Software in the Loop (SIL)

Embention

Jan 30, 2023

CONTENTS

1 Introduction **1**
- 1.1 Requisites . . . 1

2 Basic Package **3**

3 Dealing with PDI files **5**
- 3.1 Before Pipe v.6.6 . . . 5
- 3.2 After Pipe v.6.6 (Included) - PDI builder . . . 7

4 Autopilot Simulation **9**

5 Sensors simulation **15**
- 5.1 Environment . . . 15
- 5.2 Static Pressure . . . 17
  - 5.2.1 Constant value . . . 18
  - 5.2.2 Step . . . 18
  - 5.2.3 Variable pressure . . . 19
- 5.3 Dynamic Pressure . . . 20
- 5.4 Inertial Measurement Unit . . . 20
- 5.5 Magnetometer . . . 24
- 5.6 GNSS . . . 25
- 5.7 Analog to Digital Converter Port . . . 28
- 5.8 Serial communications . . . 30
  - 5.8.1 EXAMPLE: Sending a rs-232 message . . . 30

6 Monitoring Telemetry **33**

7 Connecting SIL & Veronte Pipe **37**
- 7.1 Pipe v6.6 and higher . . . 39

8 Simulation **41**
- 8.1 Complete Simulation . . . 41
- 8.2 Quadcopter Example . . . 42

CHAPTER

ONE

INTRODUCTION

A software in the loop simulation consists of a Simulink model that simulates the behaviour of the system formed by the autopilot and a vehicle, without having the physical devices connected to the computer, in contrast to the HIL which has both the autopilot and (optionally) vehicle connected to the PC. This option has several advantages when it is compared with a HIL setup:

- Complete simulations without any hardware.
- Possibility of using your own vehicle model: user can define the dynamics of his vehicule (with the desired complexity) without using external programs, like Plane Maker.
- Possibility of simulating differents kinds of sensors even if they are not fitted in Veronte. All you need is the raw sensor reading.
- All results can be exported/visualized to MATLAB workspace simultaneously.
- Veronte Block runs faster than real time, allowing the user to execute a series of simulations in a short time. This feature depends on the complexity of the model and the capability of the computer where the simulation is running.
- Light computational load.

1.1 Requisites

In order to run a SIL simulation with veronte autopilot the followings programs and toolbox are required:

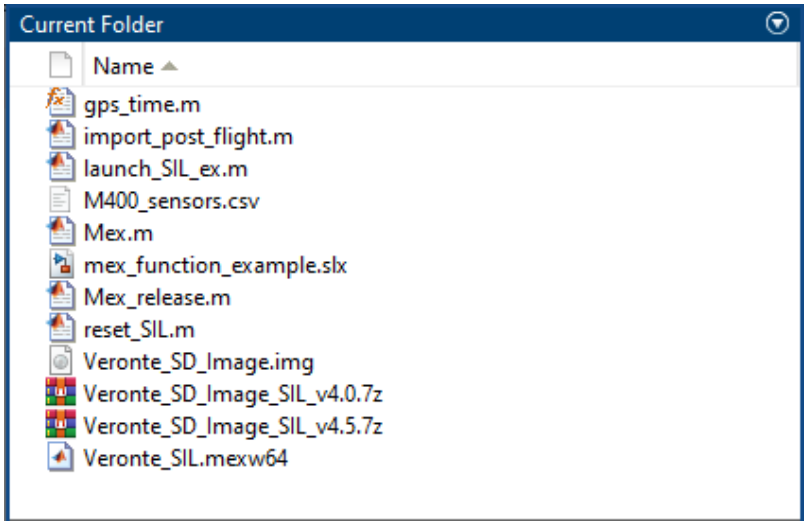
- **MATLAB + Simulink** (basic package).
- **Simulink Real-Time** : this blockset contains useful blocks to be used with buses: UDP/RS232/CAN.
- **Microsoft Visual Studio 2015** (or later) as your MEX compiler. Despite .mex file is already compiled and it works as a black box, some libraries are necessary.

Moreover, the user can be helped by other toolboxes when implementing their model, such as **Aerospace toolbox**: contains sensor blocks, flight instruments and environment blocks.

CHAPTER

TWO

BASIC PACKAGE



Basic package files

The basic SIL package consists of the followings files:

- **gps_time.m**: a matlab function which calculates the GPS/GNSS number of weeks.
- **import_post_flight**: a matlab script for loading an external source of inertial data (IMU). It reads this information from a csv file.
- **mex_function_example.slx**: simulink file. It includes a little example about how using veronte autopilot with sensor readings.
- **Mex.m**: matlab script for compiling veronte code.
- **reset_SIL**: script which must be executed before running a simulation.
- **Veronte_SD_image.img**: contains the autopilot configuration information.
- **Veronte_SIL.mexw64** : mex file. It consists of all the embedded veronte code compiled. It works as a black box. Like Veronte, each mex file has a version. In order to ask for a newer version, contact with sales team at sales@embention.com.

CHAPTER

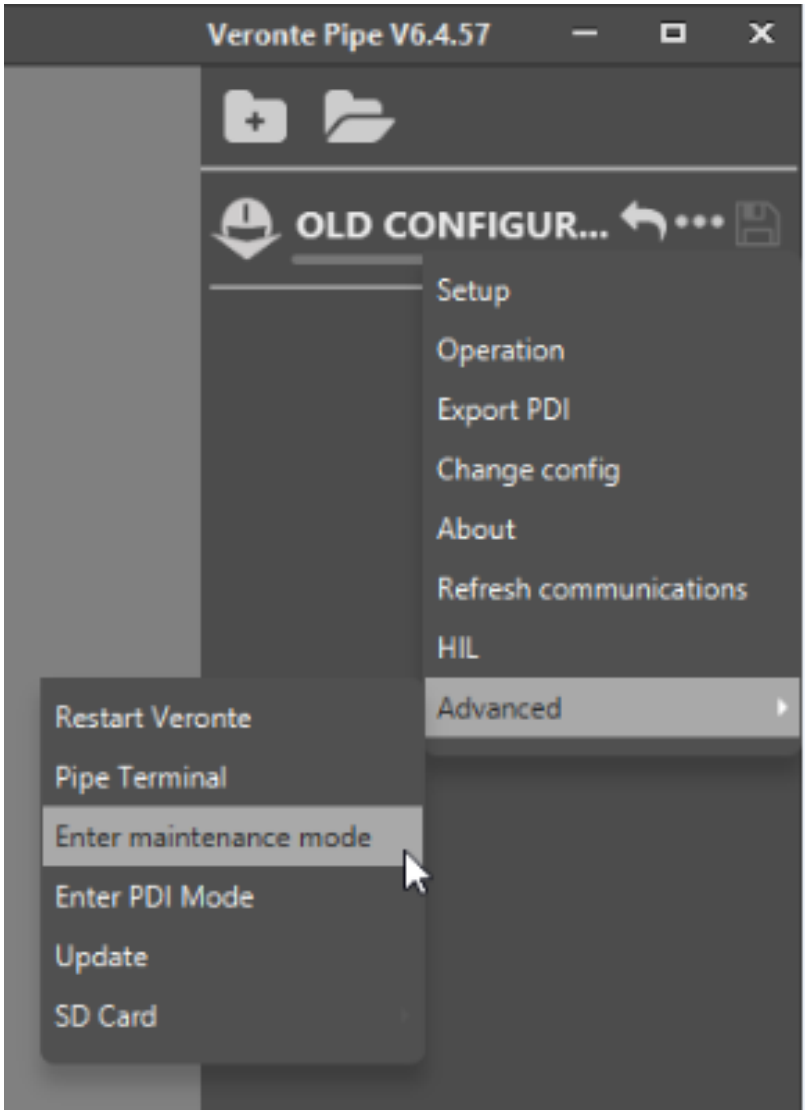
THREE

DEALING WITH PDI FILES

3.1 Before Pipe v.6.6

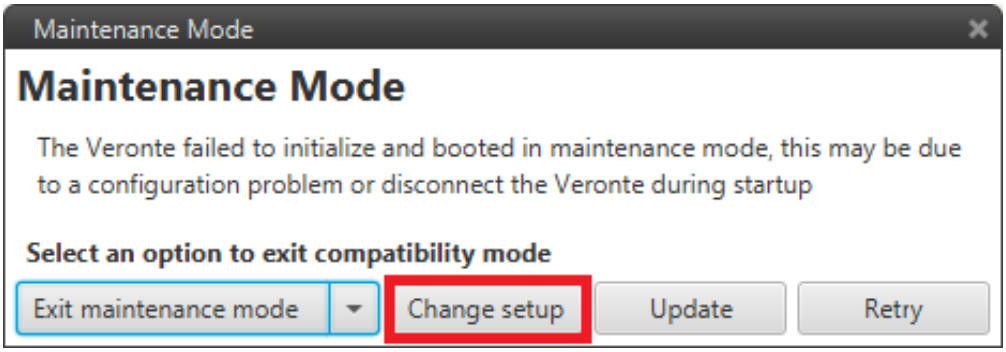
Veronte_SD_image.img contains all the configuration information. PDI are uploaded exactly the same way that would be uploaded to a physical autopilot. Once connected to Veronte Pipe, a virtual autopilot will be detected in safe mode. So the steps to upload/change the PDI configuration are:

1. **Run Simulink model**: to be able to upload new PDI files the virtual autopilot must appear in pipe. So the SIL simulation must start in order to send the proper information.
2. **Enter autopilot in maintenance mode**: then stop the simulation and press run again.



Enter Maintenance mode

3. **Upload a new PDI file**: Once the virtual autopilot appears in maintenance mode, press on change configuration. Select your PDI files and exit from maintenance mode. Stop Simulink.

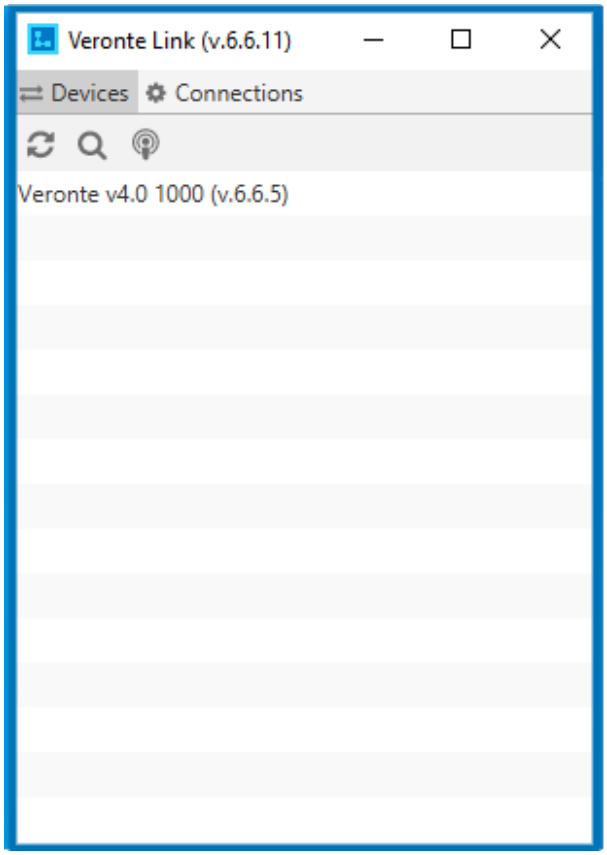


Change setup

4. **Run Simulink** with the correct configuration.

