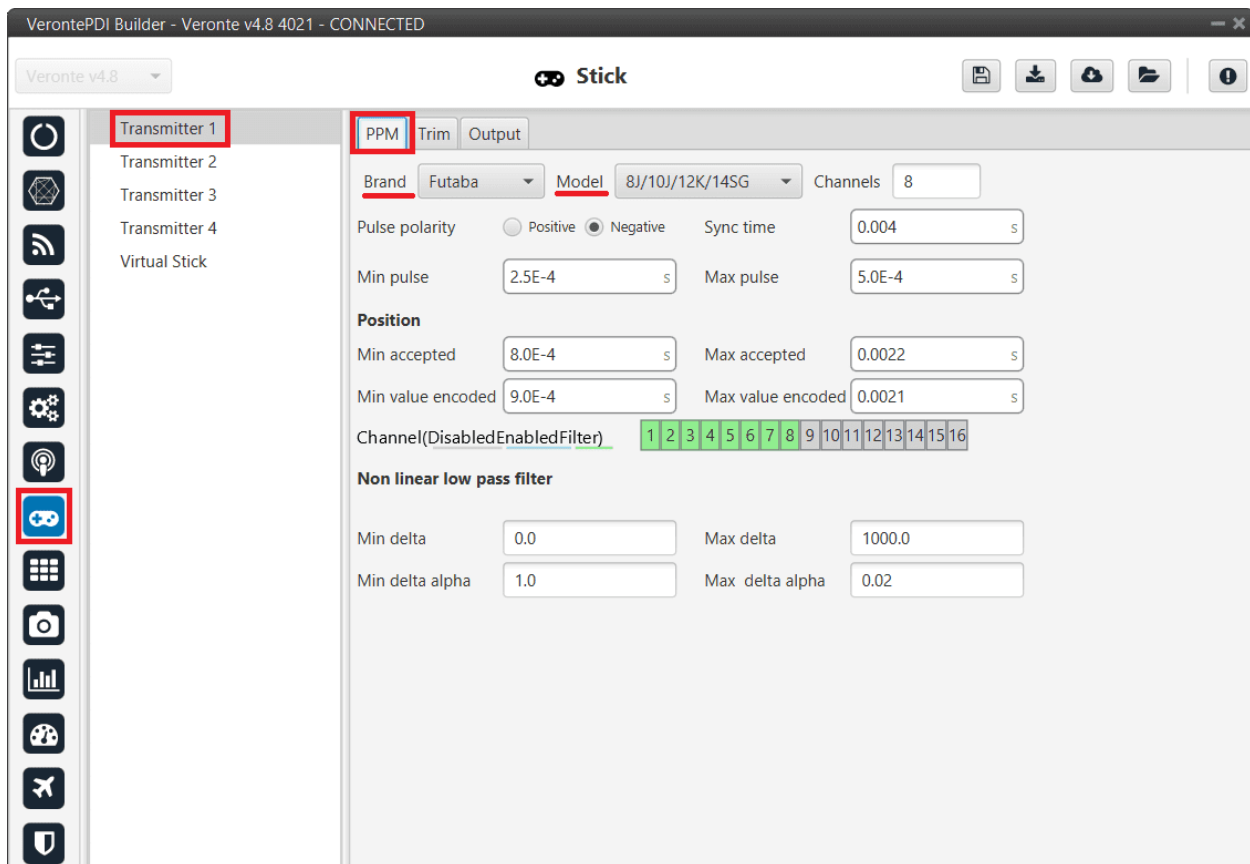


Fig. 84: Stick - PPM configuration

Air unit

1. Go to Input/Output menu → **Digital Input section**.

Make sure that the following parameters have been configured:

- Producer: **CAP 1**
 - Enabled

- Select the pin to which the transmitter is connected (normally *EQEP A* (i.e., *GPIO 17*))
- Edge detection: First rising edge
- Consumer: **PPM 1**

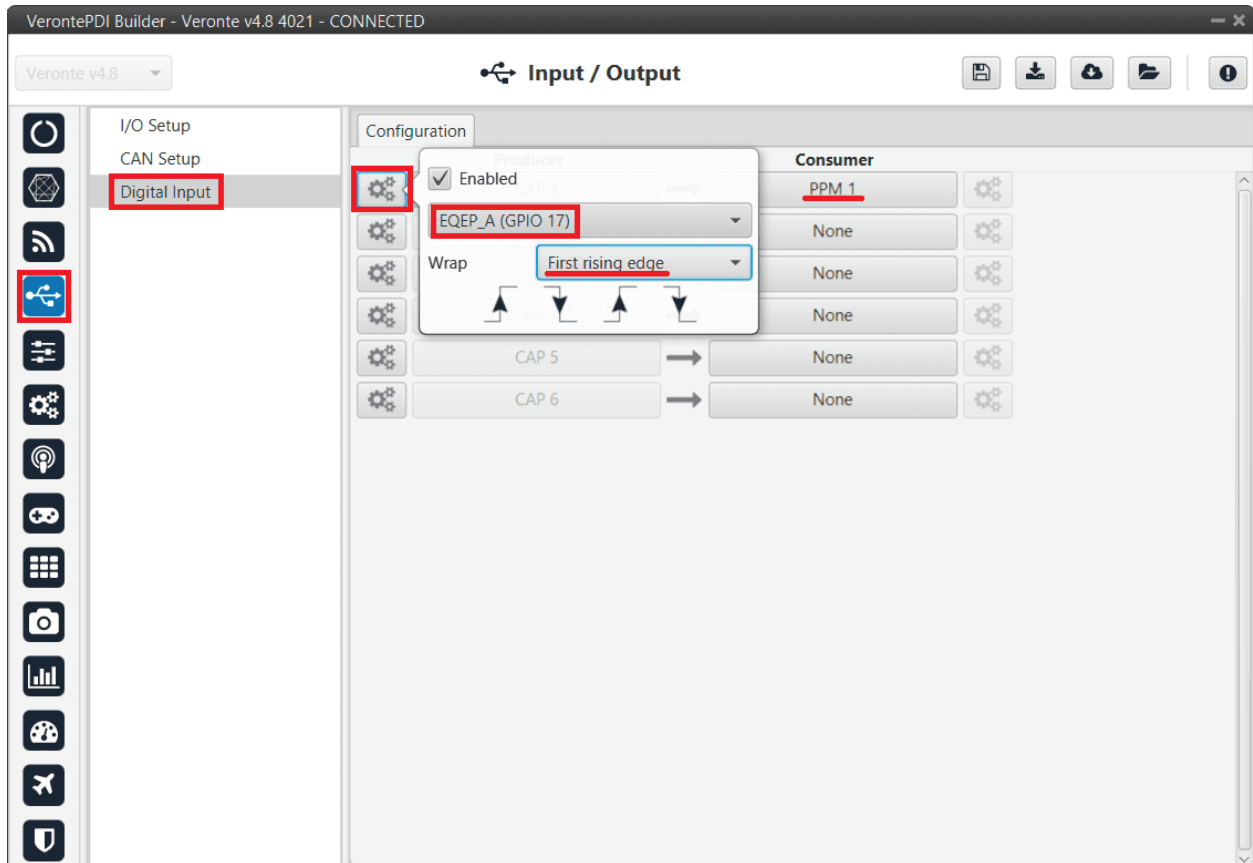


Fig. 85: Stick - Digital Input configuration

2. Go to Connections menu → **GPIO section**.

Verify that the pin to which the transmitter is connected, in this case *GPIO 17* (i.e., *EQEP A*), is set as input.

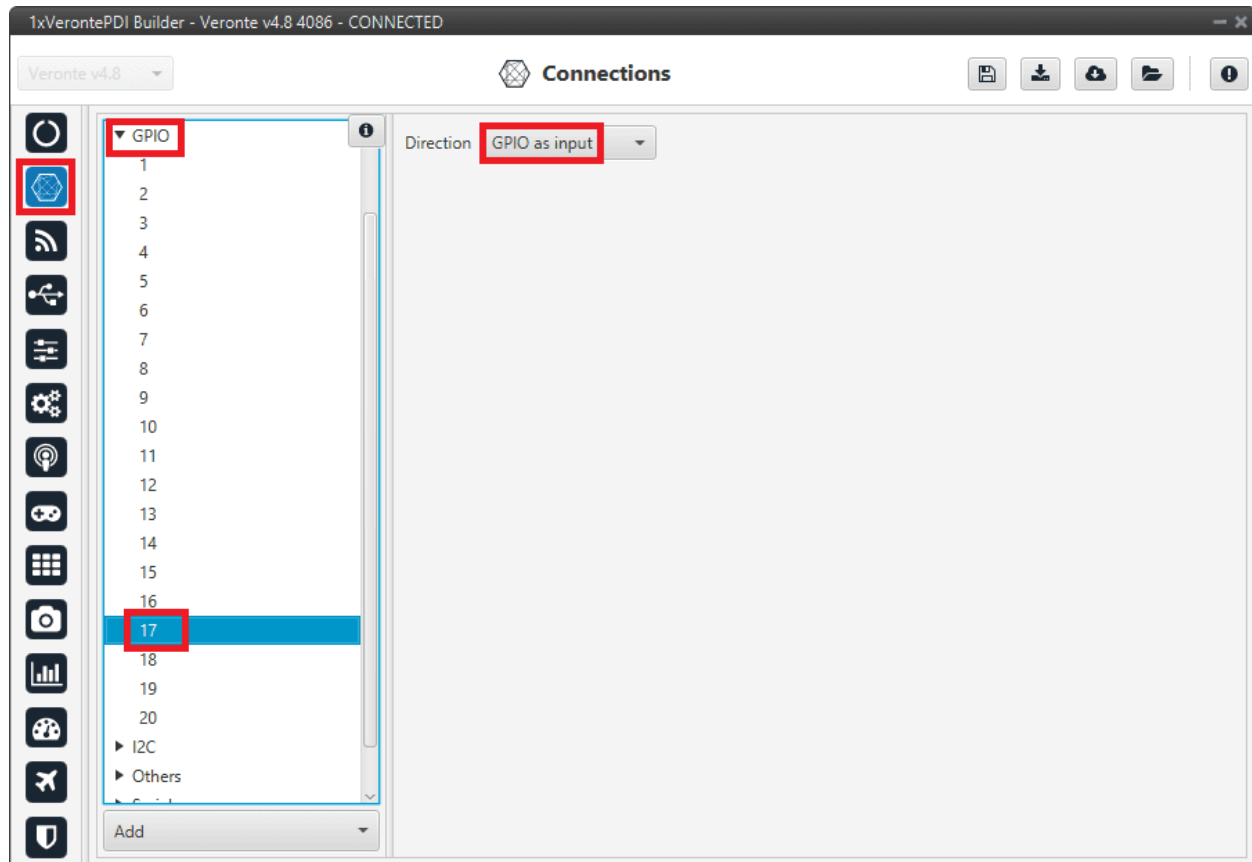


Fig. 86: Stick - GPIO/EQEP configuration

3. Go to Stick menu → Transmitter 1 section → **PPM tab**.

Select the brand of transmitter that applies.

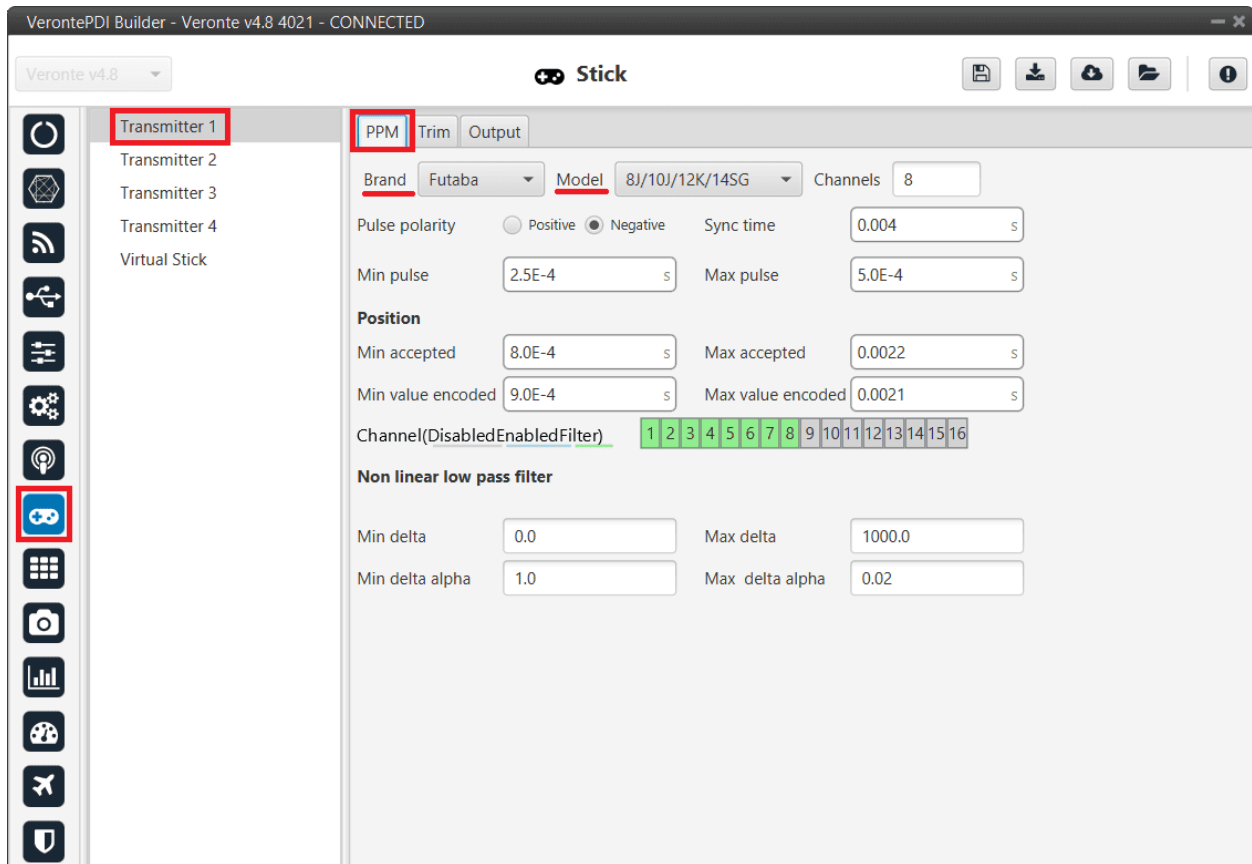


Fig. 87: Stick - PPM configuration

4. Go to Stick menu → Transmitter 1 section → **Output tab**.

Just click on **Enable**.

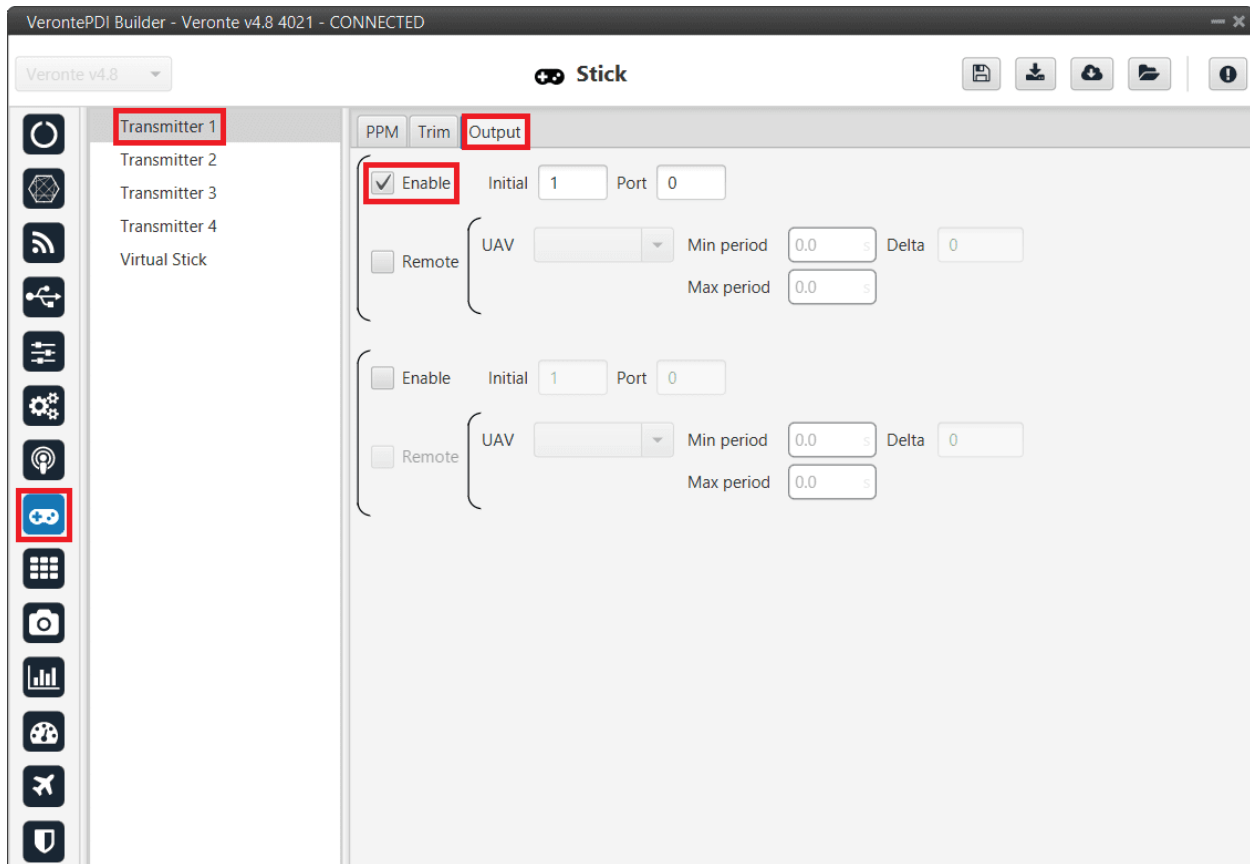


Fig. 88: Stick - Output configuration

If all these settings are correct, users can check that ‘Stick PPM 1 not detected’ variable of the **AIR unit** will be true.

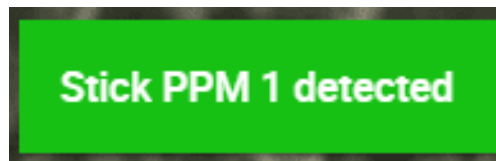


Fig. 89: Stick PPM 1 not detected variable - True

- Go to Block Programs menu → **Stick program** → Double click on the **Stick block** → **Edit sources**.