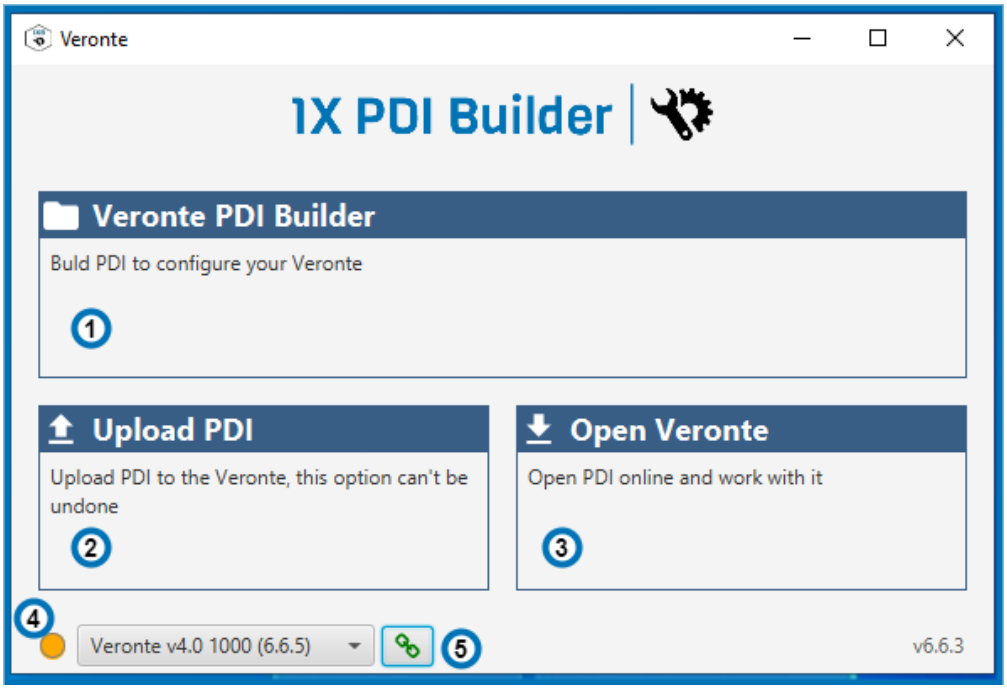
3.2 After Pipe v.6.6 (Included) - PDI builder

1. **Open VeronteLink** and create an Ethernet connection (*Connecting to Pipe*).
2. **Run Simulink Model** to simulate the autopilot information. Check autopilot information appears in VeronteLink.



Veronte Link

3. **Open PDI Builder** and select your autopilot. Then click on link button (5). PDI Builder Interface allows the user the following actions : Create or modify PDI files offline (1), upload PDI configuration to the autopilot (2), open the autopilot configuration to modify it (3), change between maintenance and normal mode (4).



PDI builder

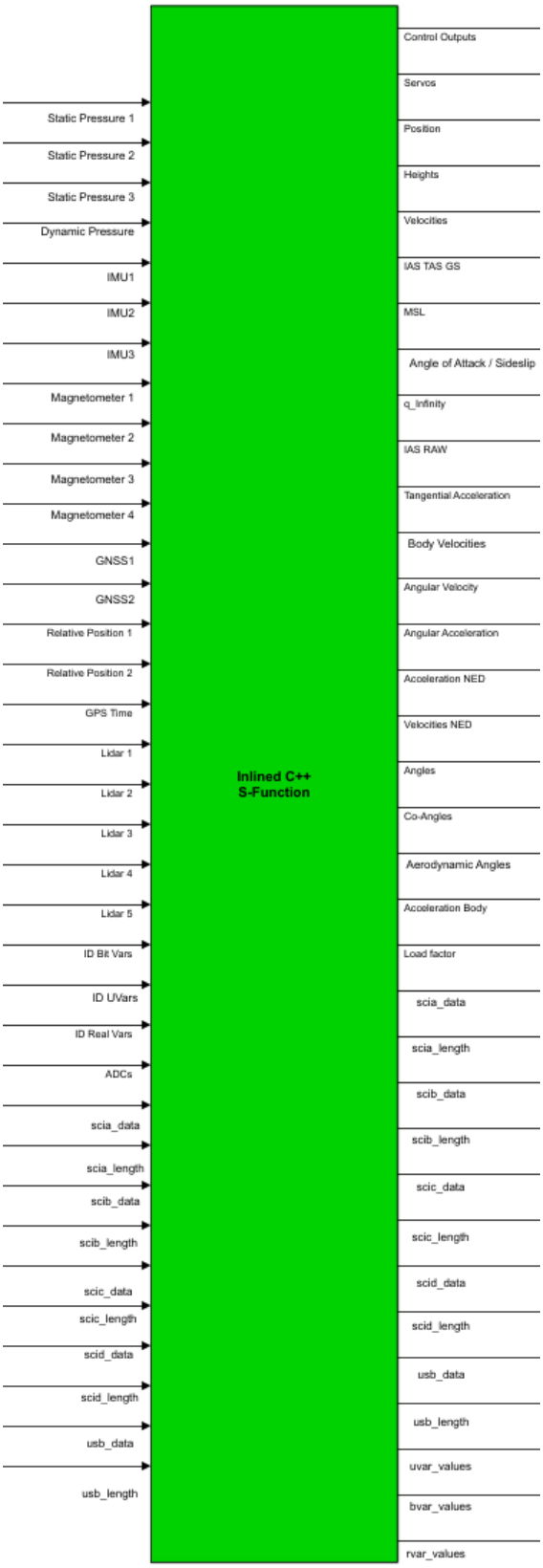
4. If you want to modify a SD image click on (3).
5. If you want to upload your PDI files, change to Maintenance mode (4). Stop the Simulation and press run again. Now the autopilot selected has an orange color. Press Upload PDI (2) and select the new configuration. Exit maintenance mode (4).

CHAPTER

FOUR

AUTOPILOT SIMULATION

The autopilot is implemented in Simulink with an S-Function. This kind of block takes a C, C++, Fortran or even Matlab code, and implements it in a block containing a certain number of inputs and outputs. A typical Veronte s-function is shown below.


Static Pressure 1
Static Pressure 2
Static Pressure 3
Dynamic Pressure
IMU1
IMU2
IMU3
Magnetometer 1
Magnetometer 2
Magnetometer 3
Magnetometer 4
GNSS1
GNSS2
Relative Position 1
Relative Position 2
GPS Time
Lidar 1
Lidar 2
Lidar 3
Lidar 4
Lidar 5
ID Bit Vars
ID UVars
ID Real Vars
ADCs
scia_data
scia_length
scib_data
scib_length
scic_data
scic_length
scid_data
scid_length
usb_data
usb_length
Inlined C++
S-Function
Control Outputs
Servos
Position
Heights
Velocities
IAS TAS GS
MSL
Angle of Attack / Sideslip
q_Infinity
IAS RAW
Tangential Acceleration
Body Velocities
Angular Velocity
Angular Acceleration
Acceleration NED
Velocities NED
Angles
Co-Angles
Aerodynamic Angles
Acceleration Body
Load factor
scia_data
scia_length
scib_data
scib_length
scic_data
scic_length
scid_data
scid_length
usb_data
usb_length
uvar_values
bvar_values
rvar_values

S-Function containing the autopilot embedded code

Inputs are described in the next table:

PIN	Signal Type	Description	Form	Size	Units
1	Input	Static Pressure 1	[pressure_measurement;sensor temperature]	2x1	Pa / K
2	Input	Static Pressure 2	[pressure_measurement;sensor temperature]	2x1	Pa / K
3	Input	Static Pressure 3	[pressure_measurement;sensor temperature]	2x1	Pa / K
4	Input	Dynamic Pressure	[pressure_measurement;sensor temperature]	2x1	Pa / K
5	Input	IMU 1	[acc_x;acc_y;acc_z;gyr_x;gyr_y;gyr_z;sensor temperature]	7x1	m/s^2 / m/s / K
6	Input	IMU 2	[acc_x;acc_y;acc_z;gyr_x;gyr_y;gyr_z;sensor temperature]	7x1	m/s^2 / m/s / K
7	Input	IMU 3	[acc_x;acc_y;acc_z;gyr_x;gyr_y;gyr_z;sensor temperature]	7x1	m/s^2 / m/s / K
8	Input	Magnetometer 1	[mag_x;mag_y;mag_z;sensor temperature]	4x1	T
9	Input	Magnetometer 2	[mag_x;mag_y;mag_z;sensor temperature]	4x1	T
10	Input	Magnetometer 3	[mag_x;mag_y;mag_z;sensor temperature]	4x1	T
11	Input	Magnetometer 4	[mag_x;mag_y;mag_z;sensor temperature]	4x1	T
12	Input	GNSS 1	[1;3;lon;lat;alt;hr_accu;vt_accu;v_n;v_e;v_d;v_accu]	10x1	deg_10^7 / cm / mm/s
13	Input	GNSS 2	[1;3;lon;lat;alt;hr_accu;vt_accu;v_n;v_e;v_d;v_accu]	10x1	deg_10^7 / cm / mm/s
14	Input	Relative Position 1	[1;x_rel;y_rel;z_rel;d_x;d_y;d_z;x_accu;y_accu;z_accu]	10x1	cm / mm * 10^-1
15	Input	Relative Position 2	[1;x_rel;y_rel;z_rel;d_x;d_y;d_z;x_accu;y_accu;z_accu]	10x1	cm / mm * 10^-1
16	Input	GPS Time	[week_number;seconds_of_week]	2x1	• / s
17	Input	Lidar 1	[lidar_measurement]	1x1	cm
18	Input	Lidar 2	[lidar_measurement]	1x1	cm
19	Input	Lidar 3	[lidar_measurement]	1x1	cm
20	Input	Lidar 4	[lidar_measurement]	1x1	cm
21	Input	Lidar 5	[lidar_measurement]	1x1	cm
22	Input	ID Bit Var	[Var_IDs]	50x1	m
23	Input	ID Unsigned Var	[Var_IDs]	50x1	m
24	Input	ID Real Var	[Var_IDs]	50x1	m
25	Input	ADCs	[adc(1-17)]	17x1	•

continues on next page

Table 1 – continued from previous page

PIN	Signal Type	Description	Form	Size	Units
26	Input	SCIA Data	[serial_data]	1024x1	•
27	Input	SCIA Length	[serial_length]	1x1	•
28	Input	SCIB Data	[serial_data]	1024x1	•
29	Input	SCIB Length	[serial_length]	1x1	•
30	Input	SCIC Data	[serial_data]	1024x1	•
31	Input	SCIC Length	[serial_length]	1x1	•
32	Input	SCID Data	[serial_data]	1024x1	•
33	Input	SCID Length	[serial_length]	1x1	•
34	Input	USB Data	[serial_data]	1024x1	•
35	Input	USB Length	[serial_length]	1x1	•

Outputs are the following:

PIN	Signal Type	Description	Form | Size		
1	Output	Control Outputs	[control_outputs(1-20)] | 20x1		•
2	Output	Servo Values	[servos(1-32)]	32x1	•
3	Output	Position	[lat;lon;alt]	3x1	rad / m
4	Output	Heights	[msl,agl]	2x1	m
5	Output	Velocities	[longitudinal_v;lateral_v;velocity(modulus)]	3x1	m/s
6	Output	IAS TAS GS	[ias,tas,gs]	3x1	m/s
7	Output	MSL	[msl_from_qnh;msl_from_ISA]	2x1	m
8	Output	Angle of Attack / Sideslip	[angle_of_attack;sideslip]	2x1	rad
9	Output	Q_Infinty	[dynamic_pressure]	1x1	Pa
10	Output	IAS RAW	[ias_raw]	1x1	m/s

continues on next page

Table 2 – continued from previous page

PIN	Signal Type	Description	Form | Size		
11	Output	Tangential Acceleration	[tangential_acceleration]	1x1	m/s^2
12	Input	Body Velocities	[lon_v;lat_v;vertical_v]	3x1	m/s
13	Output	Angular Velocities	[roll_rate;pitch_rate;yaw_rate]	3x1	rad/s
14	Output	Angular Acceleration	[acc_z_axis;acc_y_axis;acc_x_axis]	3x1	rad/^2
15	Output	Acceleration NED	[acc_north;acc_east;acc_down]	3x1	m/s^2
16	Output	Velocity NED	[v_north;v_east;v_down])	3x1	m/s
17	Output	Angles	[Yaw;Pitch;Roll])	3x1	rad
18	Output	Co-Angles	[co-Yaw;co-Pitch;co-Roll]	3x1	rad
19	Output	Aerodynamic Angles	[heading,flight_path,bank_angle]	3x1	rad
20	Output	Acceleration Body	[acc_x,acc_y;acc_z]	3x1	m/s^2
21	Output	Load factor	[nx;ny;nz]	3x1	•
22	Output	SCIA Data	[serial_data]	1024x1	•
23	Output	SCIA Length	[serial_length]	1x1	•
24	Output	SCIB Data	[serial_data]	1024x1	•
25	Output	SCIB Length	[serial_length]	1x1	•
26	Output	SCIC Data	[serial_data]	1024x1	•
27	Output	SCIC Length	[serial_length]	1x1	•
28	Output	SCID Data	[serial_data]	1024x1	•
29	Output	SCID Length	[serial_length]	1x1	•
30	Output	USB Data	[serial_data]	1024x1	•

continues on next page

Table 2 – continued from previous page

PIN	Signal Type	Description	Form | Size		
31	Output	USB Length	[serial_length])	1x1	•
32	Output	Unsigned Variables	[selected variables(1-50)]	50x1	•
33	Output	Bit Variables	[selected variables(1-50)]	50x1	•
34	Output	Real Variables	[selected variables(1-50)]	50x1	•

CHAPTER

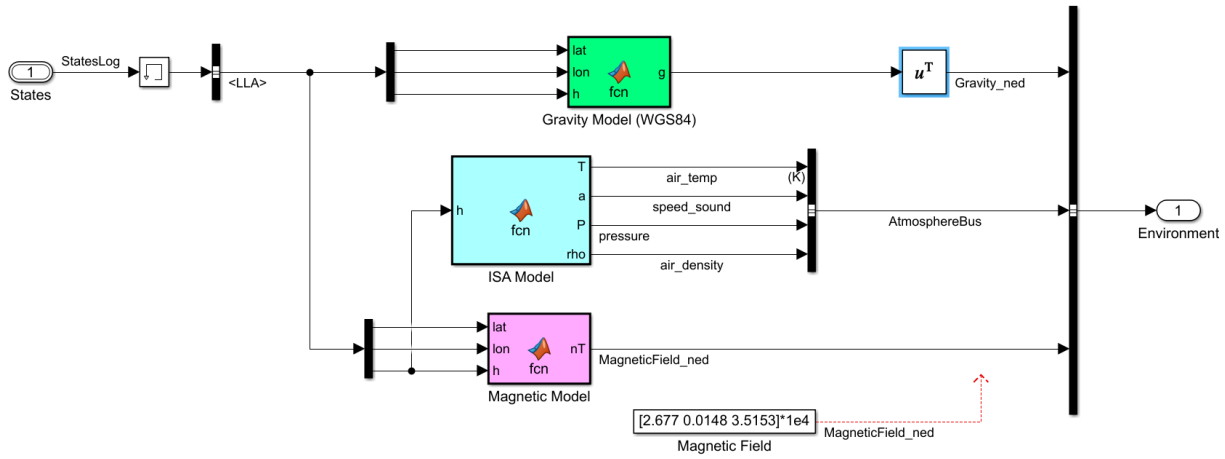
FIVE

SENSORS SIMULATION

5.1 Environment

To simulate properly a model is necessary to take in account that the environment variables changes depending on the uav position. User can choose between a simple and constant model or modify in each step the environment variables according to a complex model.

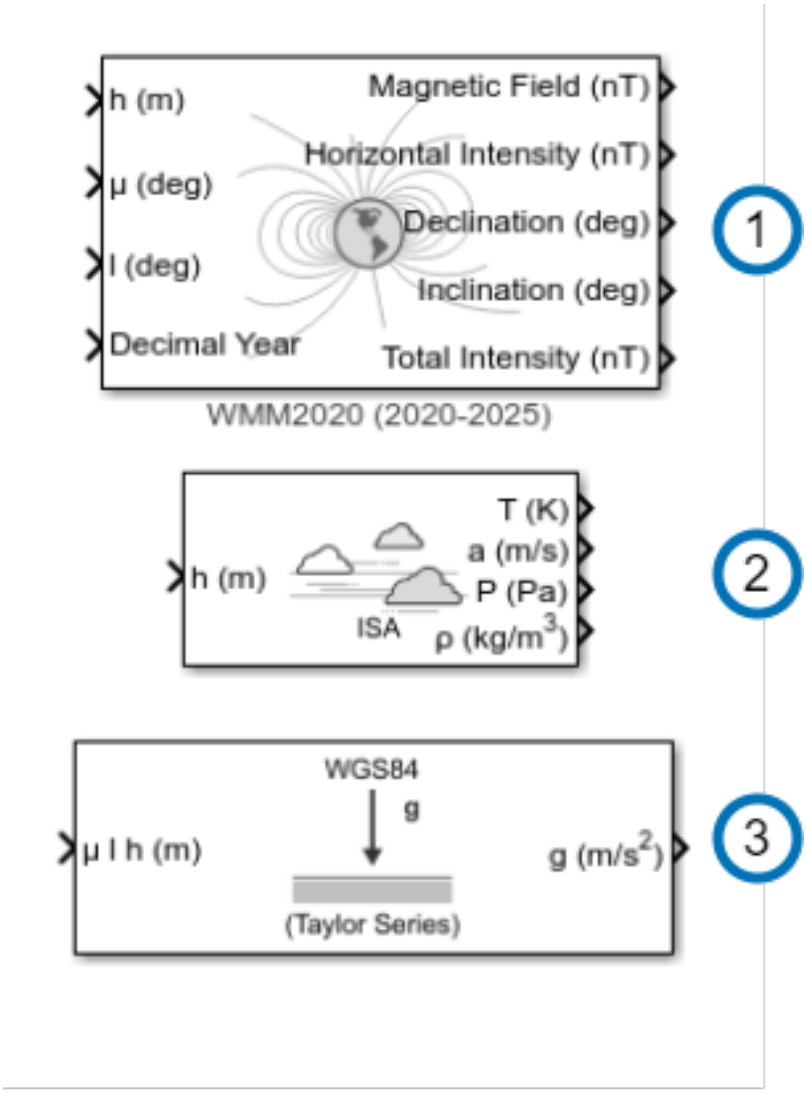
This model should group the atmospherical properties (temperature, pressure, etc.) which change with the altitude (also you can add an offset), the gravity vector as well as the magnetic field which change according to certain coordinates on earth. All this information is required by the S-function or is necessary for a good characterization of the sensors measurements. A basic example is shown below. It is divided into 3 different models (ISA atmosphere model, WGS84 model for gravity vector, and the World Magnetic Model). Each model is included in a user Matlab function whose arguments are the inputs of the block.



Environment block

Instead of creating their own functions, user can employ those that are included in Aerospace Toolbox:

1. World Magnetic Model 2015
2. ISA Atmosphere Model
3. WGS84 Gravity Model

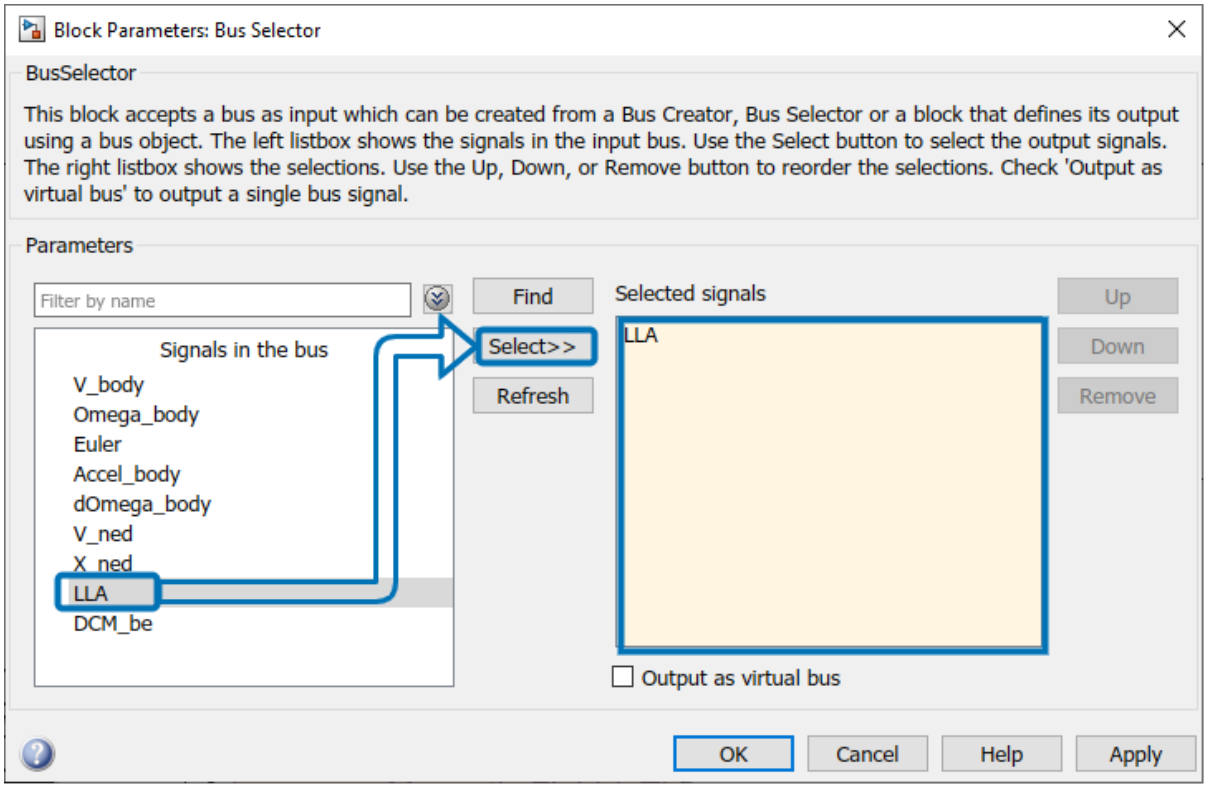


Aerospace blockset functions

The input of this block is the state (in the previous step) of your vehicule. You have to compute this state from a dynamic model whose inputs are the values of the actuators (outputs of the autopilot). These variables (position, velocity, acceleration, etc.) can be group in a vector or a bus. If a vector is chosen then you have to pick the desired variables with a demux block or a selector block. In the case of a bus, the information is separated with a bus selector block.