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**.

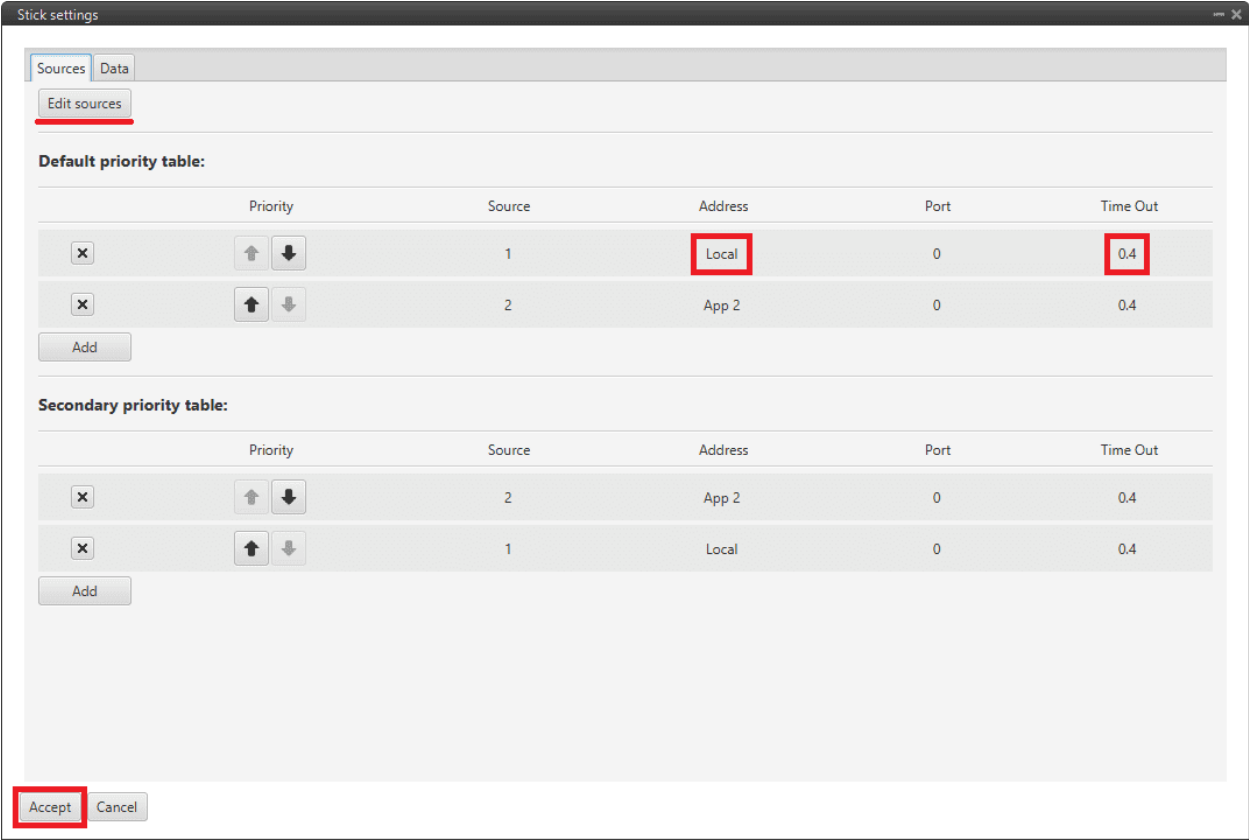


Fig. 90: Stick block configuration

Then, if all is correct, users can check that ‘Stick not detected’ variable of the AIR unit will be true.



Fig. 91: Stick not detected variable - True

And that means that the communication between the GND and the AIR unit is correctly configured.

3.5.5.2 USB joystick

Veronte software is able to detect USB devices such as joystick. The axis of these devices can be read and configured to send stick information to Veronte Autopilot 1x.

To configure them:

1. Connect the USB joystick to the computer.
2. Configure a Virtual Stick as explained in *Virtual Stick Integration*.

3.5.5.3 Virtual Stick

To configure a virtual stick, follow the next steps:

1. Go to Stick menu → Virtual Stick section → **Input variable tab**.

Enable the virtual stick and enter an **update period** (we recommend **0.02 s**).

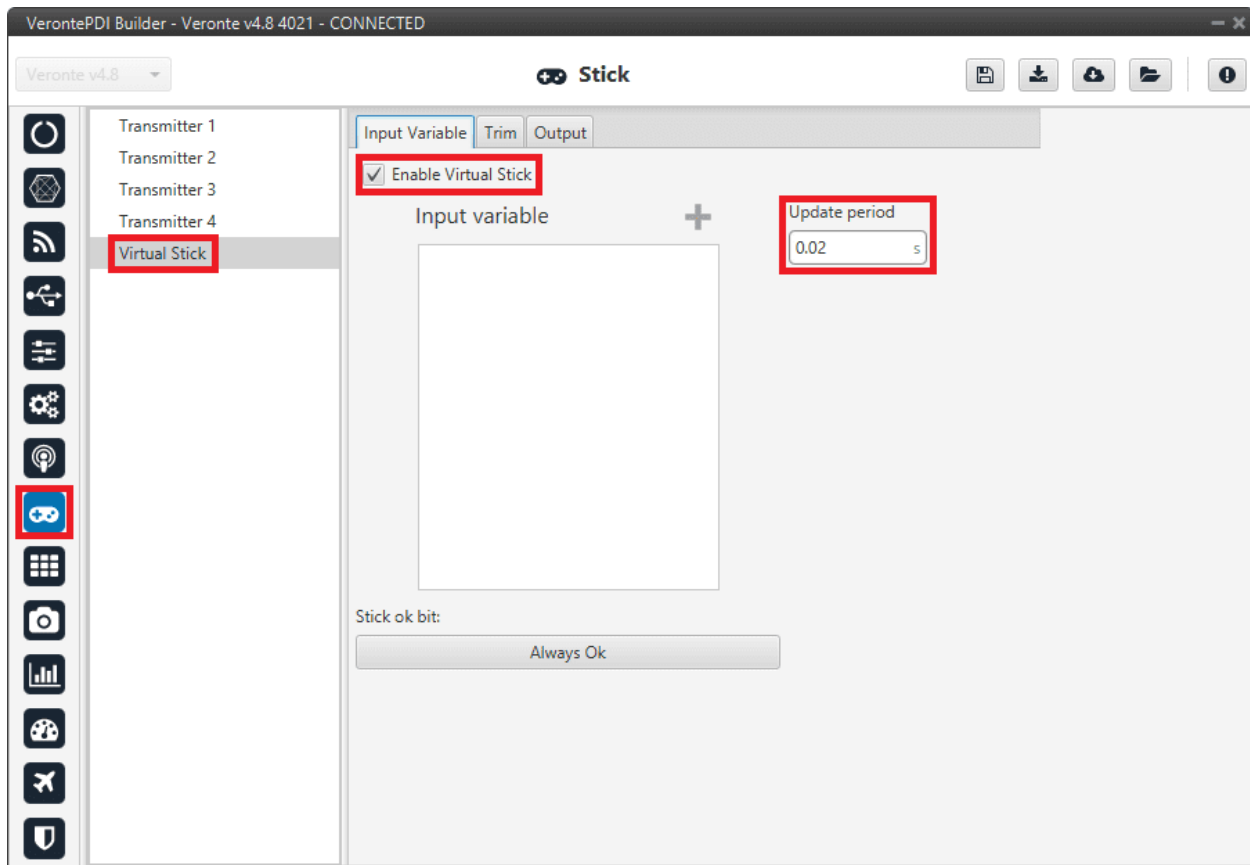


Fig. 92: Virtual Stick configuration

2. Go to Stick menu → Virtual Stick section → **Output tab**.

Just click on **Enable**.

3. Go to Block Programs menu → **Stick program** → Double click on the **Stick block** → **Edit sources**.

Input **App 2** to receive the stick information from the virtual stick widget and put it as the **highest priority** in the priority table. We recommend a Time Out of **0.4 s**.

4. Configure a **Virtual Stick Widget**.

Please find an example of how to configure it in [Virtual stick widget](#) in the Integration examples section of the **Veronte Ops** manual.

If the user creates a virtual stick to process the information received through a different channel than PPM (e.g., by CAN or ADC), the user will also have to:

- Go to Stick menu → Virtual Stick section → **Input variable tab**.

Add the variables containing the stick information in **Input Variable**.

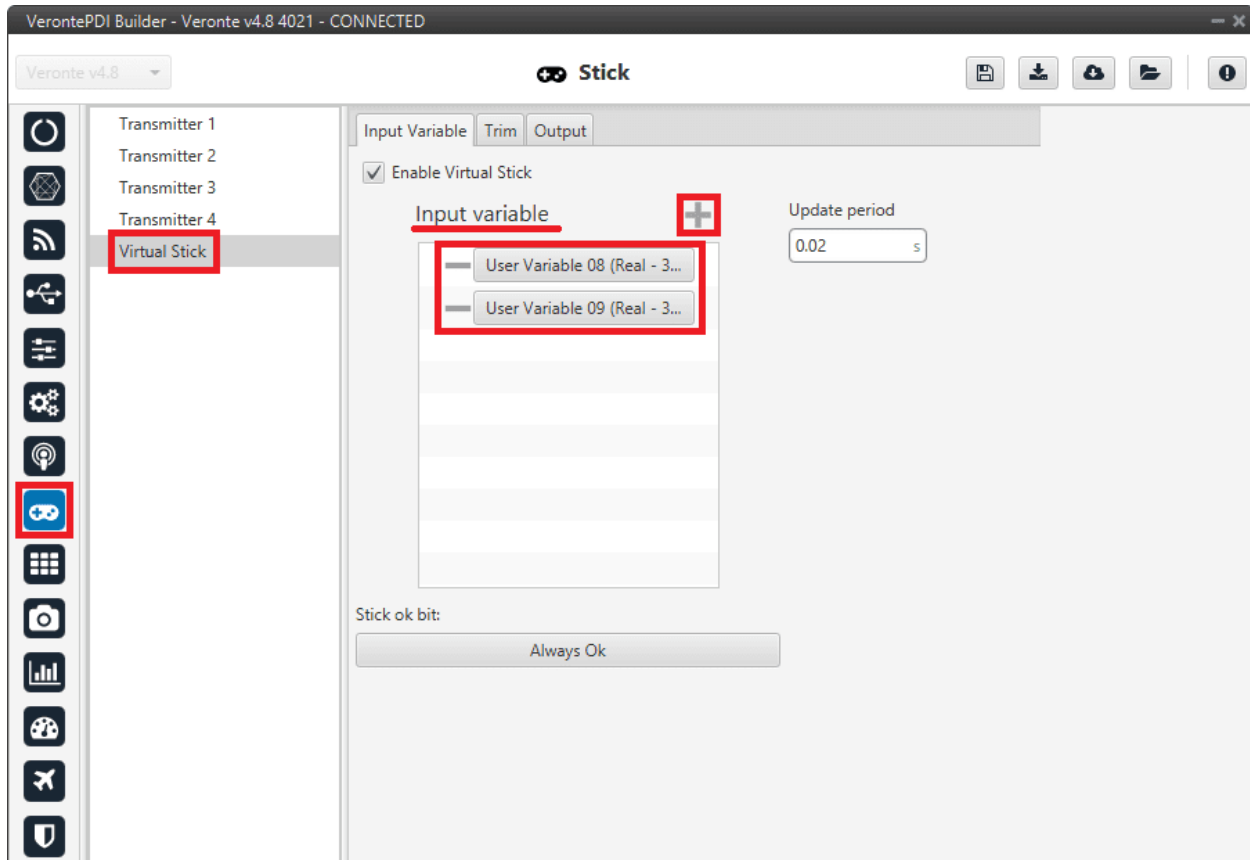


Fig. 93: **Virtual Stick with Input variables configuration**

3.5.6 Veronte products

3.5.6.1 Autopilot 4x

To establish on a **Veronte Autopilot 4x** version 1.8 the **communication between Autopilots 1x and Arbiters** via CAN, the following connection is required:

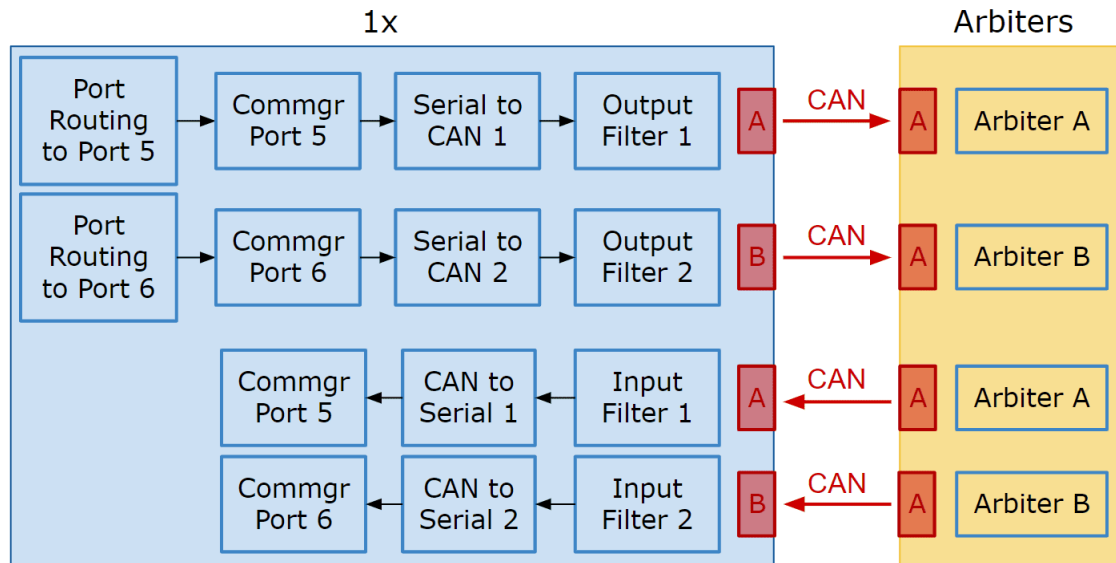


Fig. 94: Communication diagram 1x - Arbiters

Follow the steps below to make this configuration:

1. Go to Communications menu → **Ports section**.

Remove Port 5 and 6 from the Forward group and **Add Port 5 and 6 to the Route group**, with target Arbiters' Address:

- Address of Arbiter A: **50 000** + Serial number
- Address of Arbiter B: **54 000** + Serial number

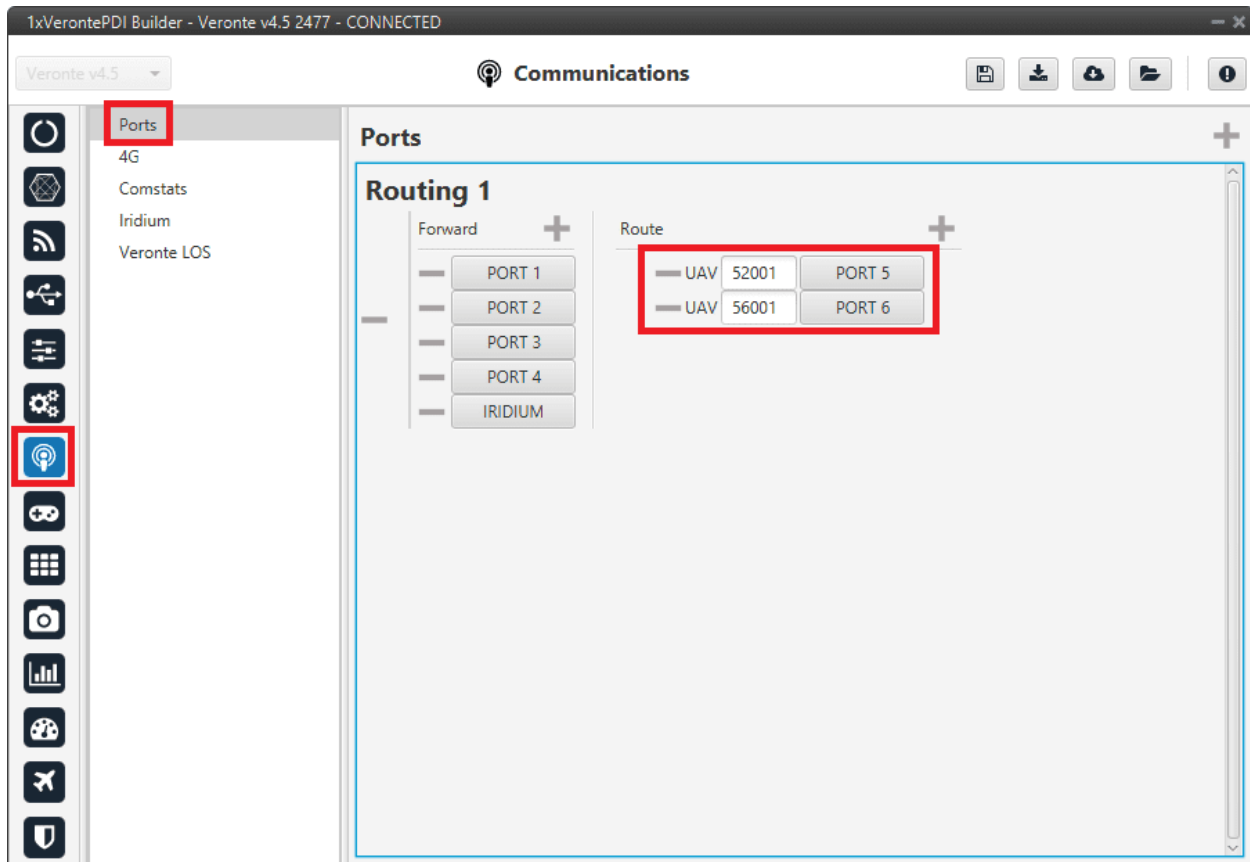


Fig. 95: Routing configuration

Note: This is just an example, users can choose *Ports* other than 5 and 6.