Bus selector block

For the environment block the only variables required are these shown in the picture below:



Environment input

5.2 Static Pressure

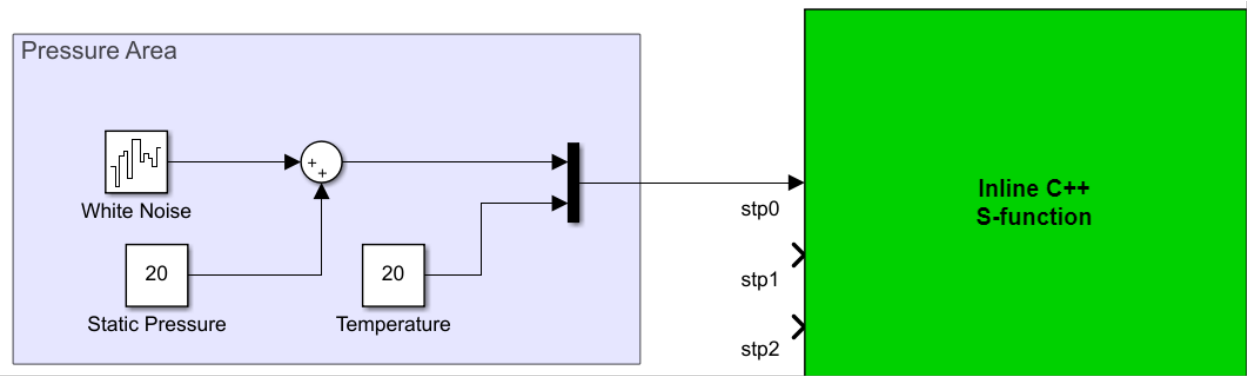
Static Pressure inputs in S-function simulate the real ones in Veronte. The information required consists of raw measurements and the sensor device temperature. The S-function contains 3 ports as the autopilot hardware. Then this information should be used according to the static pressure sensor selected in the configuration.

Normally the same information should feed the 3 ports, althougth you can simulate that one of them is not working properly.

Some examples of how implement the static pressure are shown below:

5.2.1 Constant value

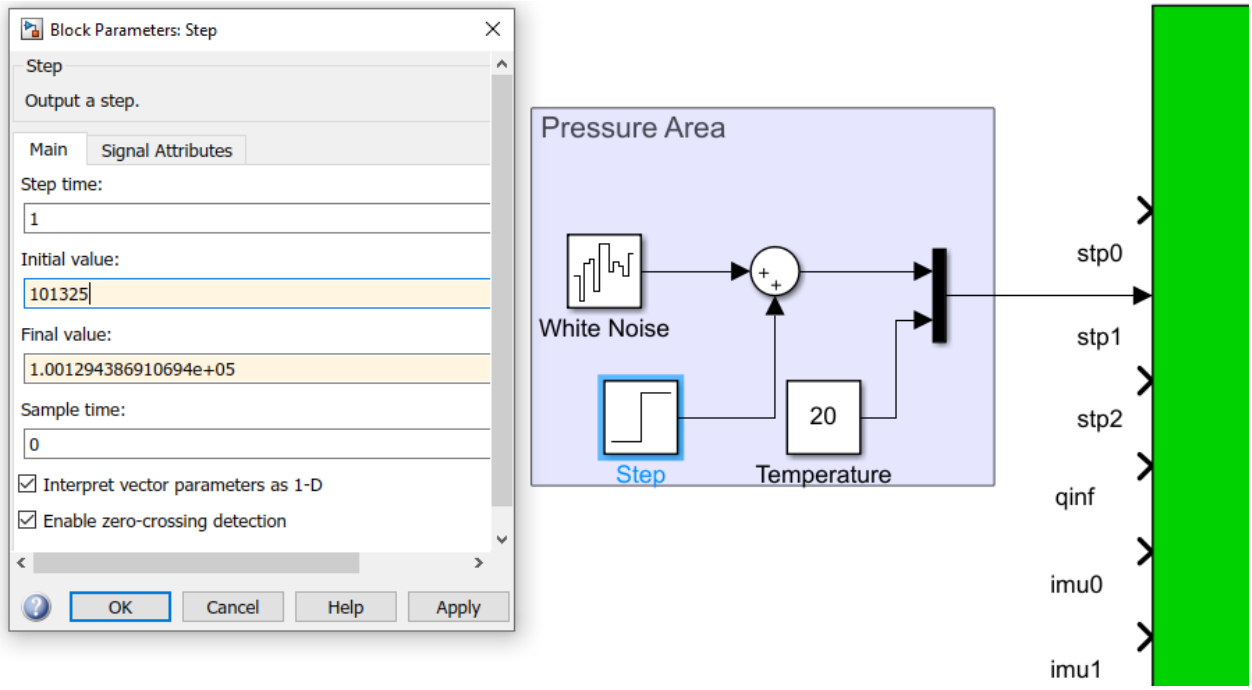
Only a block constant for raw pressure and another for temperature. Also it is possible to add some white noise.



Constant pressure

5.2.2 Step

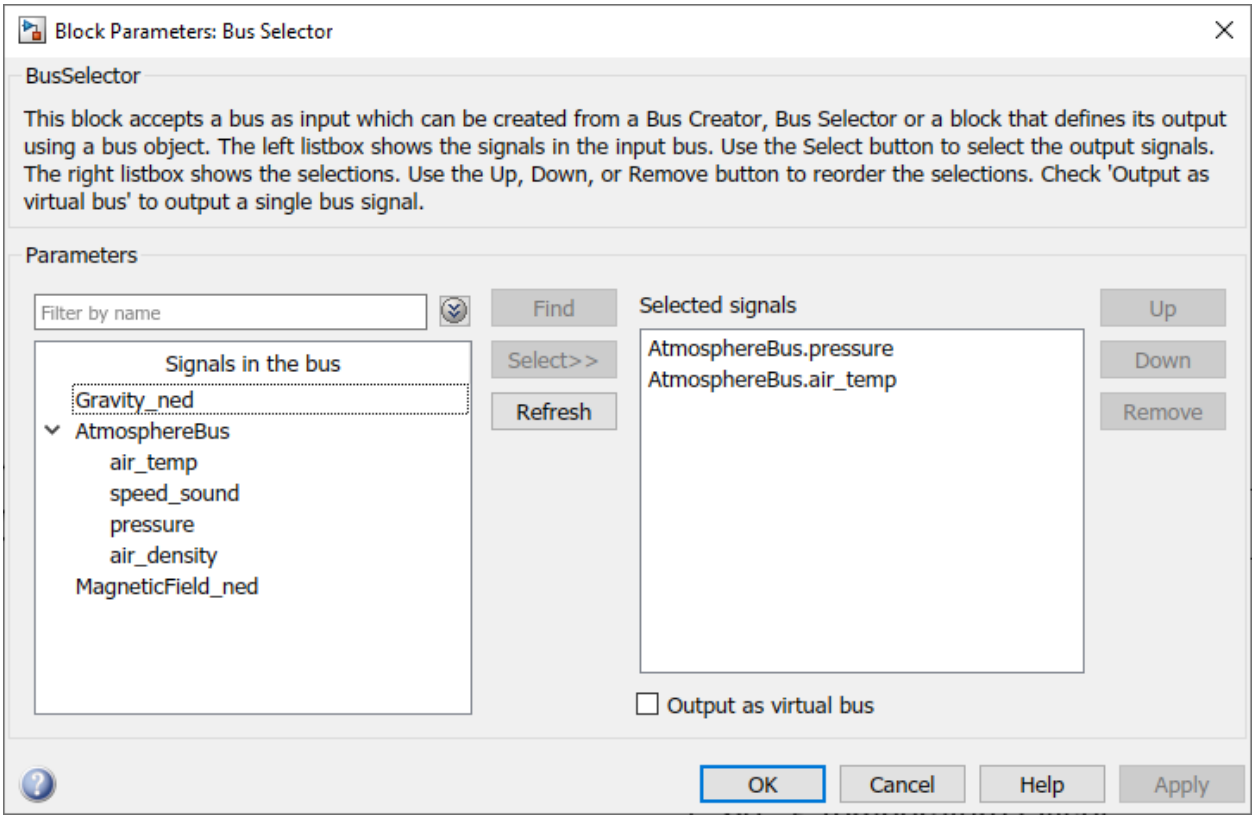
If you want to simulate a leap in pressure measurements you can add a step to the previous configuration. In the example below a difference in 100 meters is represented.



Step input in pressure

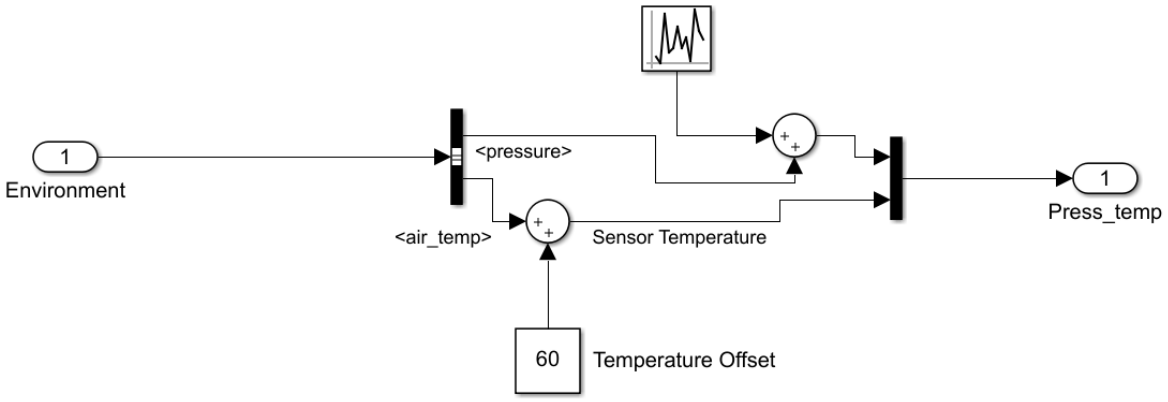
5.2.3 Variable pressure

If you want a more accurate model which modify this value according to vehicle position you need to enter pressure information from the environment block. This block is necessary when user is simulating movement because pressure is an input to the fusion algortihms (Ex.:Kalman filter). You had to select the raw measurement and the temperature from the bus that cointains all the atmosphere properties.



Selecting environment variables

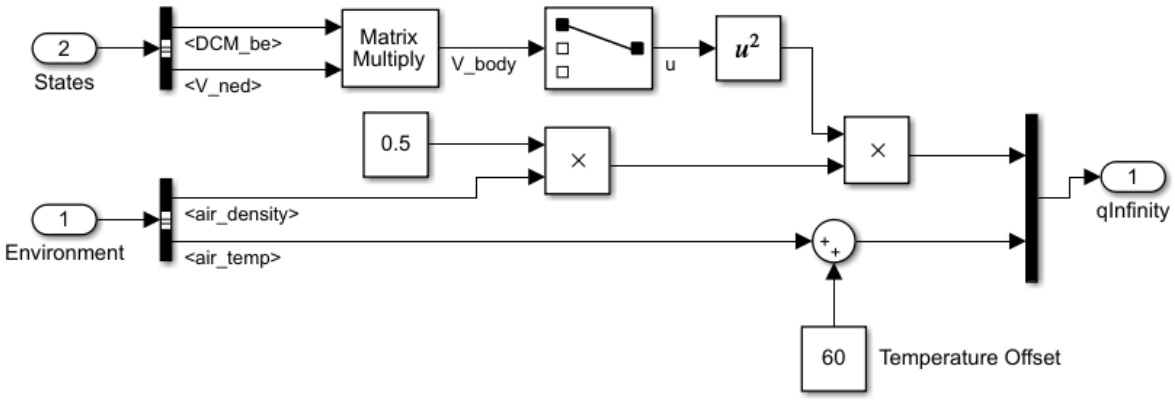
Finally, the complete group results as the image below. The temperature is compute as the ambient temperature plus an offset.



Variable pressure

5.3 Dynamic Pressure

The dynamic or velocity pressure input needs the static pressure raw measurement and the flow velocity. In our example the autopilot is supposed to be mounted in the X-axis in body frame. Therefore, from velocity in NED frames we apply a rotation to obtain velocity in body frames and then we pick the first component of the vector. The value of density is taken from the environment model.



Dynamic Pressure subsystem

5.4 Inertial Measurement Unit

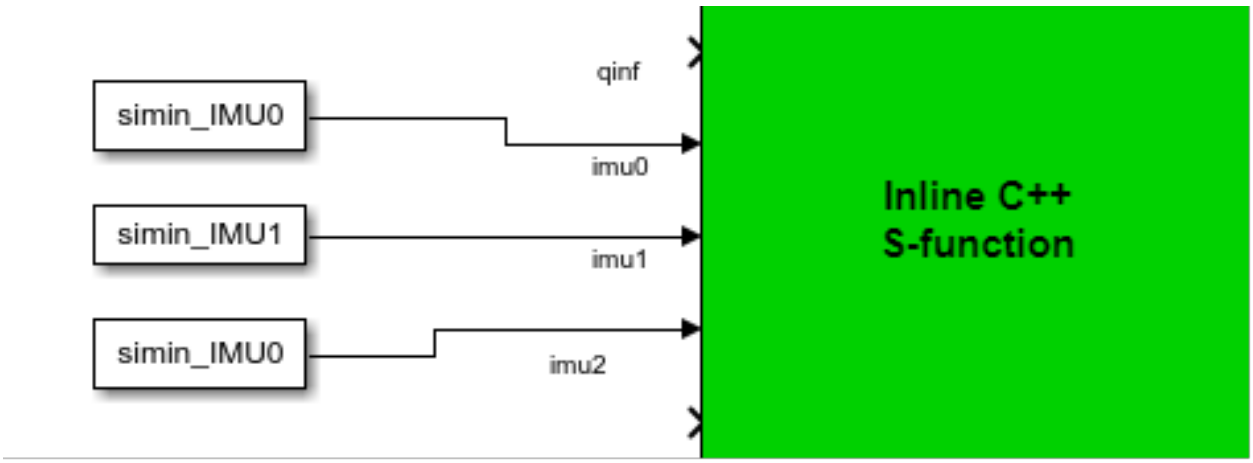
This device measures and informs about velocity, attitude and forces combining readings of accelerometers and gyroscopes. Veronte needs to receive 7 measurements: accelerometer in 3 axes, gyroscopes in 3 axes and device temperature. In the S-function there are 3 inputs for IMUs. The first one is the main unit and the second one the secondary unit. These units are mounted differently in the autopilot (is not aliganed with autopilot), so user has to keep in mind the rotation matrix which autopilot is using. This matrix is pre-configurated in each PDI and cannot be changed.

$$R_{main} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

$$R_{secondary} = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

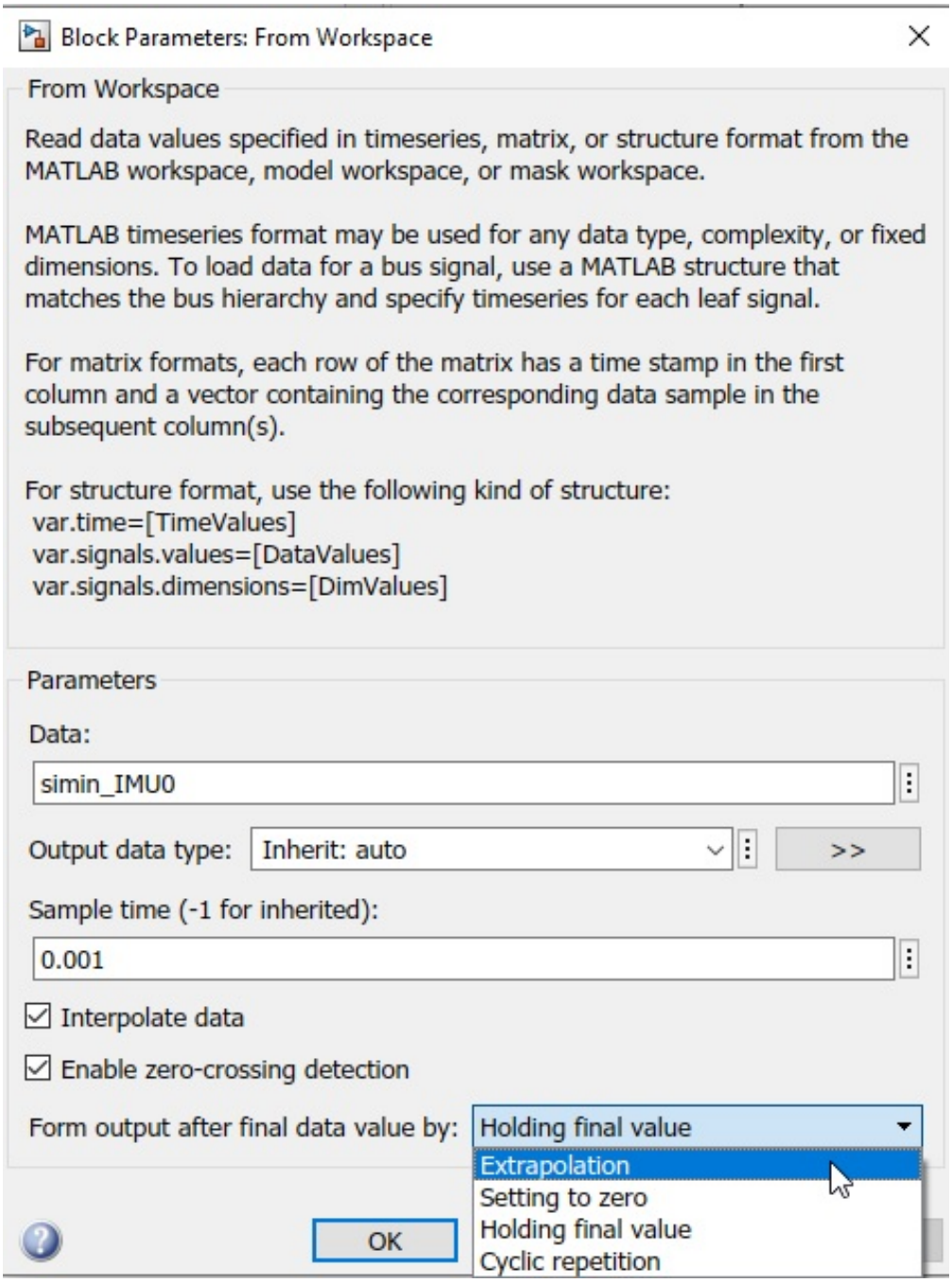
Veronte IMU rotation matrices

There are some ways to implement a suitable group of readings for a IMU. You can create a vector with constant values. Another option could be to store some data (i.e. from a previous flight), load in the matlab workspace, and then send this values to Simulink using the block name as **From workspace**.



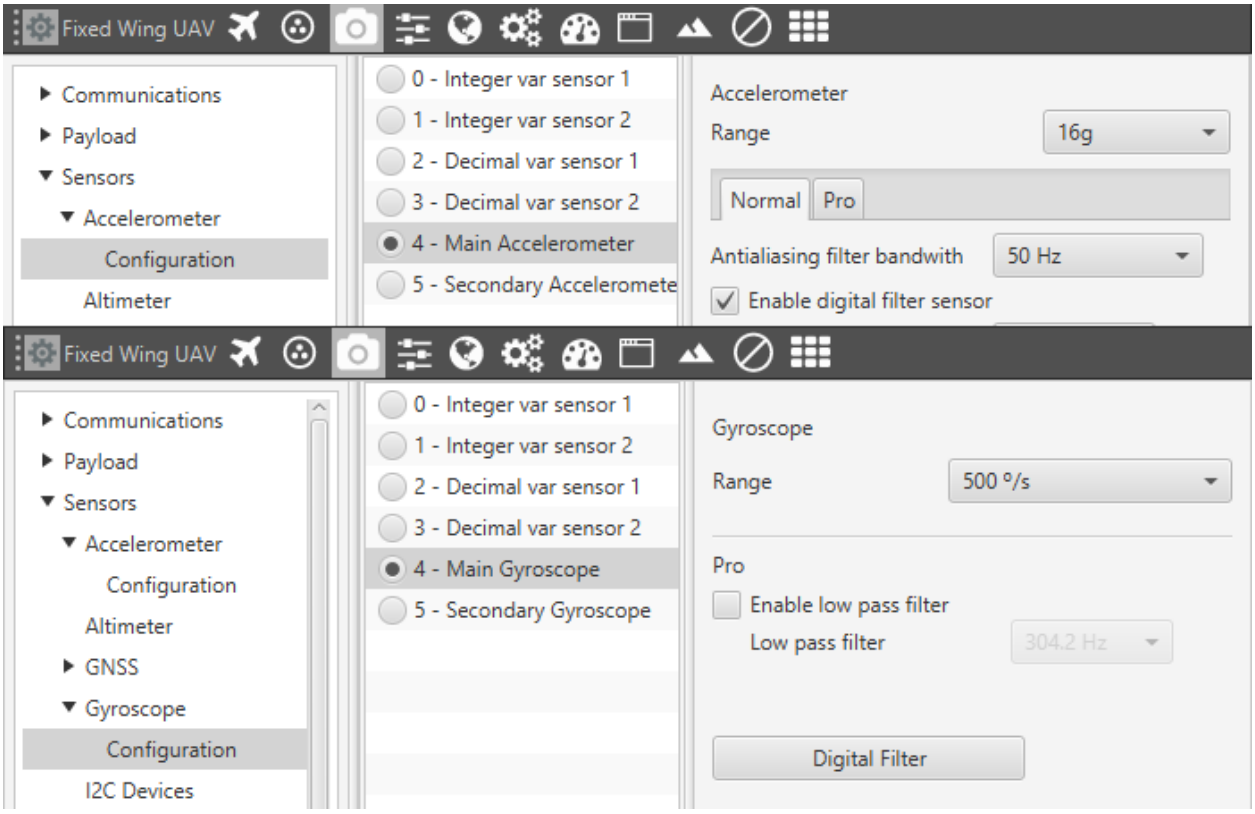
IMU data from workspace

This blocks allows the user to read from an array of values (and interpolate when there are no information in this step). Moreover, user can choose between several options in case data vector is over. For example, it is possible to extrapolate the information or restart the list of values.