2. Go to Input/Output menu → **I/O Setup** section.

Connect **Commgr Port 5** to **Serial to CAN 1** consumer and **Commgr Port 6** to **Serial to CAN 2** consumer:

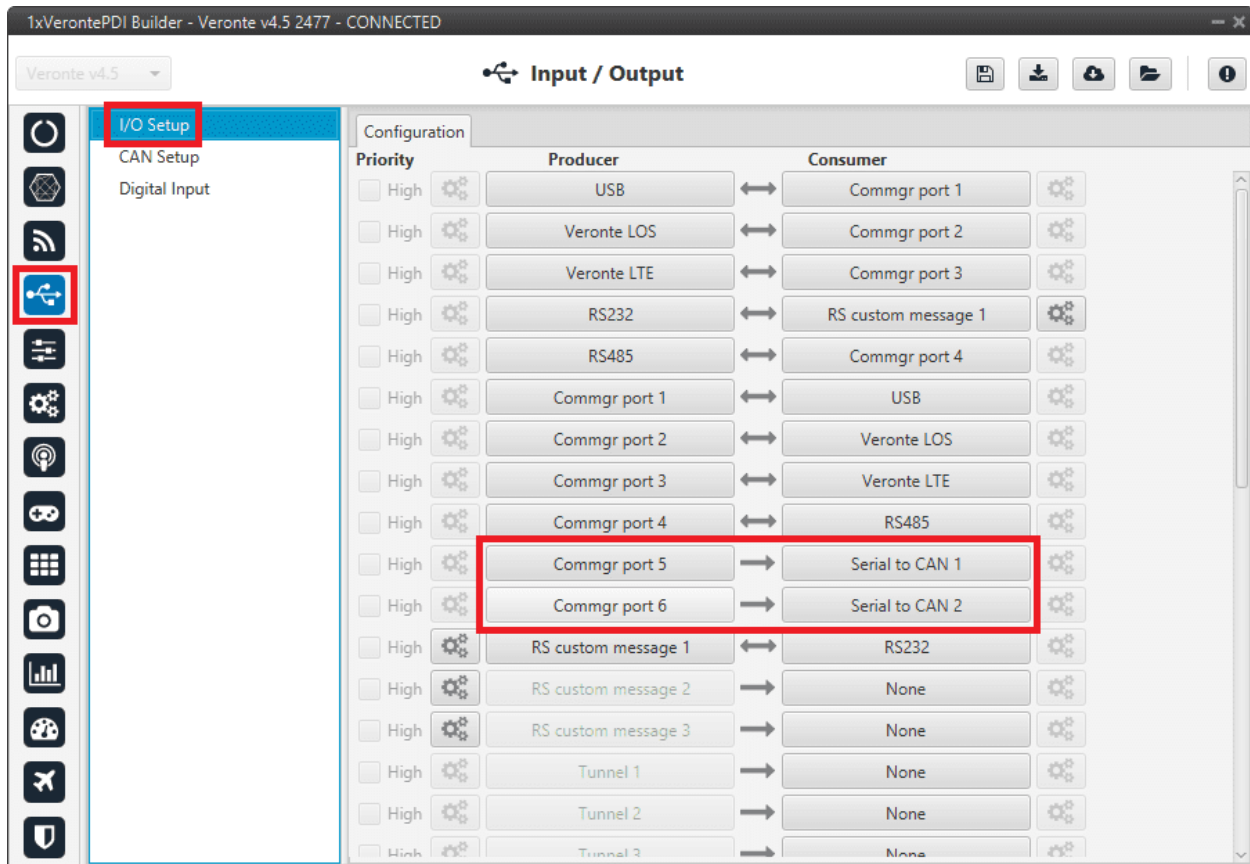


Fig. 96: I/O Setup - Serial to CAN

Then, connect **CAN to Serial 1** to **Commgr Port 5** and **CAN to Serial 2** to **Commgr Port 6**:

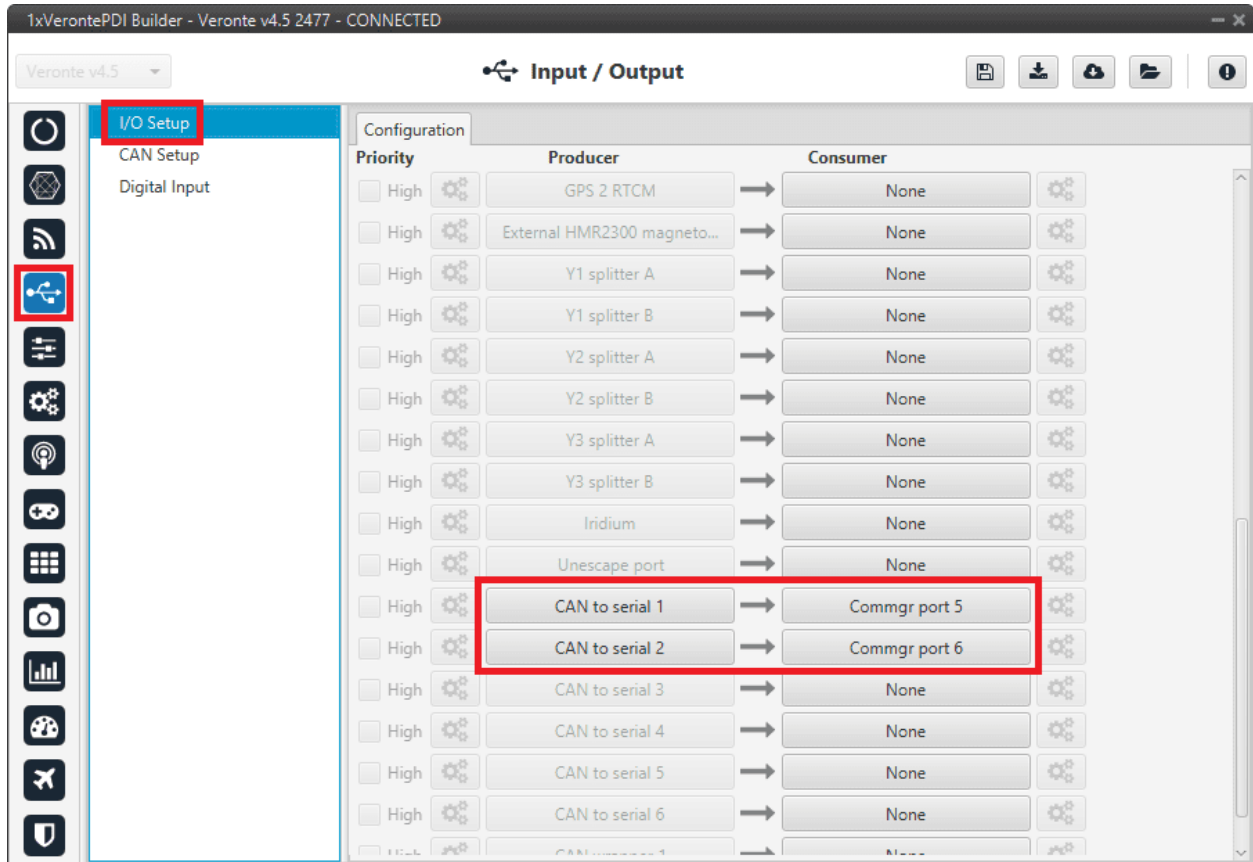


Fig. 97: I/O Setup - CAN to Serial

Note: This is just an example, users can choose *Serial to CAN* and *CAN to Serial* other than 1 and 2.

3. Go to Input/Output menu → CAN Setup section → **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**:

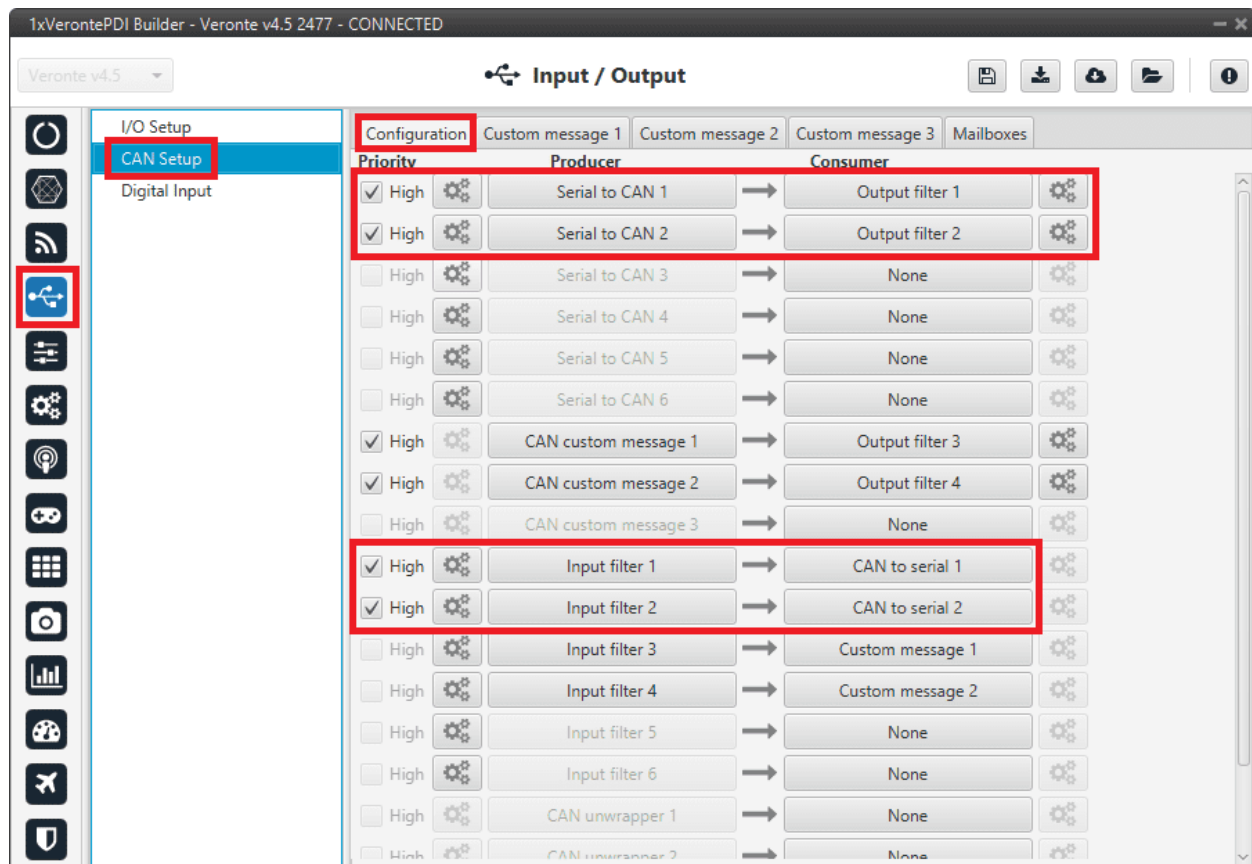


Fig. 98: CAN Setup

- Set for both **Serial to CAN 1/2** the same **CAN ID: 1302**.

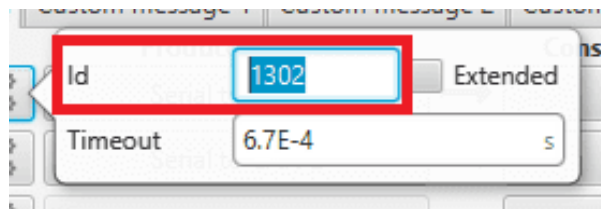


Fig. 99: CAN Setup - Serial to CAN 1/2 configuration

- Select for **Output filter 1** the **CAN A**, as this is the CAN of 1x Autopilot that is connected to Arbiter A:

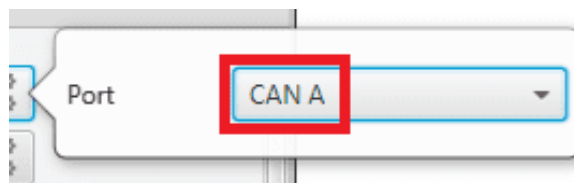


Fig. 100: CAN Setup - Output filter 1 configuration

- Select for **Output filter 2** the **CAN B**, as this is the CAN of 1x Autopilot that is connected to Arbiter B:

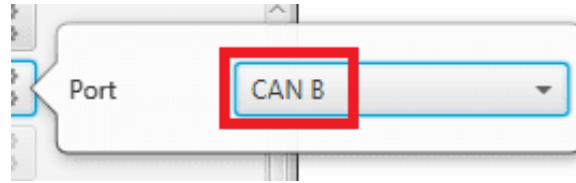


Fig. 101: CAN Setup - Output filter 2 configuration

- Configure both **Input filter 1/2** with **CAN ID:1301**.

Select for **Input filter 1** the **CAN A**, as this is the CAN of 1x Autopilot that is connected to Arbiter A:

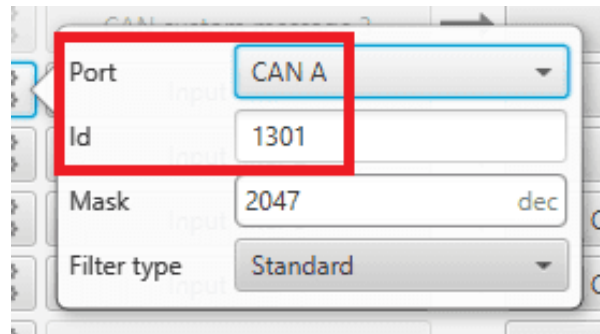


Fig. 102: CAN Setup - Input filter 1 configuration

Select for **Input filter 2** the **CAN B**, as this is the CAN of 1x Autopilot that is connected to Arbiter B:

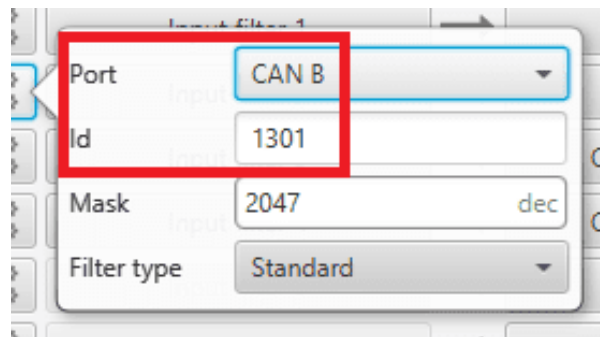


Fig. 103: CAN Setup - Input filter 2 configuration

Note: This is just an example, users can choose *Input filter* and *Output filter* other than 1 and 2.

- Go to Input/Output menu → CAN Setup section → **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**:

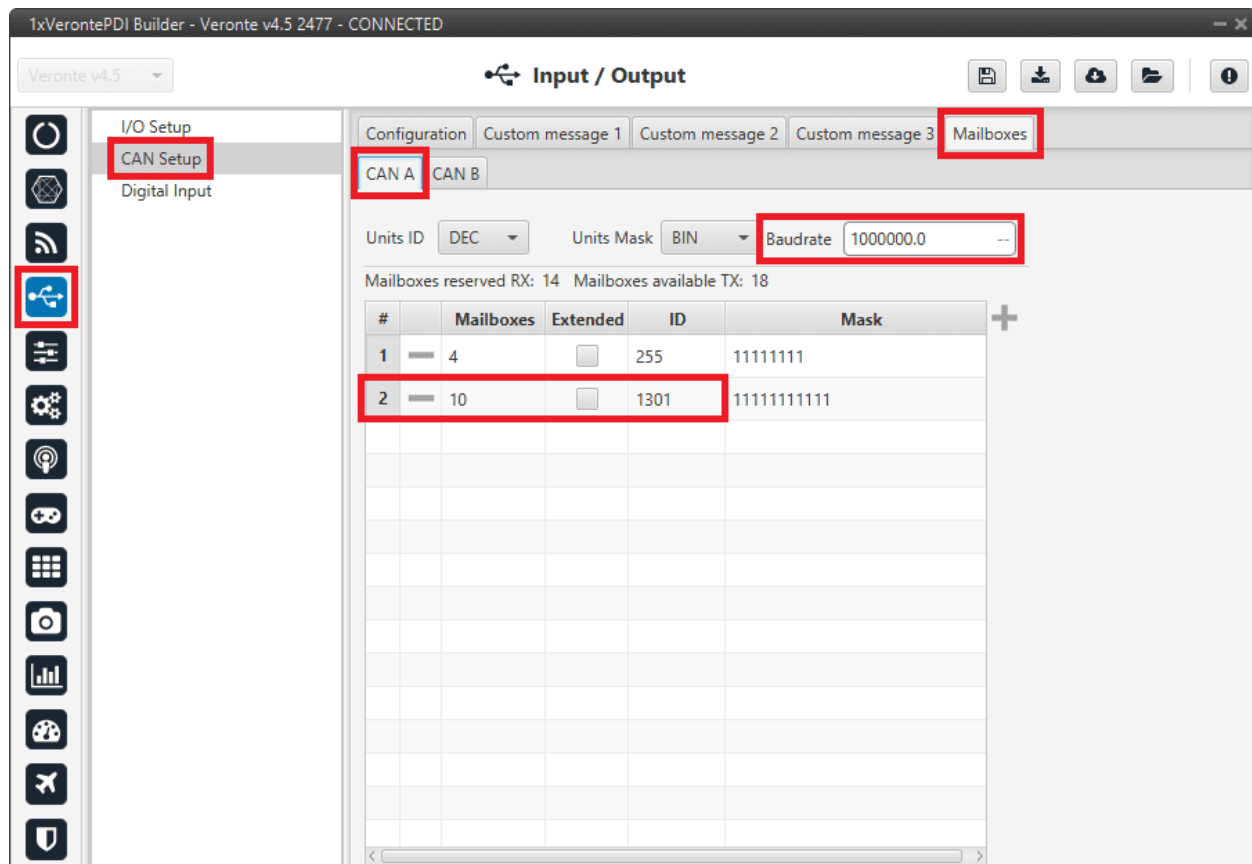


Fig. 104: **Mailboxes - CAN A configuration**

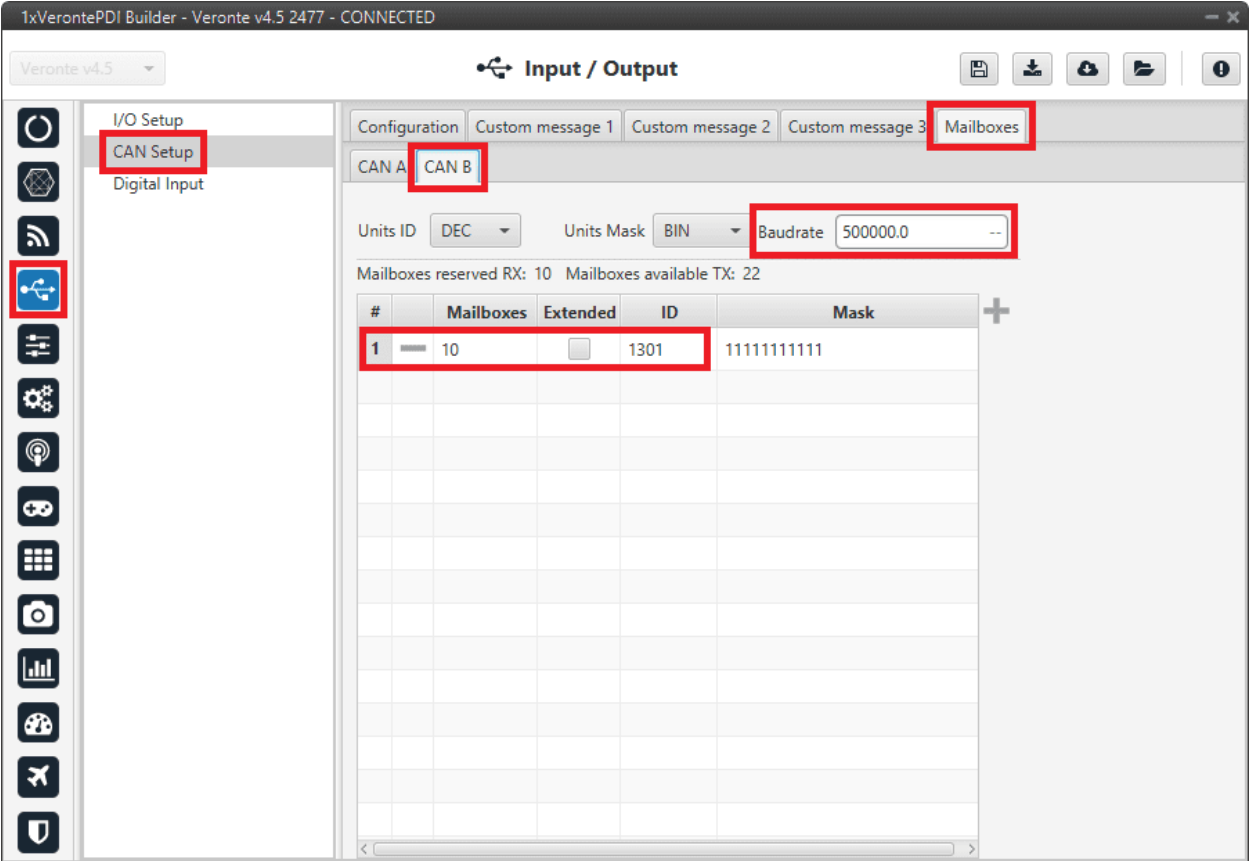


Fig. 105: Mailboxes - CAN B configuration

3.5.6.2 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:

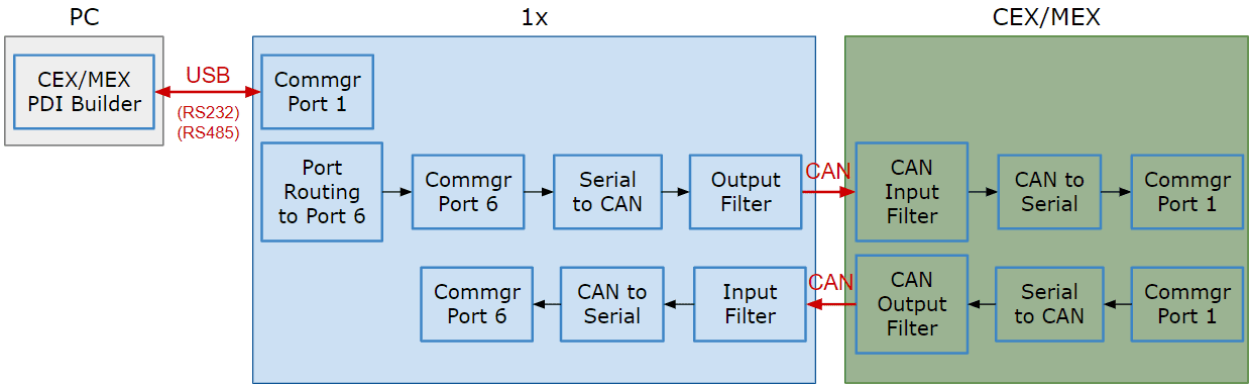


Fig. 106: Communication diagram 1x - CEX/MEX

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** is **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:

1x PDI Builder side

1. Go to Input/Output menu → **I/O Setup section**.

Check the connection between the computer and the 1x (usually via USB, but RS232 and RS485 are also possible).

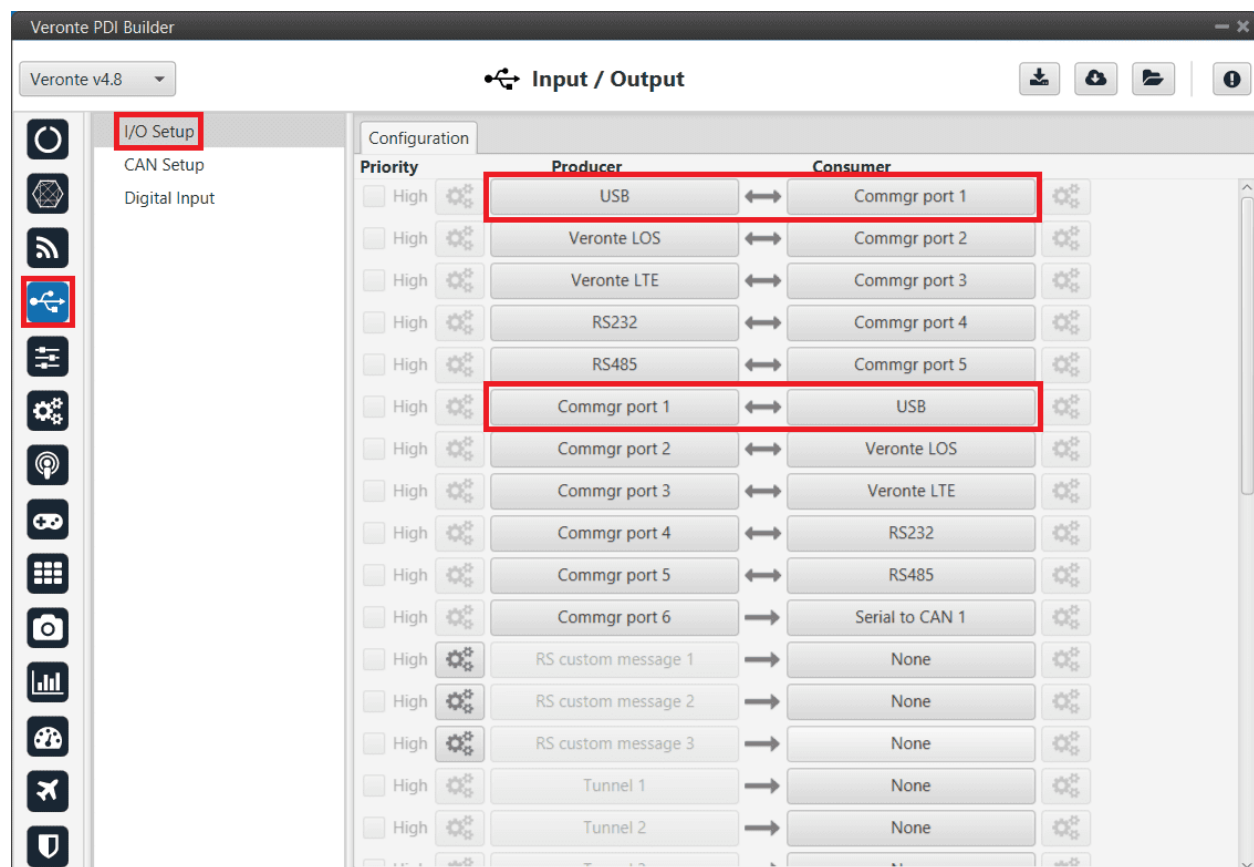


Fig. 107: 1x - USB communication

2. Go to Communications menu → **Ports section**.

Remove Port 6 from the Forward group and **Add Port 6 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 theoretical address does not work, 999 (unknown) can be used as sometimes the address has not been set in CEX/MEX.

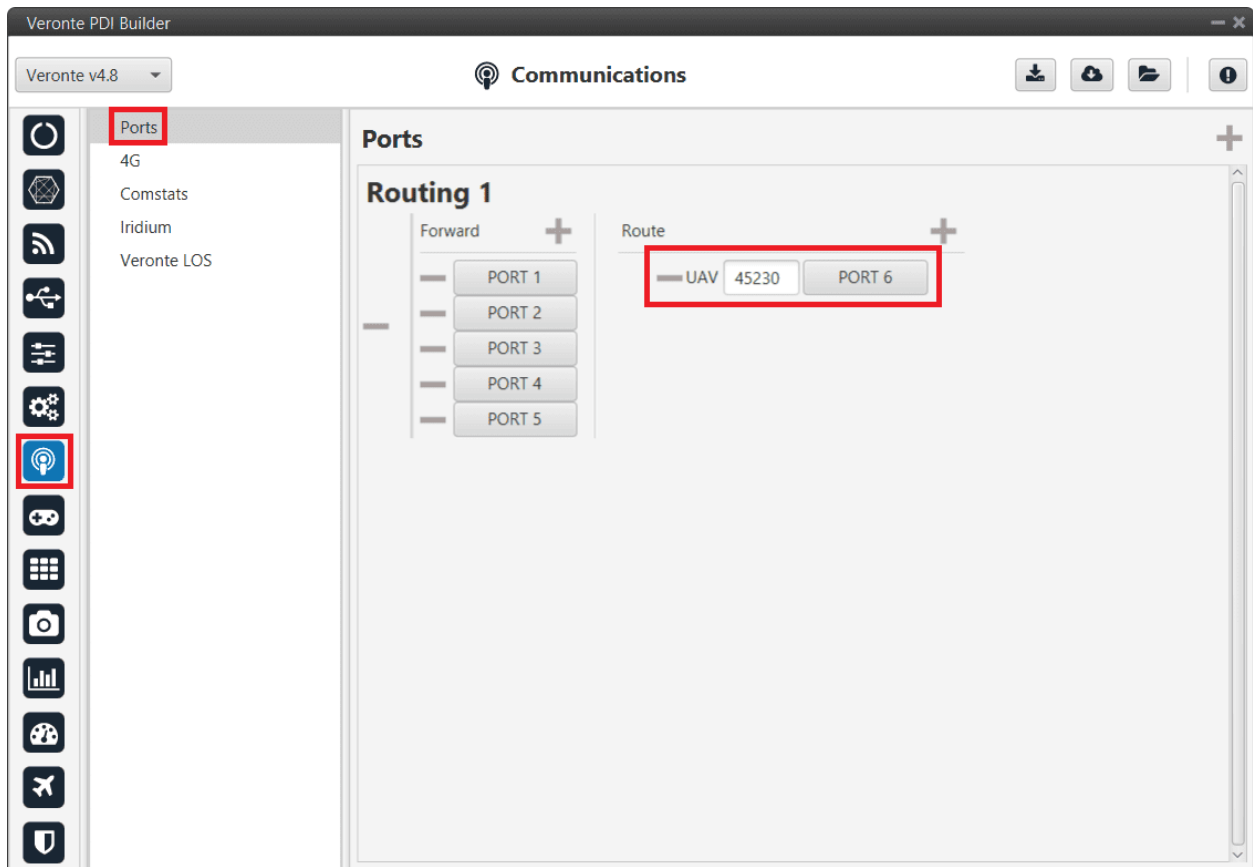


Fig. 108: 1x - Routing

3. Go to Input/Output menu → **I/O Setup** section.

Connect **Commgr Port 6** to **Serial to CAN 1** consumer:

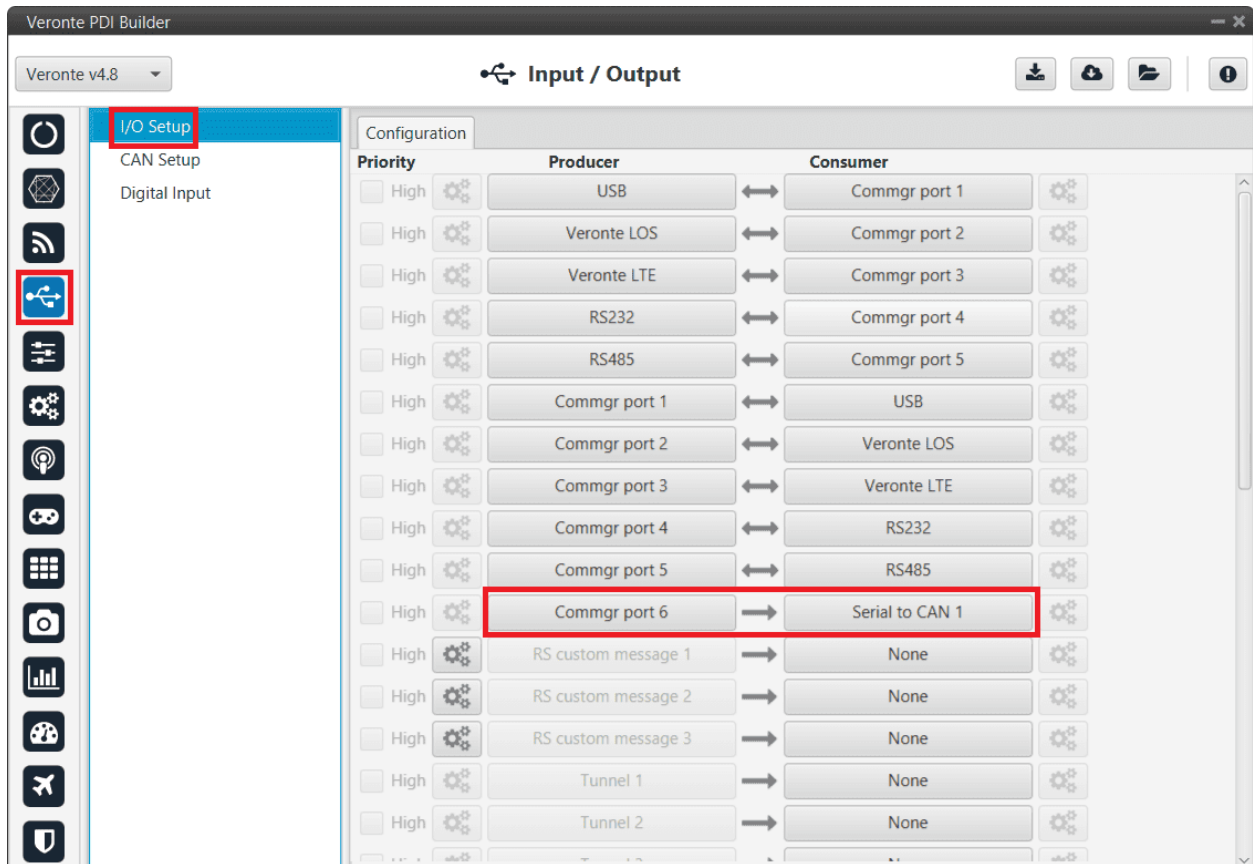


Fig. 109: 1x - I/O Setup - Serial to CAN

Then, connect **CAN to Serial 1** to **Commgr Port 6**:

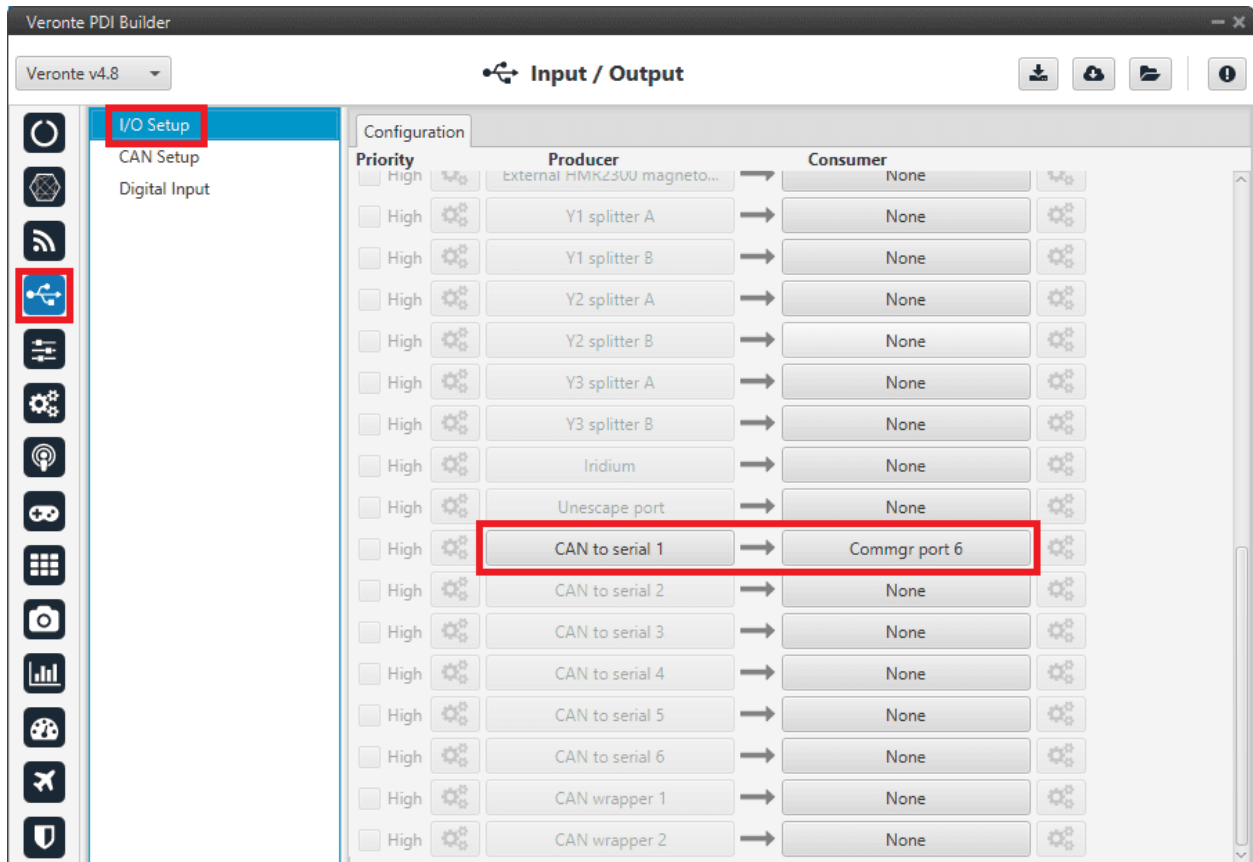


Fig. 110: 1x - I/O Setup - CAN to Serial

4. Go to Input/Output menu → CAN Setup section → **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**:

Warning: For correct communication, mark both as **High priority** (with the Priority checkbox).