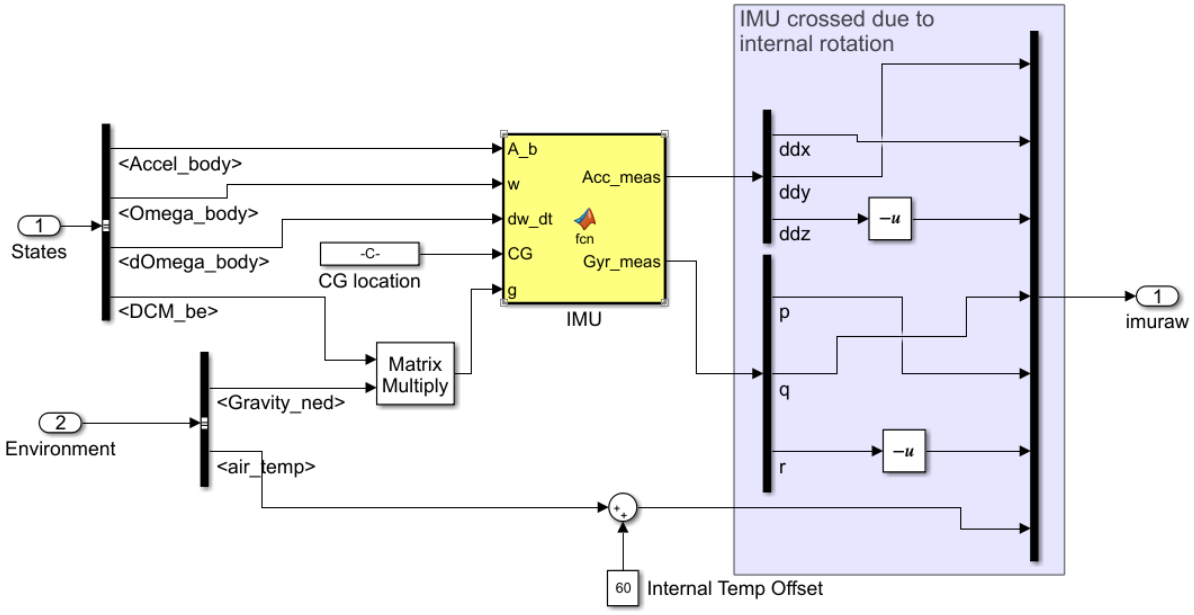
Methods applied when the final data value is reached

Another method is reading this values from Environment (gravity vector in NED, and air temperature), and from states (acceleration in body axes, angular velocity, angurlar acceleration, and the rotation matrix from NED to body). This values are entered to a Matlab function where IMU behaviour is simulated and the measurements are computed). Finally, user have to cross the measurements or apply a rotation matrix according to IMU sensor orientation. In the example below, this data is feeding the first port (in the PDI configuration this IMU is selected). Therefore, user has to cross the signals to fit the rotation matrix.



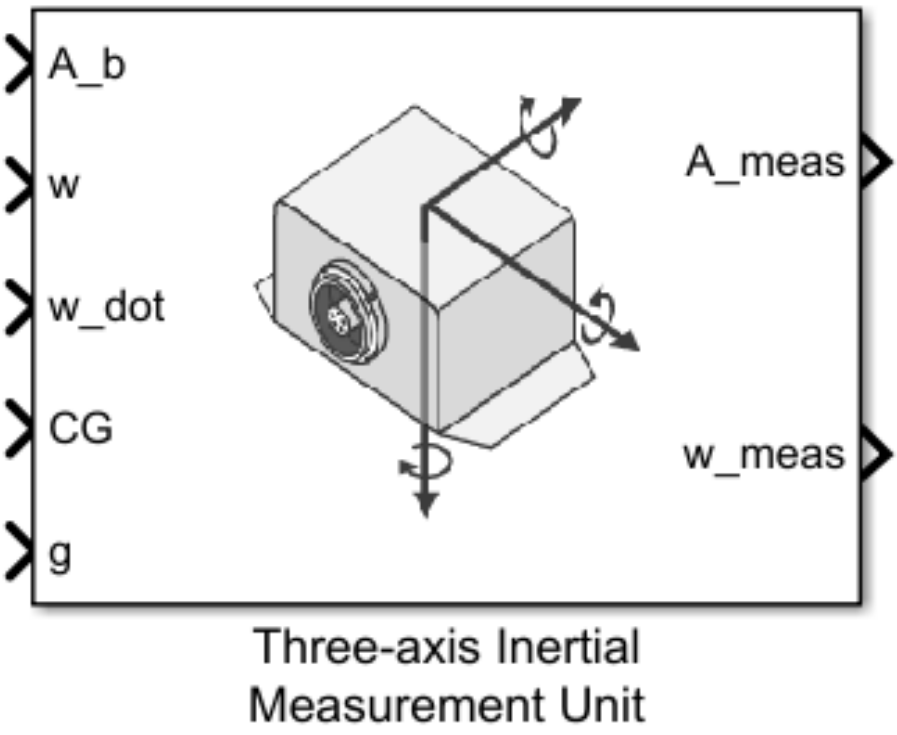
PDI configuration for main IMU

The complete subgroup results as follows:



IMU subgroup

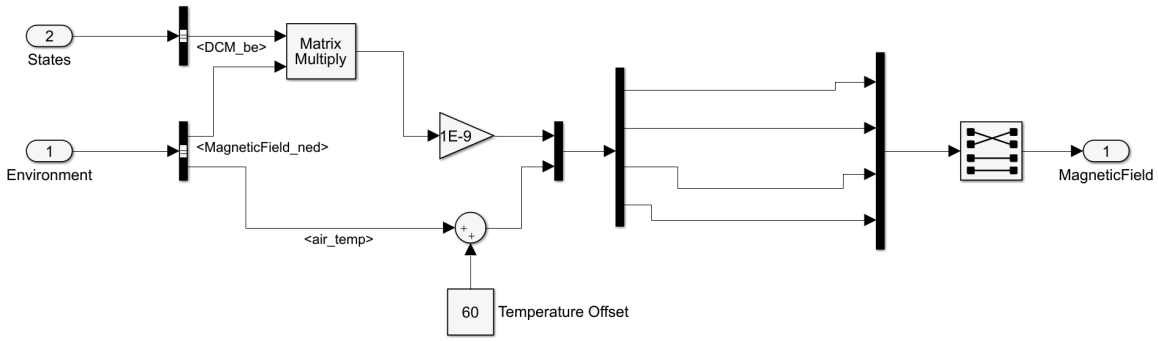
Instead of use a user function, Aerospace blockset include some functions for IMU simulation:



IMU block from Aerospace Toolbox

5.5 Magnetometer

The magnetometer block is simply a rotated environment magnetic field where the temperature of sensor has been added (same as before OAT + 60). S-function has 4 port for magnetometer readings (the internal one and 3 external - HMR2300, LIS3MDL, HSCDTD008A-). Also you can simulate another magnetometer and send the information by a serial port. Just as IMUs, user must have in mind how the magnetometer is mounted (rotation matrix). Therefore, the signal could be crossed as in the example below. You can use a Matrix multiplication block, or if it is simple, you can change directly the orientation with a selector crossing block.

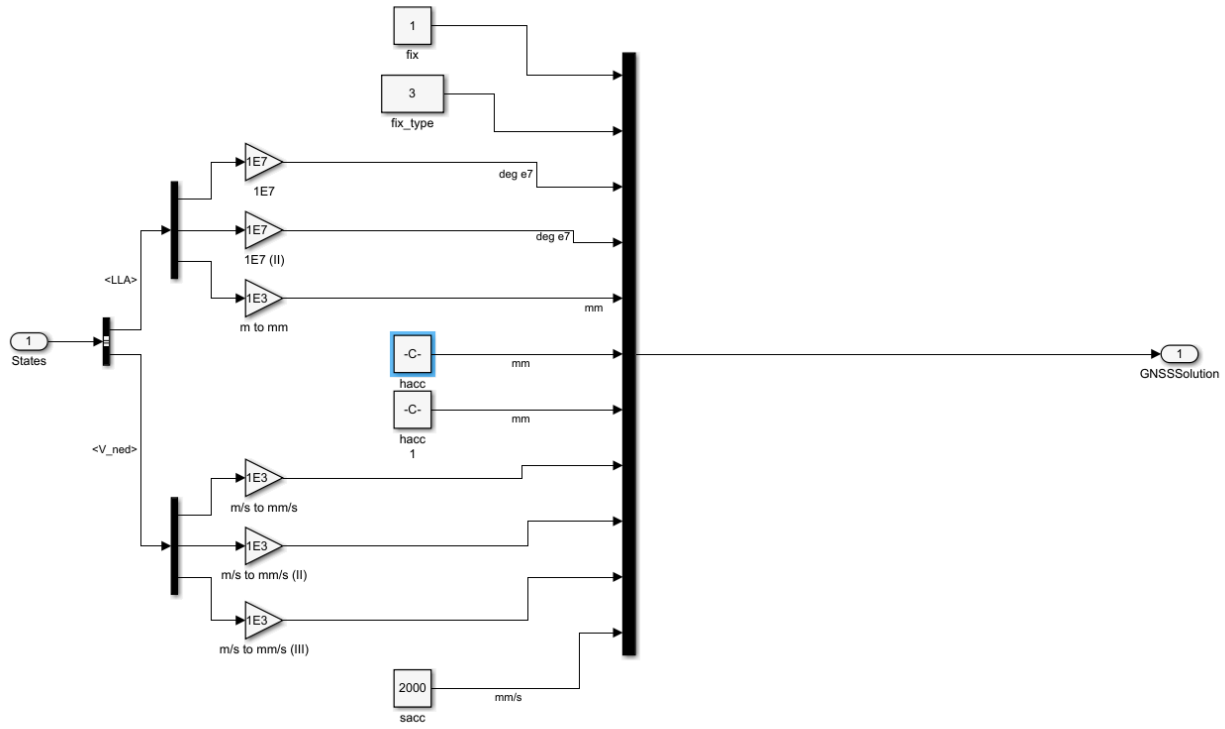


Magnetometer

5.6 GNSS

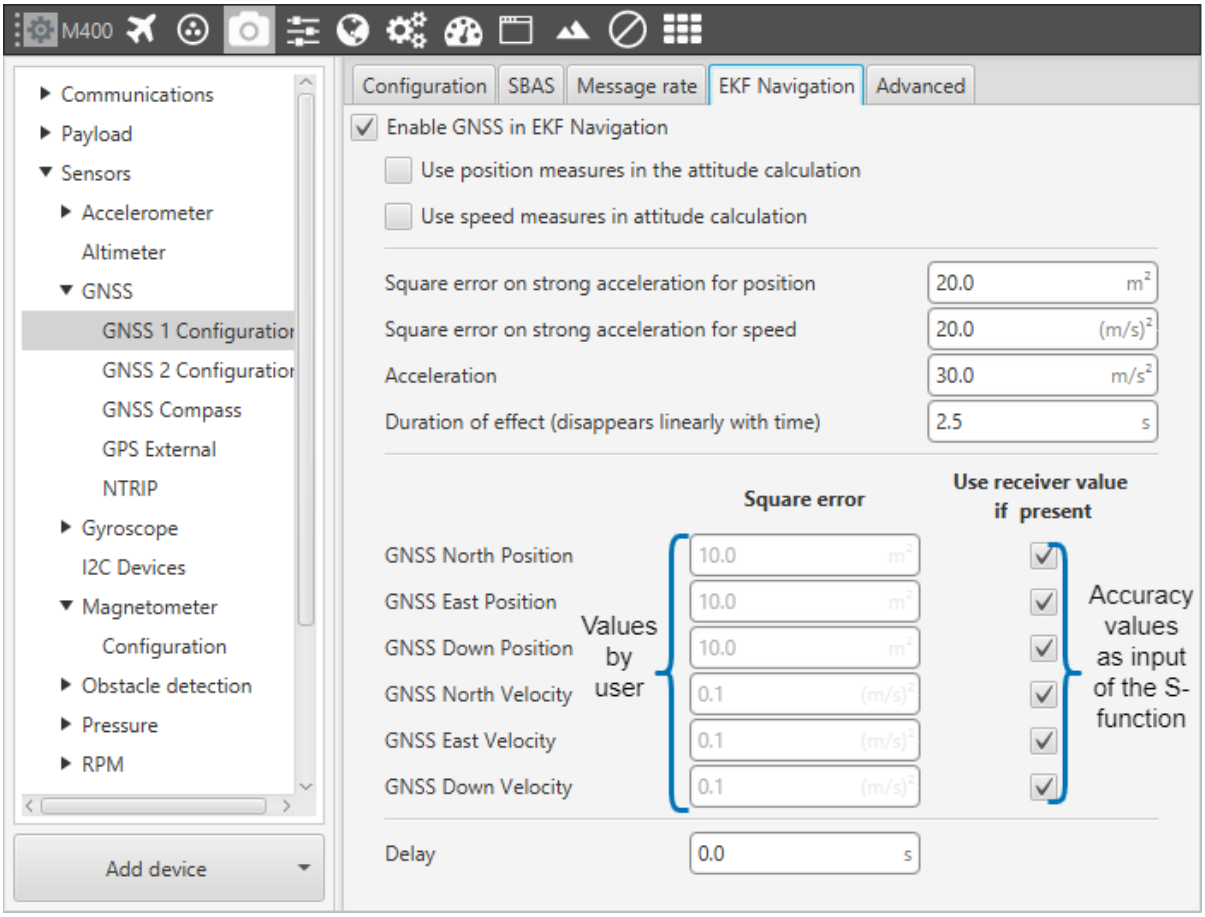
GNSS receiver ports (there are 2 ports -GNSS1 and GNSS2-) expect to receive an array with the following information:

1. Fix status
2. Fix type
    - 0: no fix
    - 1: dead reckoning only
    - 2: 2D-fix
    - 3: 3D-fix
    - 4: GNSS + dead reckoning fix
3. Latitude
4. Longitude
5. Altitude
6. Horizontal accuracy
7. Vertical accuracy
8. North Velocity
9. East velocity
10. Down Velocity
11. Speed Accuracy



GNSS array

The angle inputs are in degrees*10^7, and the distance inputs in millimeters. The accuracy values are equivalent to the square root of the square error. These values are supposed to be computed by the GPS device and are used in the EFK for GNSS solution. However, in the configuration files user can choose between these ones or values set by user.



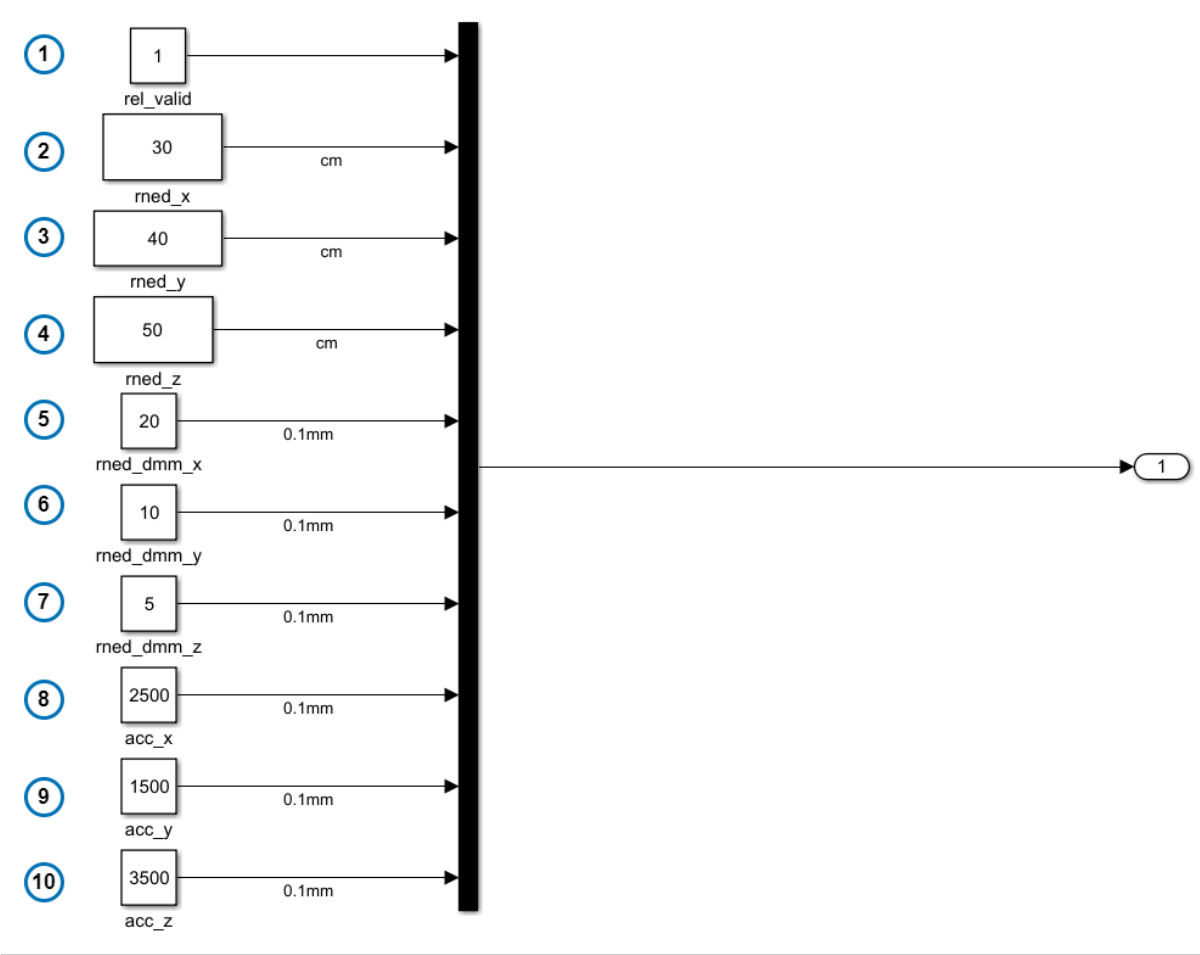
GNSS variances

RTK Example Block (Relative Position)

To enable RTK feature user has to modify the configuration (more information can be found in Veronte Autopilot Manual), and include more inputs by S-function. This input is named as *Relative Position*, and it requires an array of 10 elements.

1. **Status** : 0 is Data invalid and 1 is Data valid
2. **RelPosN** : North component of relative position vector (cm)
3. **RelPosE** : East component of relative position vector (cm)
4. **relPosD** : Down component of relative position vector (cm)
5. **relPosHPN** : High precision North component (mm)
6. **relPosHPE** : High precision East component (mm)
7. **relPosHPD** : High precision Down component (mm)
8. **accN** : Accuracy of relative position North component (mm)
9. **accE** : Accuracy of relative position East component (mm)
10. **accD** : Accuracy of relative position DOwn component (mm)

High precision components must be in range -99 to 99 millimeters. The full component of the relative position vector (in cm) is given by the addition of the 2 components. An example of this subgroup is shown below:

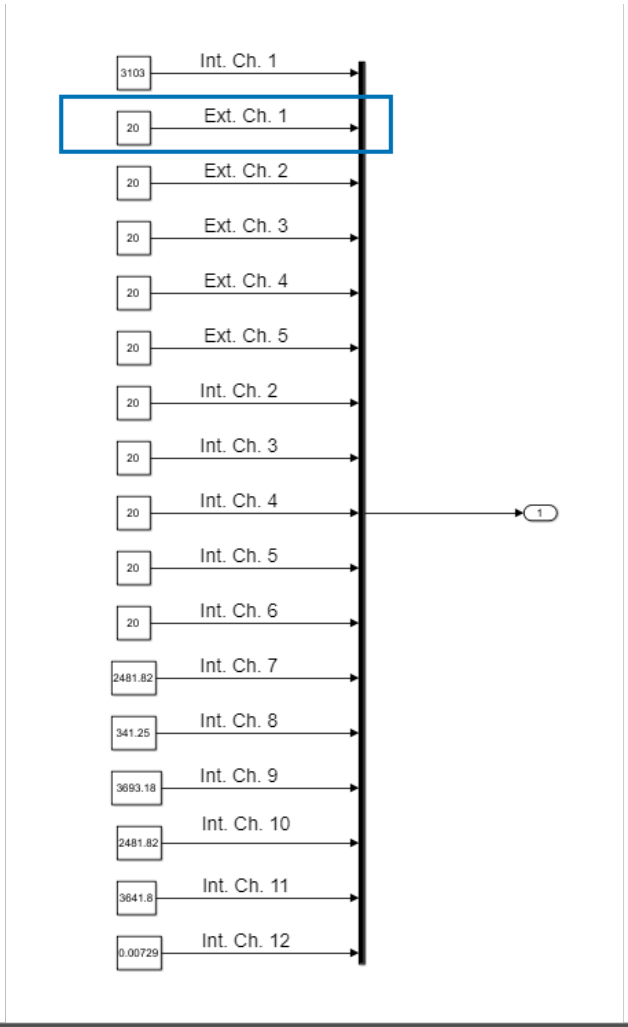


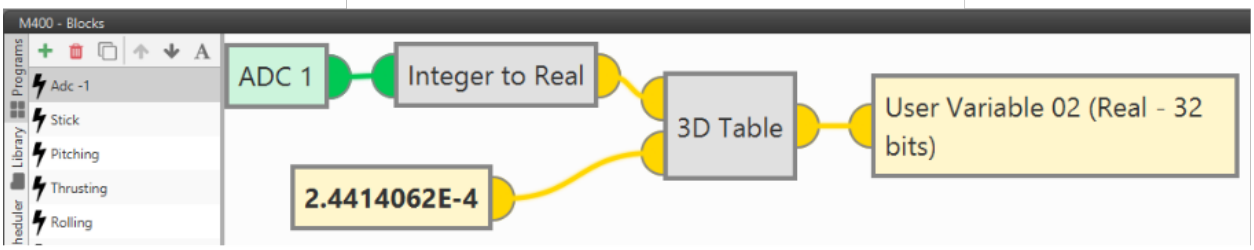
RTK inputs

5.7 Analog to Digital Converter Port

Veronte is equipped with 5 external ADC channel (linked to 5 pins) and 12 internal channels. Therefore, in total, user has to create an array of 17 elements. This values are stored as internal variables in Veronte, and you can use them in certain user programs. The order of this array is :Internal ADC Channel 1, External ADC Channel 1-5, Internal ADC 2-12.

In the picture below an example is shown (with the first external ADC pin).