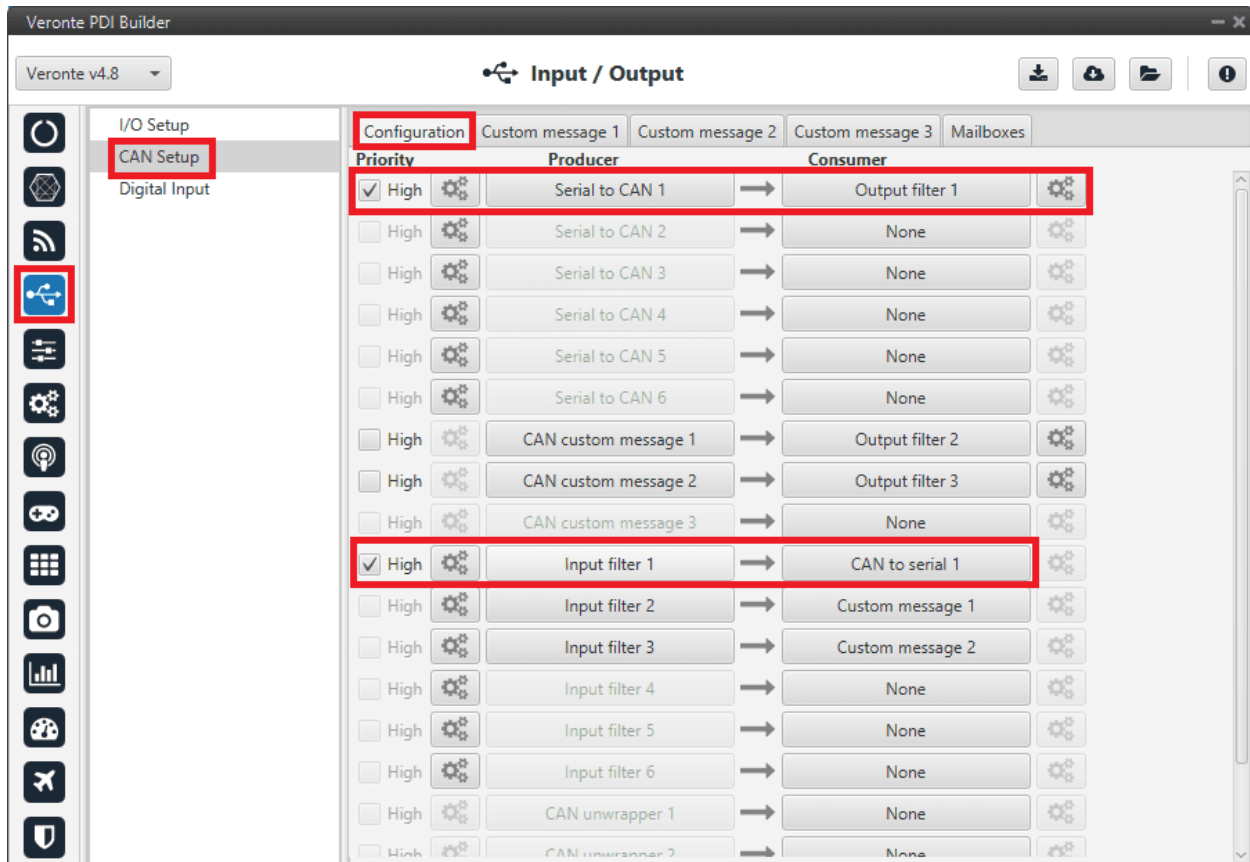


Fig. 111: 1x - CAN Setup

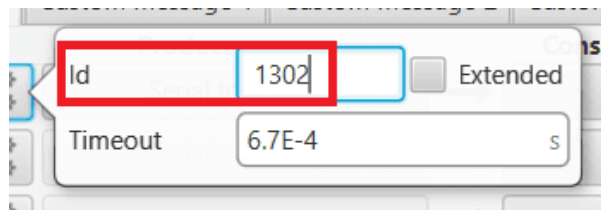


Fig. 112: 1x - CAN Setup - Serial to CAN configuration

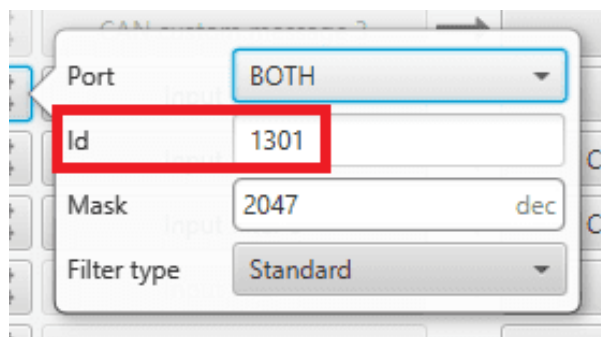


Fig. 113: 1x - CAN Setup - Input filter configuration

- Go to Input/Output menu → CAN Setup section → **Mailboxes tab**.

Finally, configure the reception **mailbox with ID 1301**, assign at least 4 mailboxes:

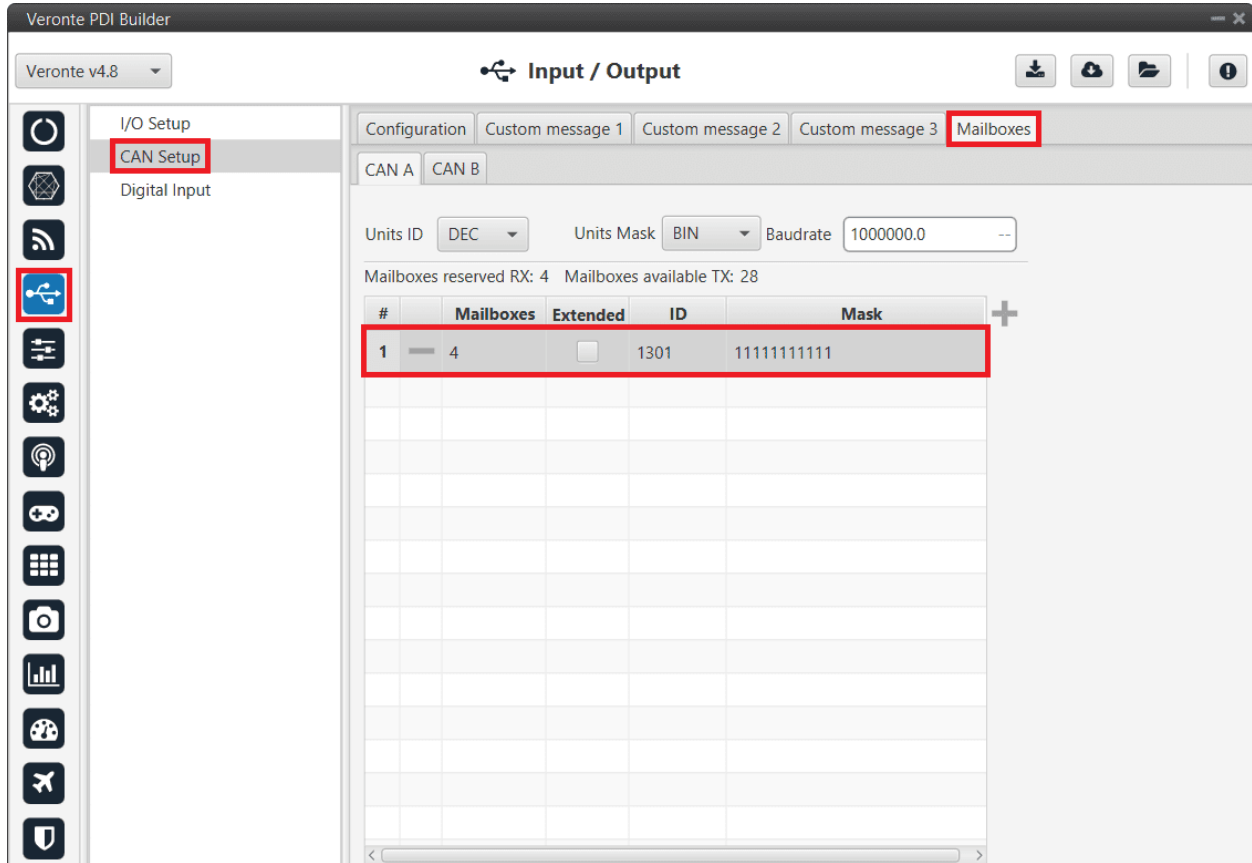


Fig. 114: 1x - Mailboxes configuration

CEX PDI Builder side

Note: This part is already built for CEX default configuration, but the user can check it.

- Go to Input/Output menu → CAN I/O section → **Configuration tab**.

Connect a **CAN Input filter** with the right CAN Address (**CAN ID 1302**) to **CAN to Serial 1**.

In addition, connect **Serial to CAN 1** with the right CAN Address (**CAN ID 1301**) to a **CAN Output filter** port:

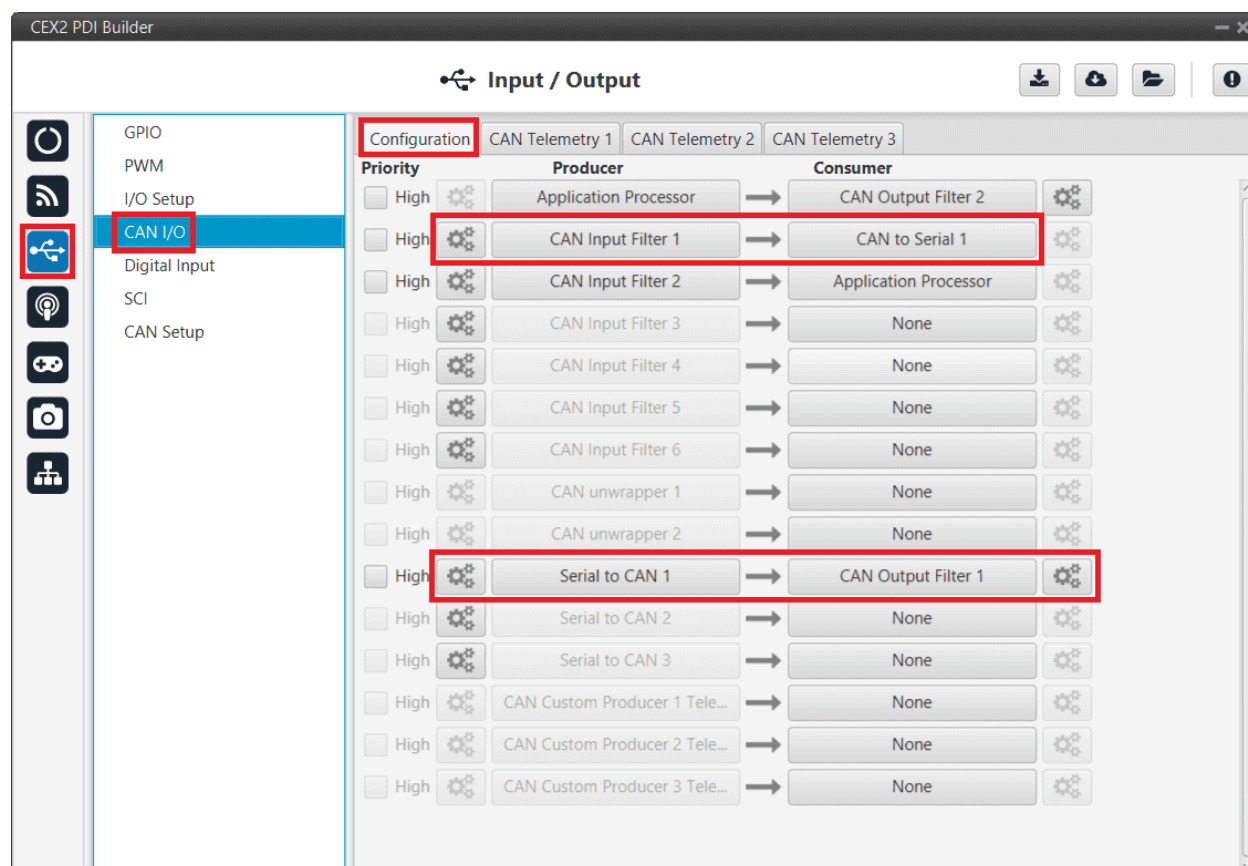


Fig. 115: CEX - CAN I/O

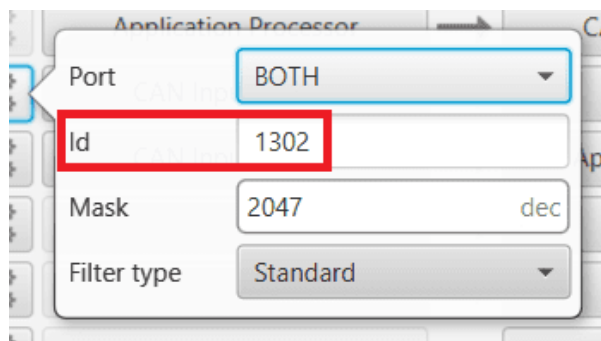


Fig. 116: CEX - CAN I/O - Input filter configuration

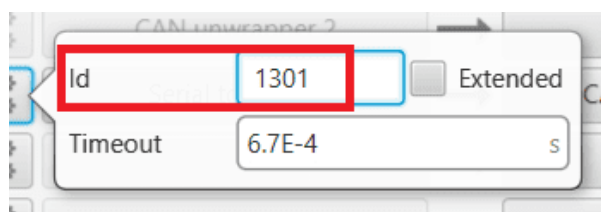


Fig. 117: CEX - CAN I/O - Serial to CAN configuration

7. Go to Input/Output menu → **I/O Setup** section.

Connect **CAN to Serial 1** to any **Commgr Port 1** in CEX.

In addition, connect **Commgr Port 1** to **Serial to CAN 1** consumer:

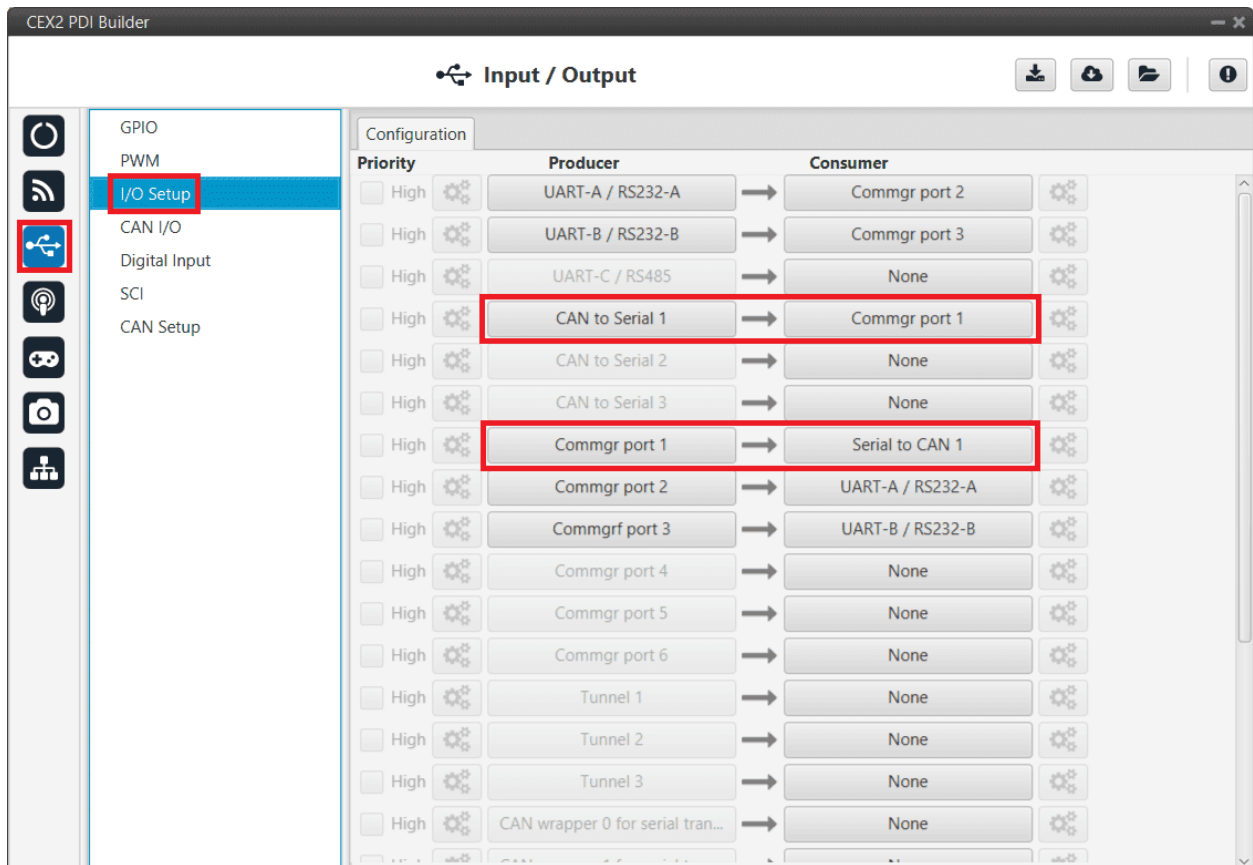


Fig. 118: CEX - I/O Setup

8. Go to Input/Output menu → **CAN Setup** section.

Finally, configure the reception **mailbox with ID 1302**, assign at least 4 mailboxes:

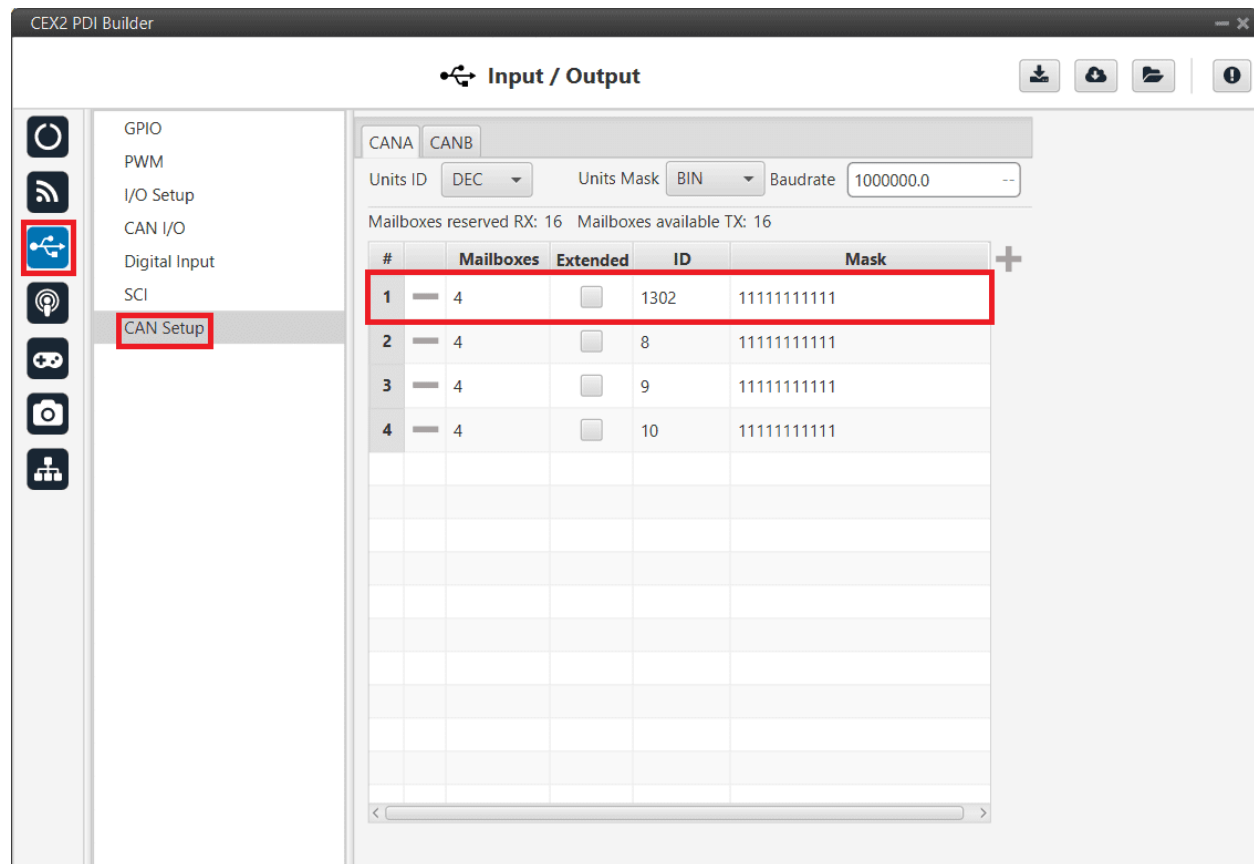


Fig. 119: CEX - CAN Seup (Mailboxes) configuration

3.5.6.3 MC01

In order to communicate a Veronte Autopilot 1x with a MC01 via CAN, the following connection is required:

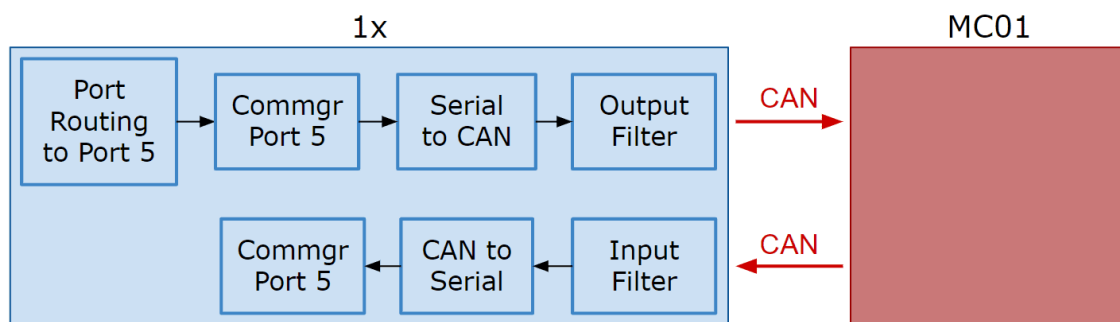


Fig. 120: Communication diagram 1x - MC01

The following steps explain how to configure the communication between a 1x Autopilot and a MC01.

MC01 PDI Builder side

- By default, MC01 is configured with a connection Serial to CAN, with the following **Standard** CAN IDs:
 - Tx CAN Id: 1301

- Rx CAN Id: 1302

1x PDI Builder side

2. Go to Communications menu → **Ports section**.

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).

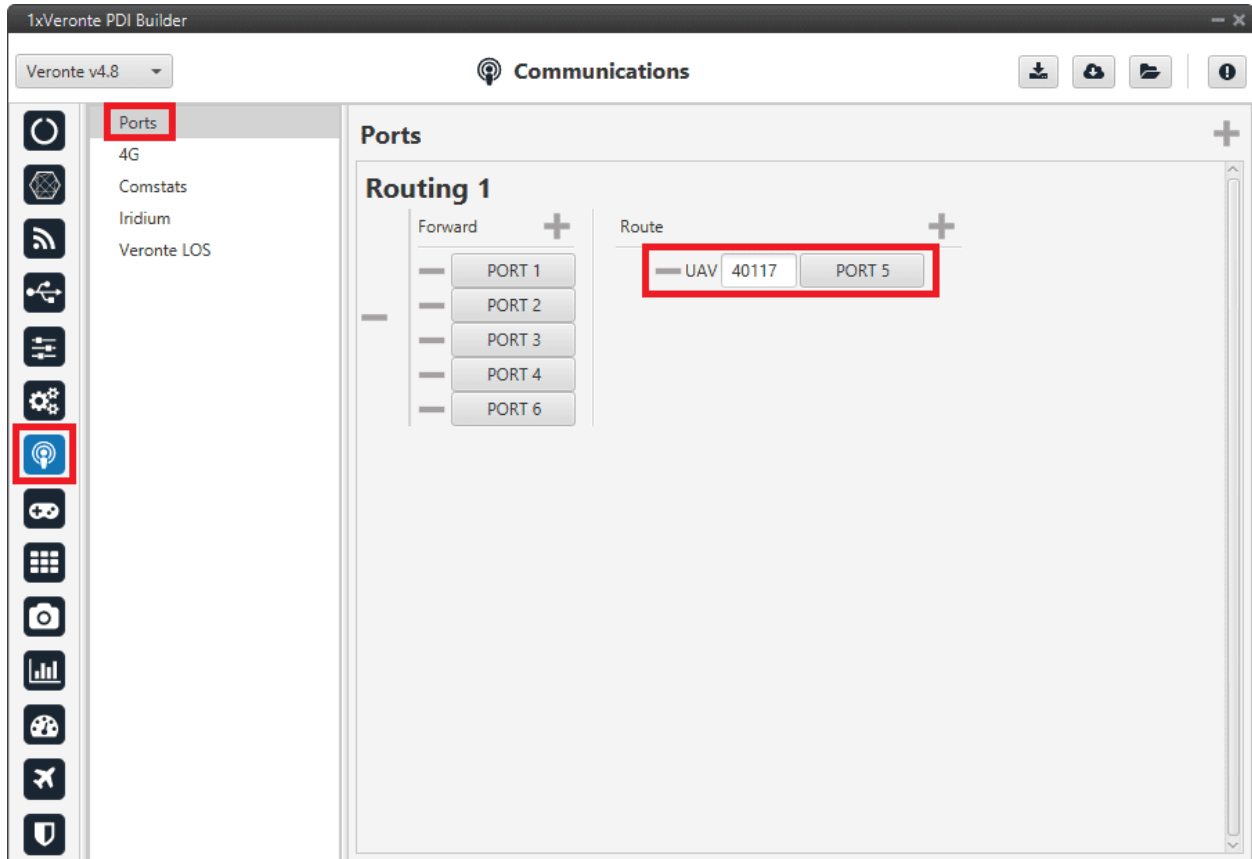


Fig. 121: Routing configuration

3. Go to Input/Output menu → **I/O Setup section**.

Connect the **Commgr Port 5** to the **Serial to CAN 1**.

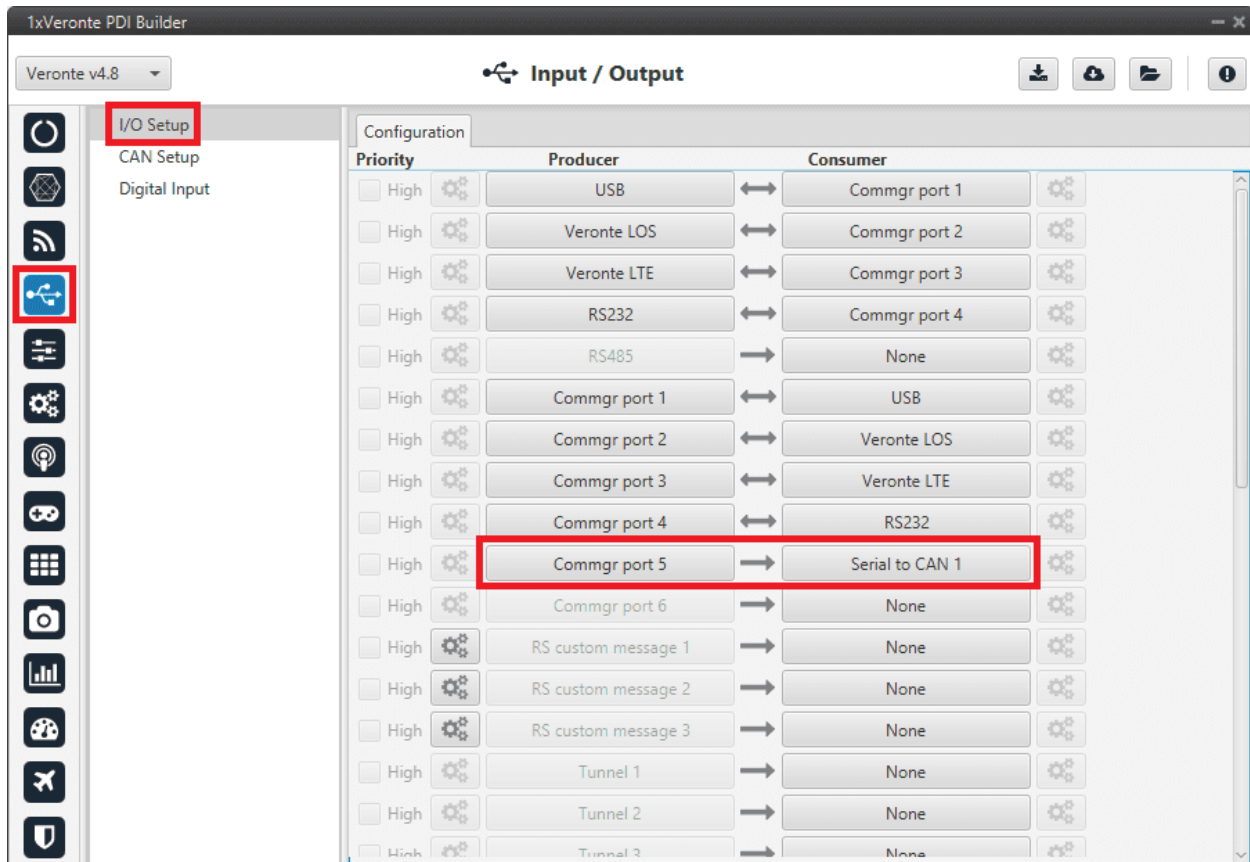


Fig. 122: I/O Setup - Serial to CAN

Then, connect **CAN to Serial 1** to **Commgr Port 5**:

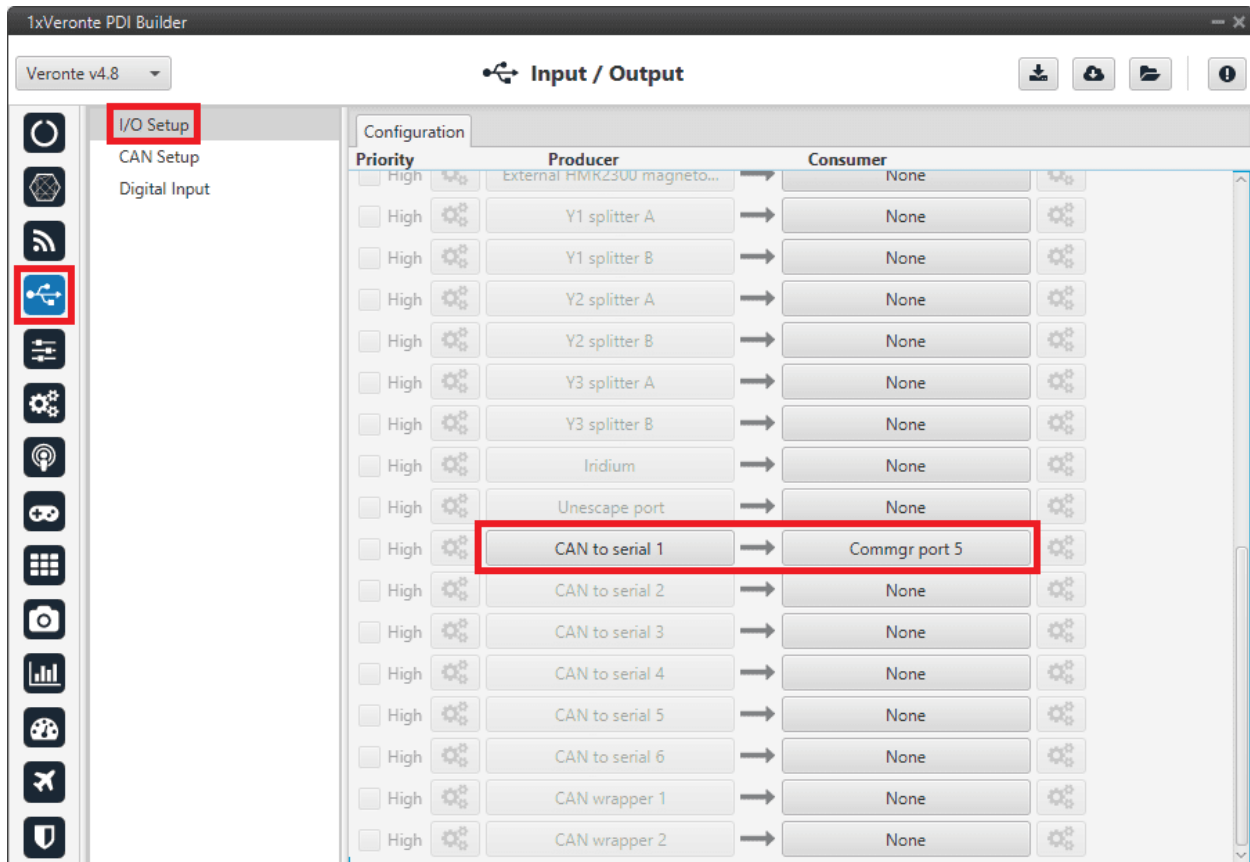


Fig. 123: I/O Setup - CAN to Serial

- Go to Input/Output menu → CAN Setup section → **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**:

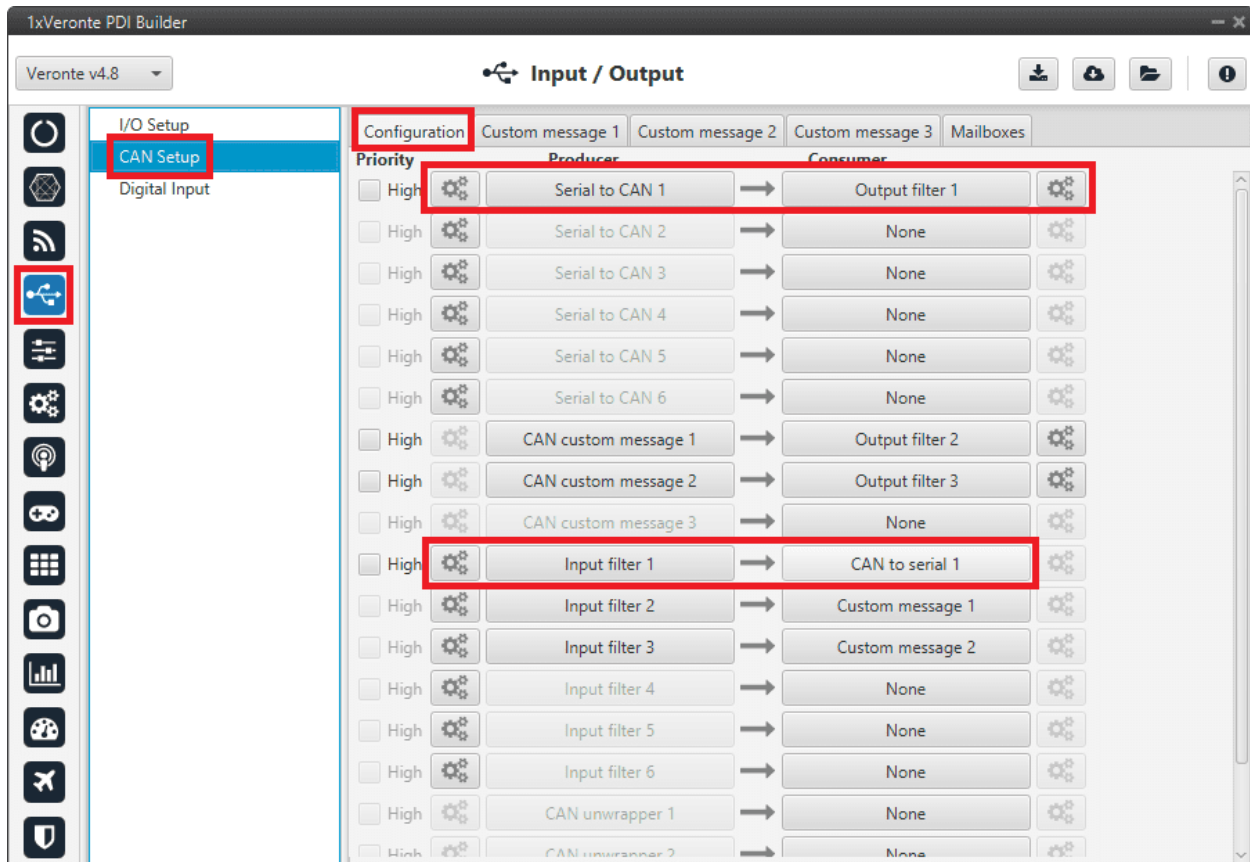


Fig. 124: CAN Setup

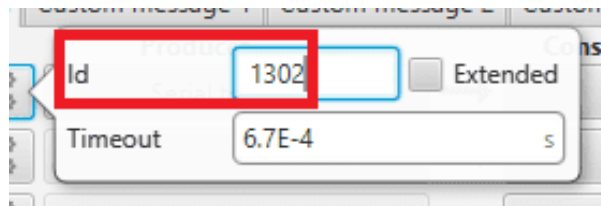


Fig. 125: CAN Setup - Serial to CAN configuration

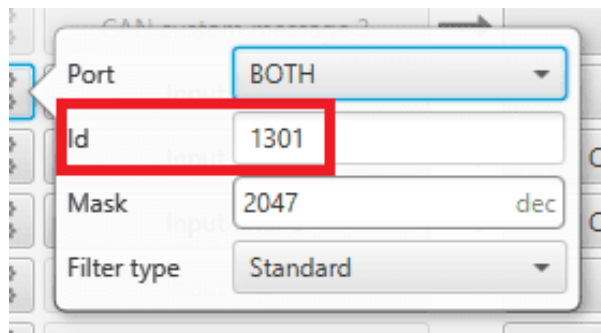


Fig. 126: CAN Setup - Input filter configuration

5. Go to Input/Output menu → CAN Setup section → **Mailboxes tab**.

Finally, configure the reception **mailbox with ID 1301**, assign at least 1 mailbox:

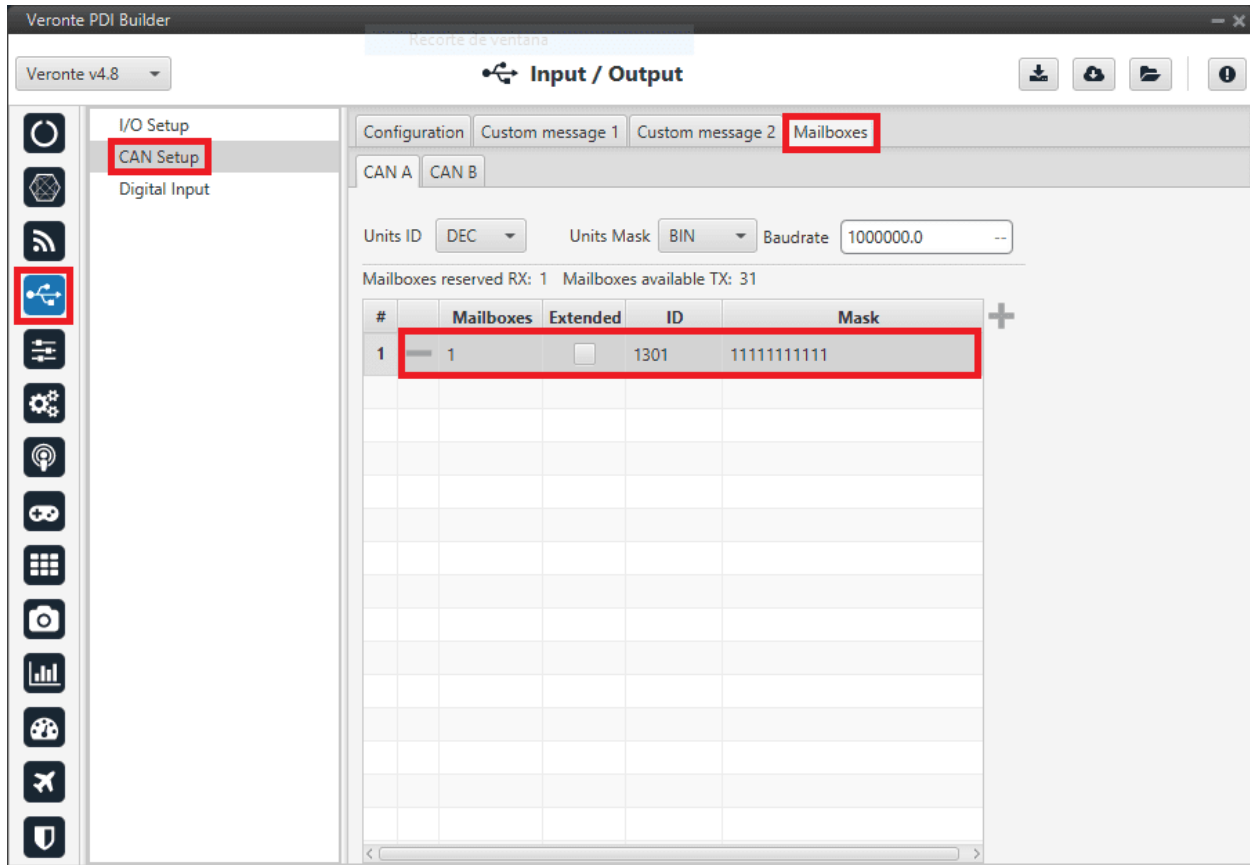


Fig. 127: Mailboxes configuration

3.5.6.4 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** in the *General case - PPM stick integration* section of this manual.

In the **step 1** of that explanation, there is already a transmitter configured with the required configuration of the VSE, users will find it as **Brand**: Embention and **Model**: Stick Expander.

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 Communications menu → **Veronte LOS** section.

Set the **Baudrate** on Veronte LOS to **9600**.

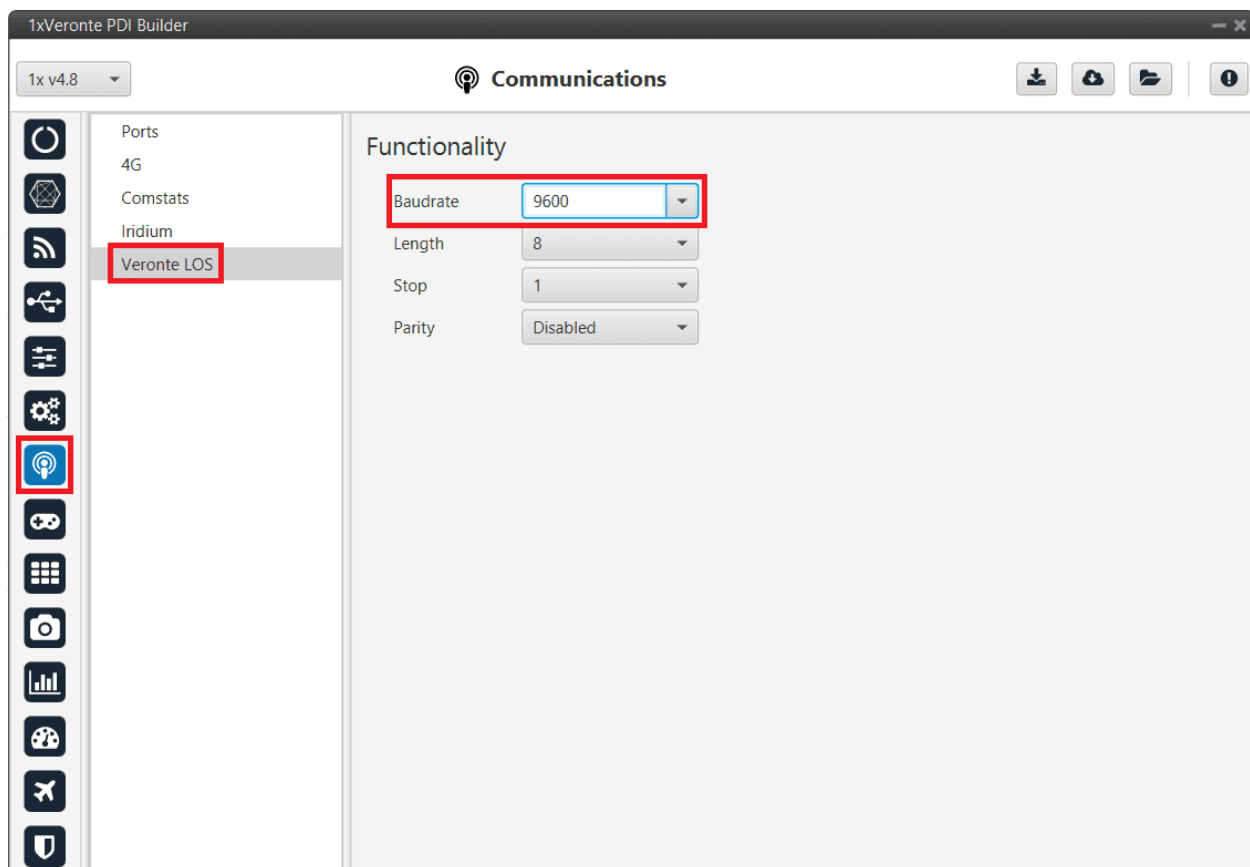


Fig. 1: Veronte LOS baudrate

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:

In 1x PDI Builder

1. Go to Input/Output menu → **I/O Setup section**.

Connect the **RS custom message producer** (where the message is configured) to a **Tunnel** with **App2** address. In this case the message is configured in the *RS custom message 1* producer.

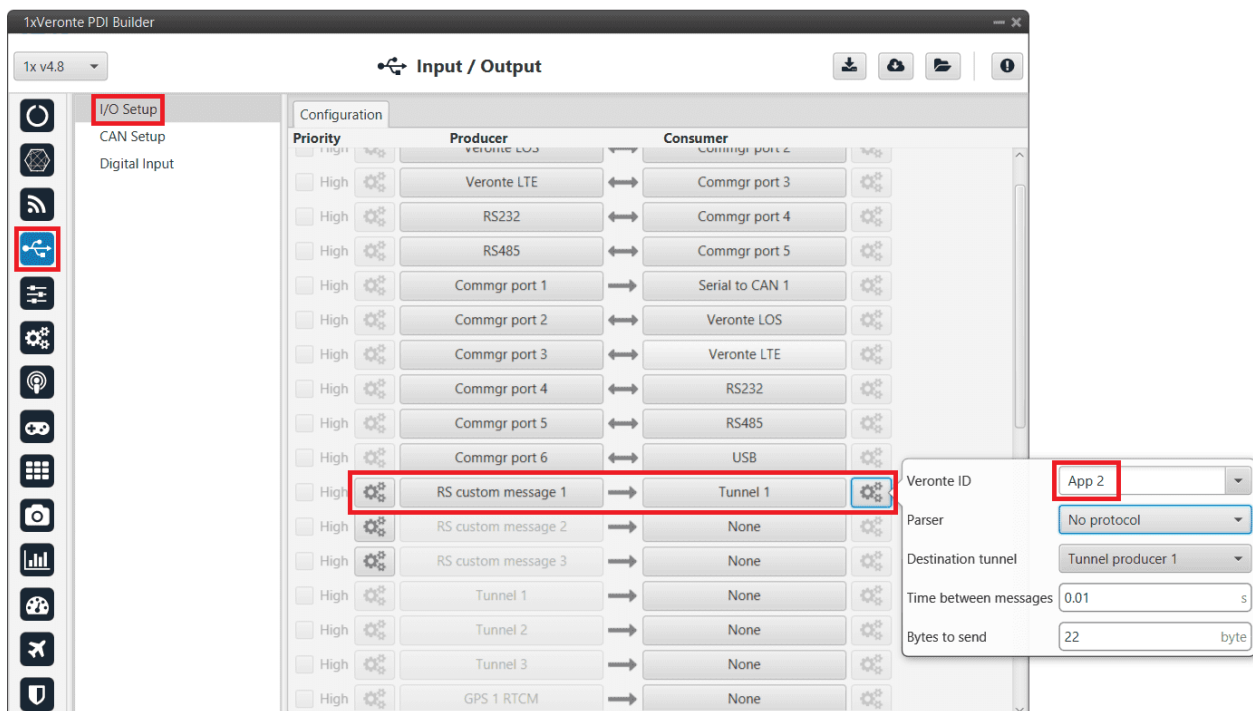


Fig. 2: RS custom message → Tunnel

In 1x PDI Calibration

2. Go to **Terminal tab**.

Click on **Agree**:

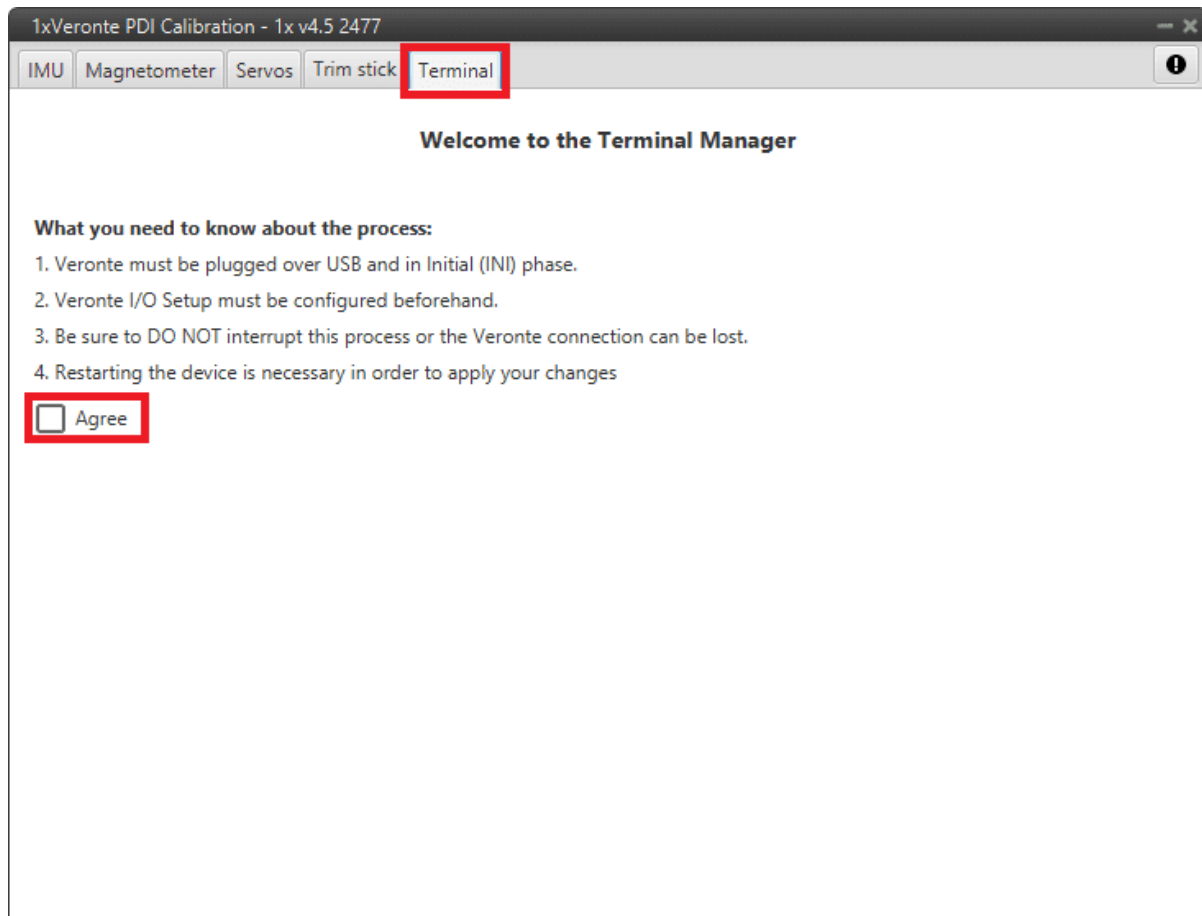


Fig. 3: Terminal tab

3. Next, select the **Tunnel 1** (this is the one that has been configured in **1x PDI Builder**) and click on **Launch**:

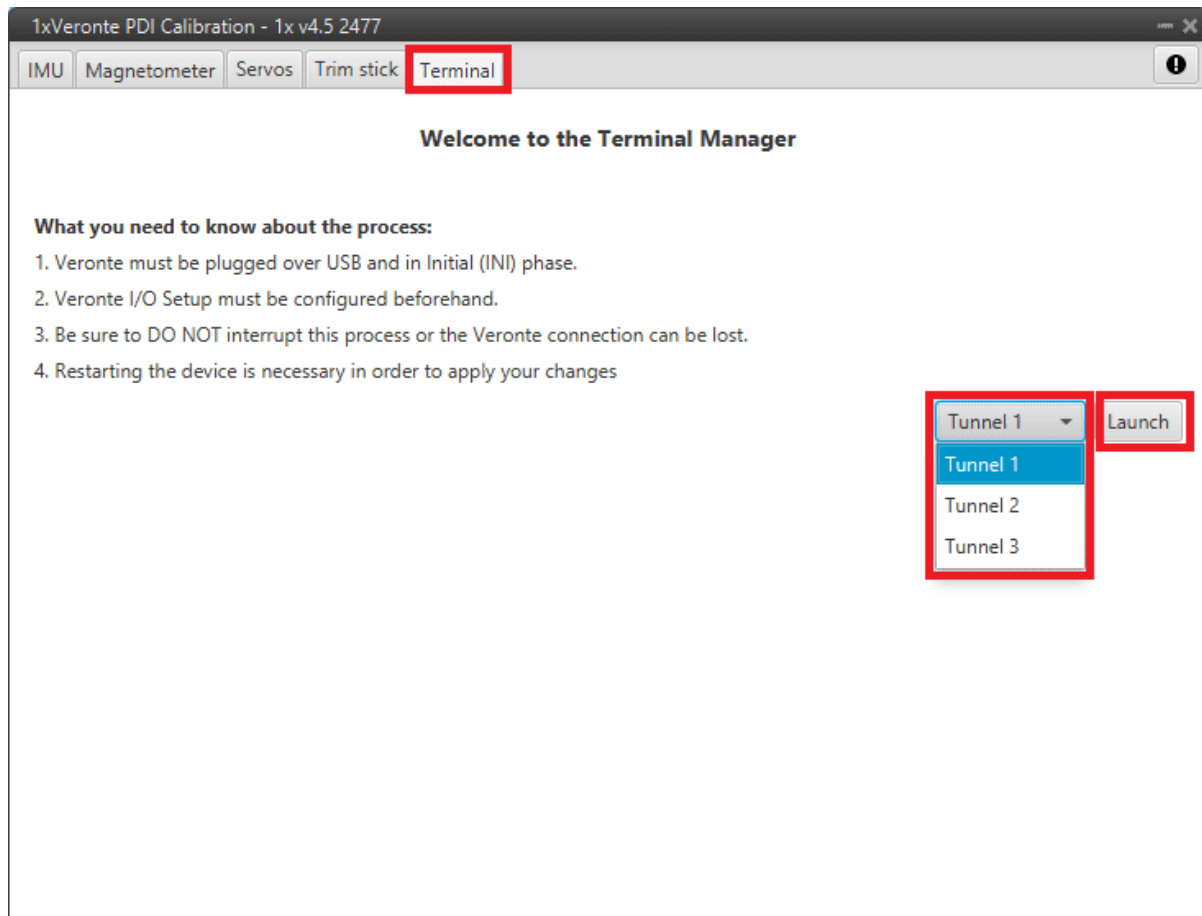


Fig. 4: Terminal tab - Tunnel selected

The tunnel console should open and the user will be able to view the message being sent:

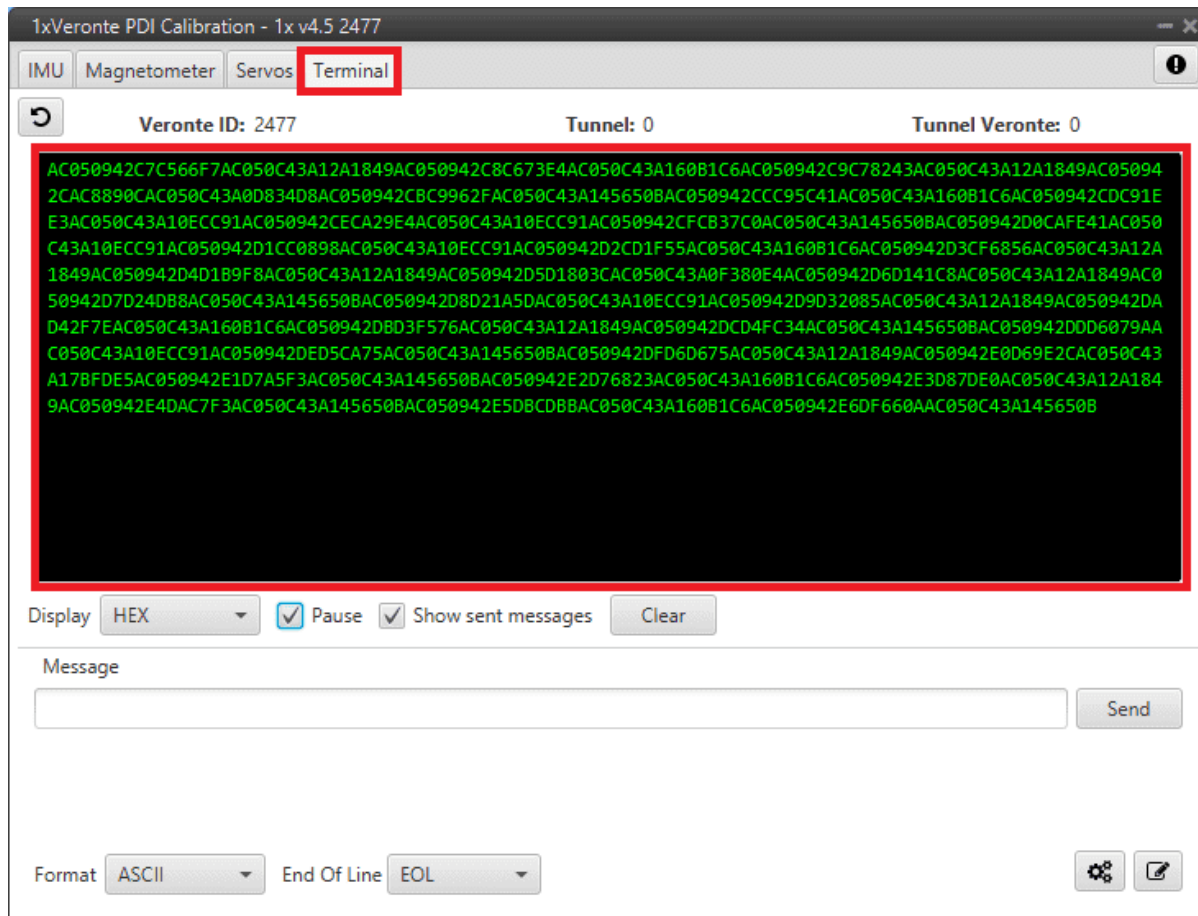


Fig. 5: Tunnel console

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 maintenance 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.

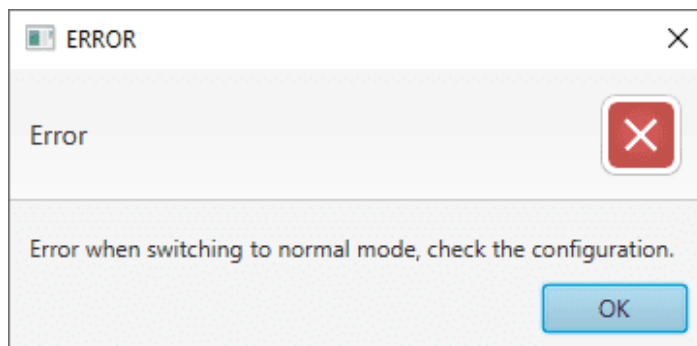


Fig. 7: Error message

Therefore, Veronte Autopilot 1x will be in ‘Maintenance mode (loaded with errors)’:

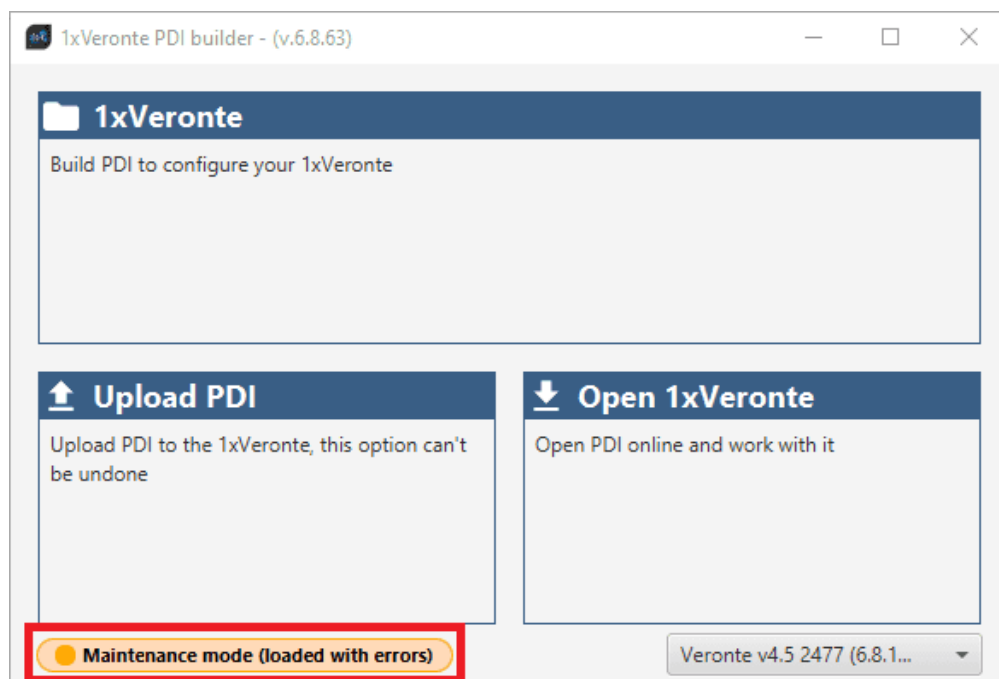


Fig. 8: Maintenance mode (loaded with errors)

To check what the source of the problem is, the user can consult the **Veronte Ops Platform panel**, which will show what the PDI Error is. For more information on this panel, see [Platform panel](#) section of the **Veronte Ops** user manual.

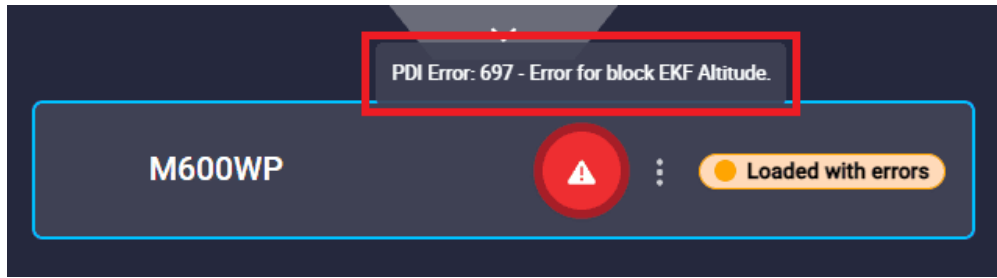


Fig. 9: PDI Error - Veronte Ops

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 Radios paired but 1x air unit not showing

If the radios of both Autopilots 1x, air and ground unit, are paired but the air unit does not appear connected in **Veronte Link**, check the **Ports** configuration on the 1x **ground unit**. To do this:

Go to Communications menu → **Ports section**, and it should be similar to the configuration shown in the figure below:

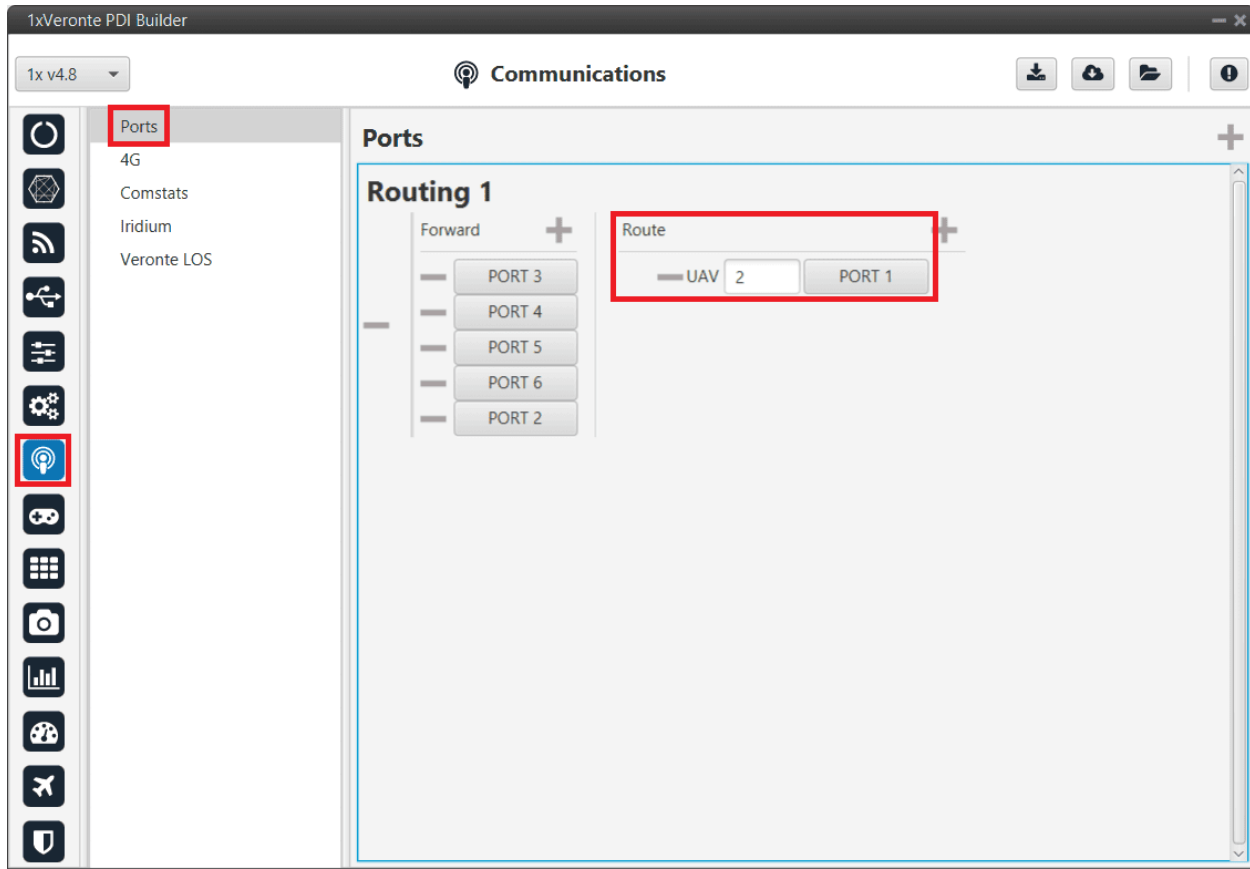


Fig. 10: 1x ground unit - Ports configuration

4.6 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**.

4.7 Trajectory Overshoot

If the user observes significant **meandering** 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:

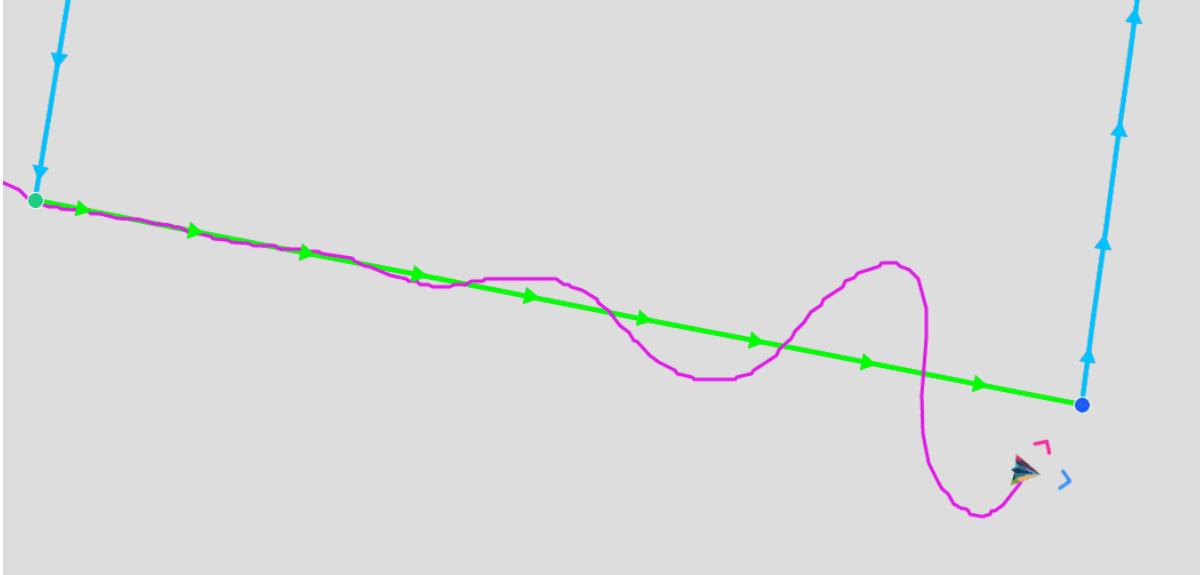


Fig. 11: Trajectory overshoot

4.8 Unstable communication with CEX/MEX

If communication with a CEX/MEX appears to be OK, but suddenly fails, i.e. there is unstable communication between the devices, it may be because **Serial to CAN** and **CAN to Serial** communications have not been marked as **High priority**. In order to do this:

Go to Input/Output menu → CAN Setup section → **Configuration tab**, and mark **Serial to CAN** and **CAN to Serial** communications as **High priority** (with the Priority checkbox):

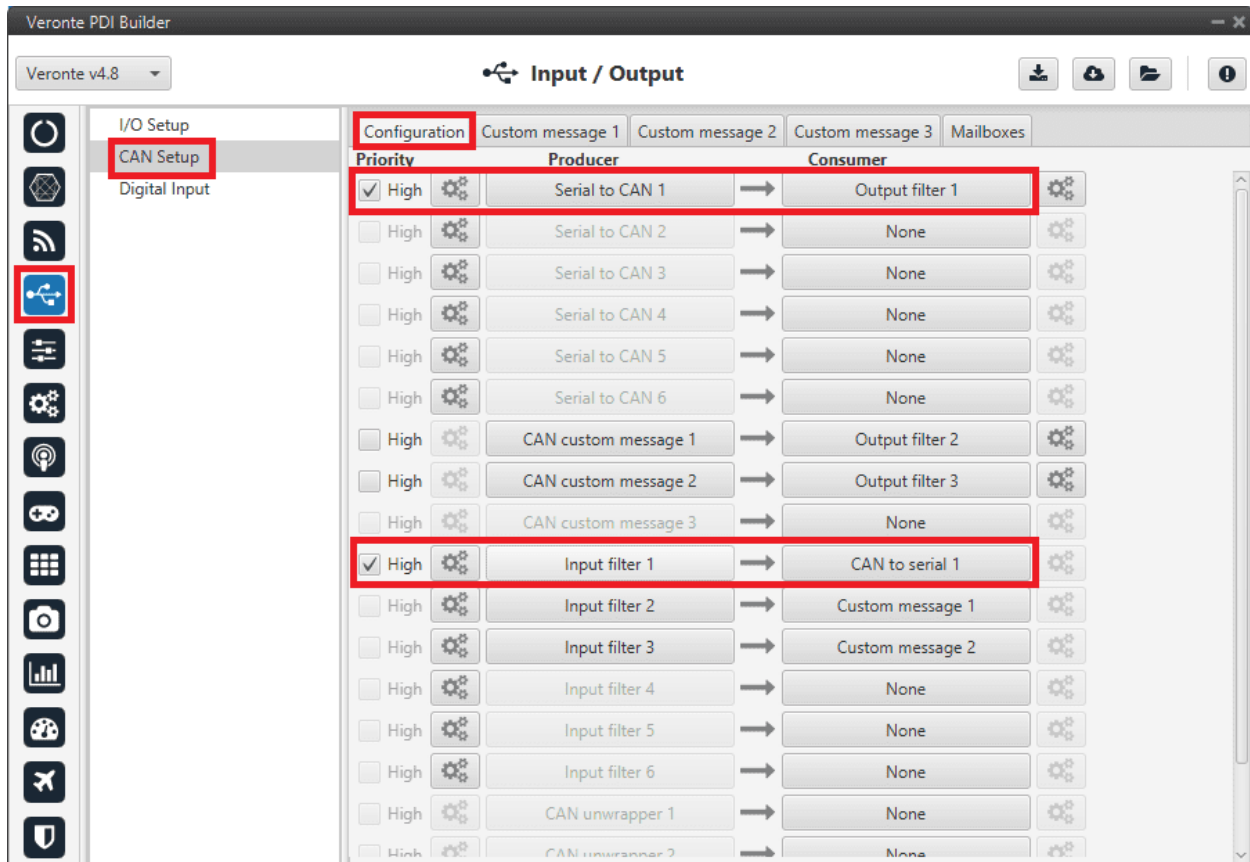


Fig. 12: CAN Setup - High priority

5.1 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 the user is not sure what he/she is doing.