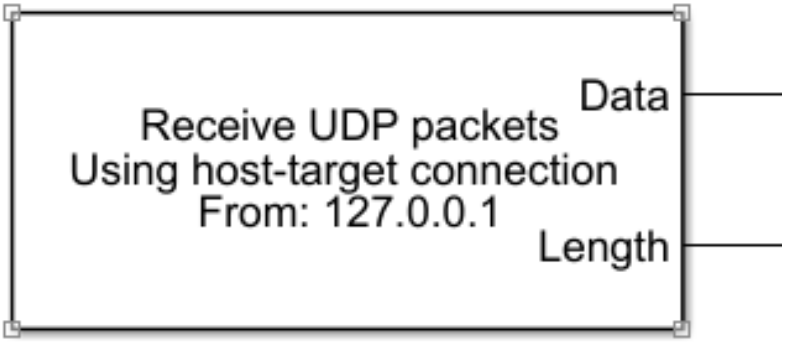




ADC readings

5.8 Serial communications

Veronte can manage input and output serial ports (more information in Veronte Autopilot Manuals), and we can simulate these as inputs and outputs on the S-function. An easy way to create serial frames (data in length wires) is by using the simulink UDP block. Therefore, the data coming in to veronte should be sent though UDP (if this approach is taken):



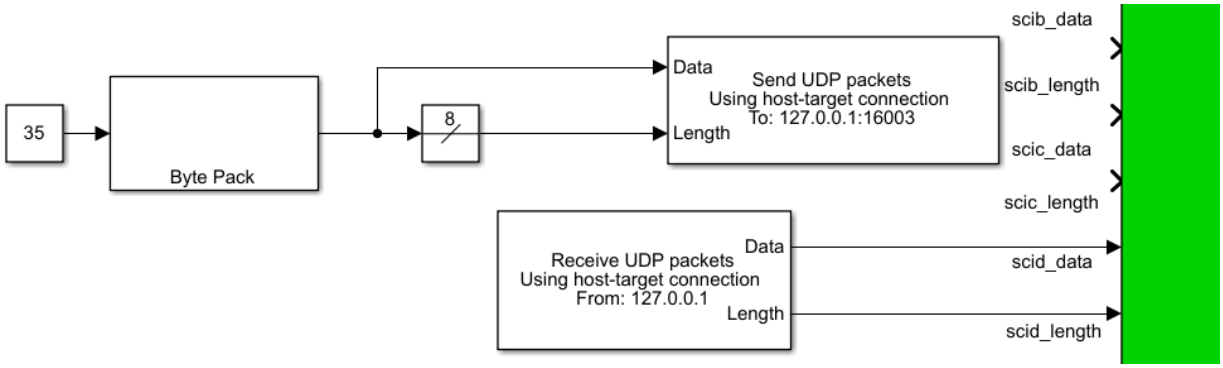
UDP Block

The ports that Veronte includes and that are represented in the S-function are the following:

- **USB** : USB port
- **SCIA** : 4G connection
- **SCIB** : Radio
- **SCIC** : Serial Port 485
- **SCID** : Serial Port 232

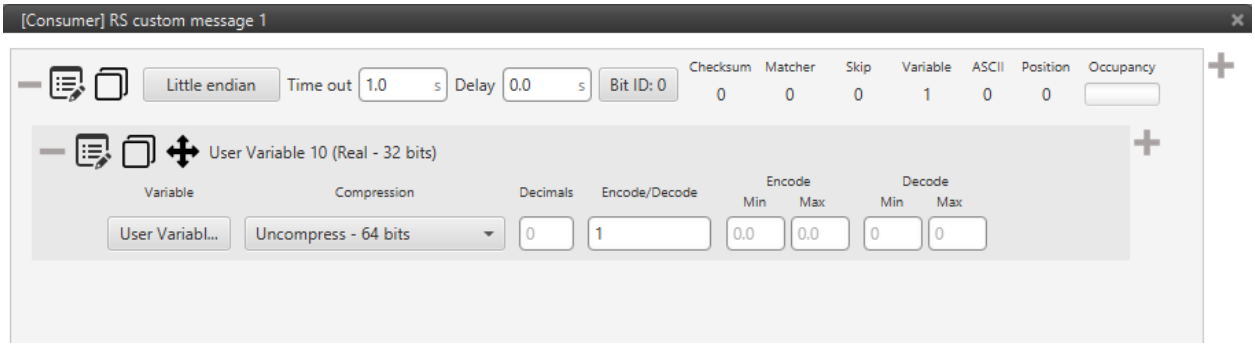
5.8.1 EXAMPLE: Sending a rs-232 message

In the example below we have sent a constant value as a rs-232 message. Firstly, you have to create the message as a bit array with **Byte Pack Block**. Then, it is neccesary to receive this information as UDP Packets to corresponding port (in this case 16003). **Width block** is used to compute data length. Then this UDP packet is send to S-function.



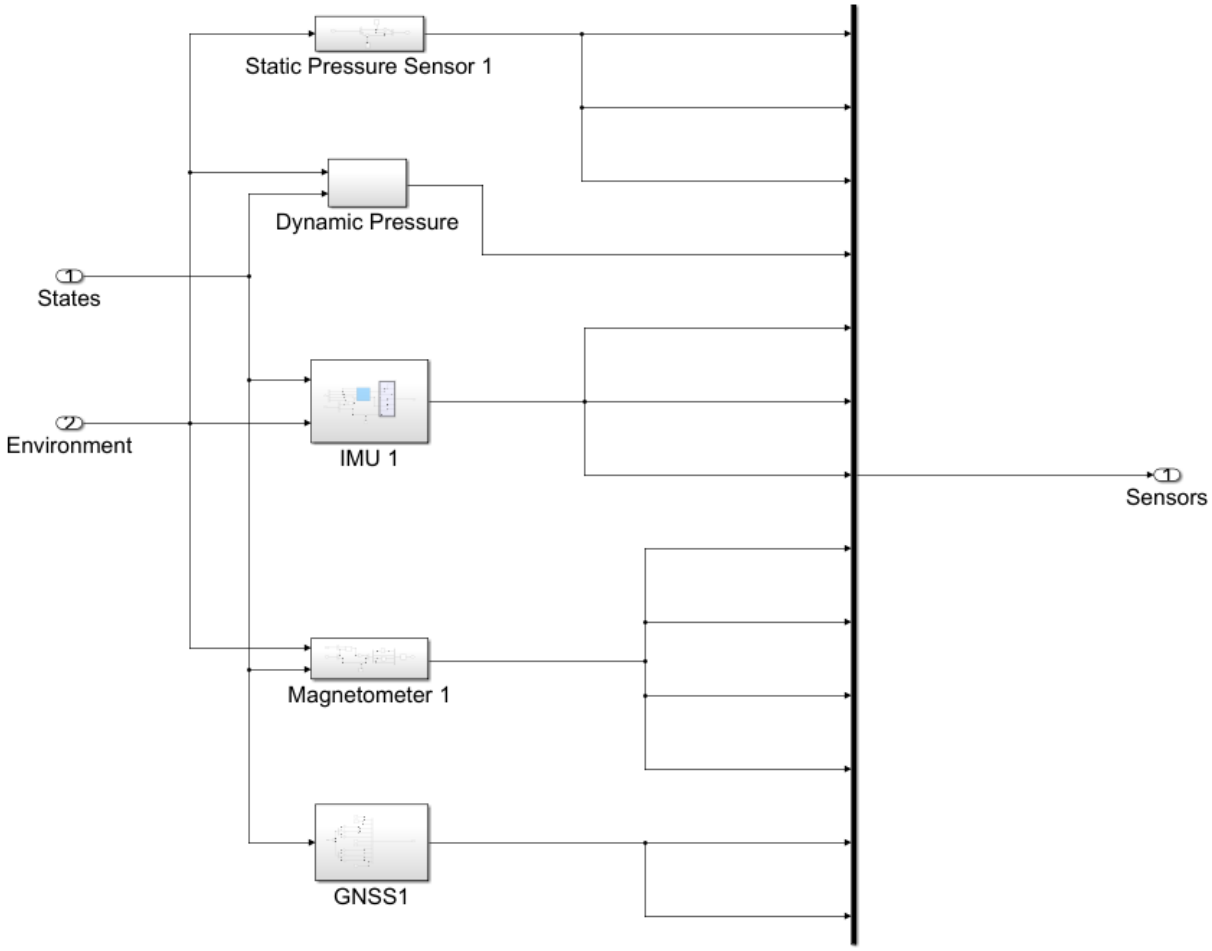
Sending a rs-232 message in Simulink

Finally, we have to configure a custom message to store this value in a user variable as follows:



Custom message

Sensors measurements are the inputs of the mex blocks (embedded code). To perform a correct simulation user have to set the inputs with the same scheme as Veronte reads them. Each sensor have a certain vector/array which usually includes raw data in one or more coordinates, the sensor temperatures, variances or square errors. User can set constant values for this variables or compute a complex environment model depending on the state of the plaform (position, velocity, etc.). This section aims to ilustrate how to implement the inputs described in the previous section. The structures that are shown here are orientative and, of course, can be adapted by the user:



Sensors inputs

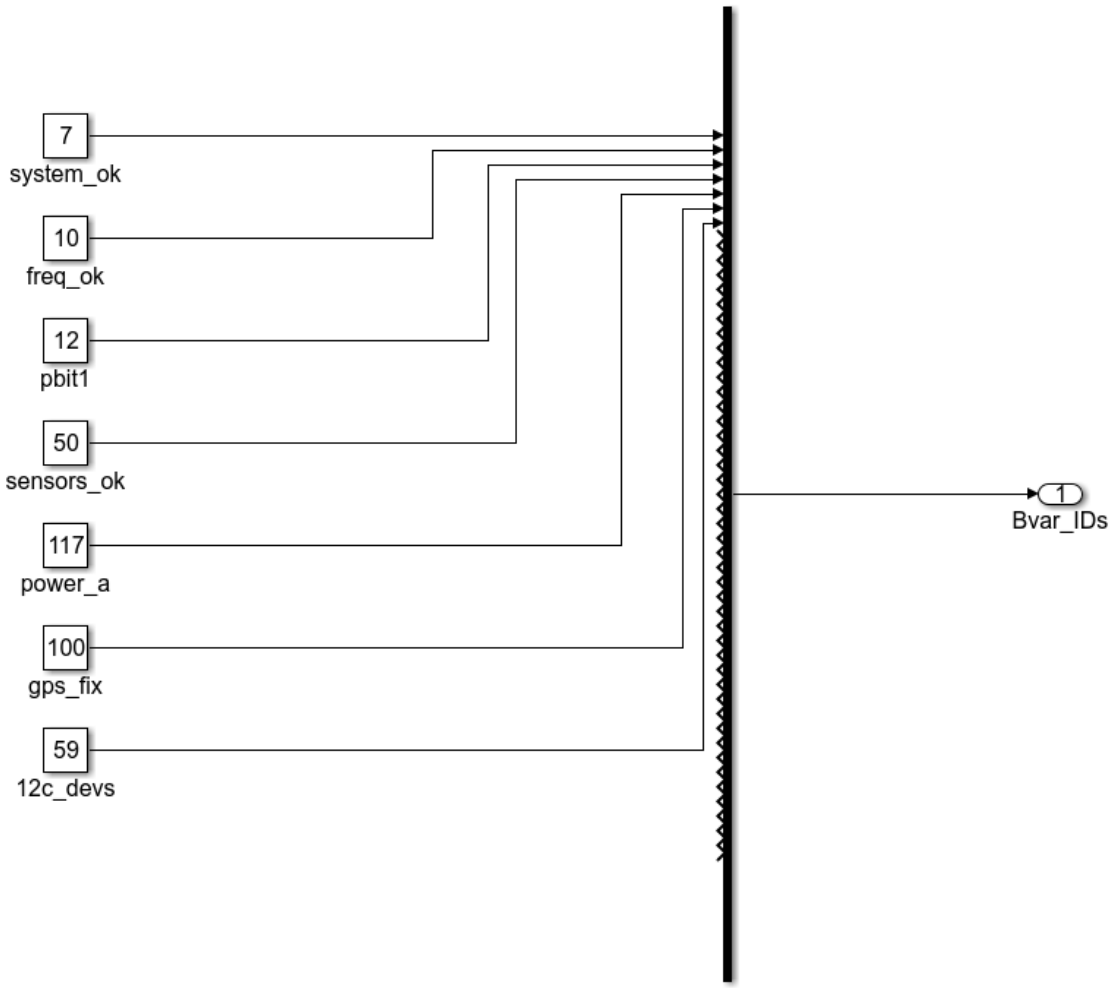
In the previous example the same type readings (static pressure, magnetic field, etc.) field all the port of each kind of sensor (Then the user can select the correct one in the configuration).

CHAPTER

SIX

MONITORING TELEMETRY

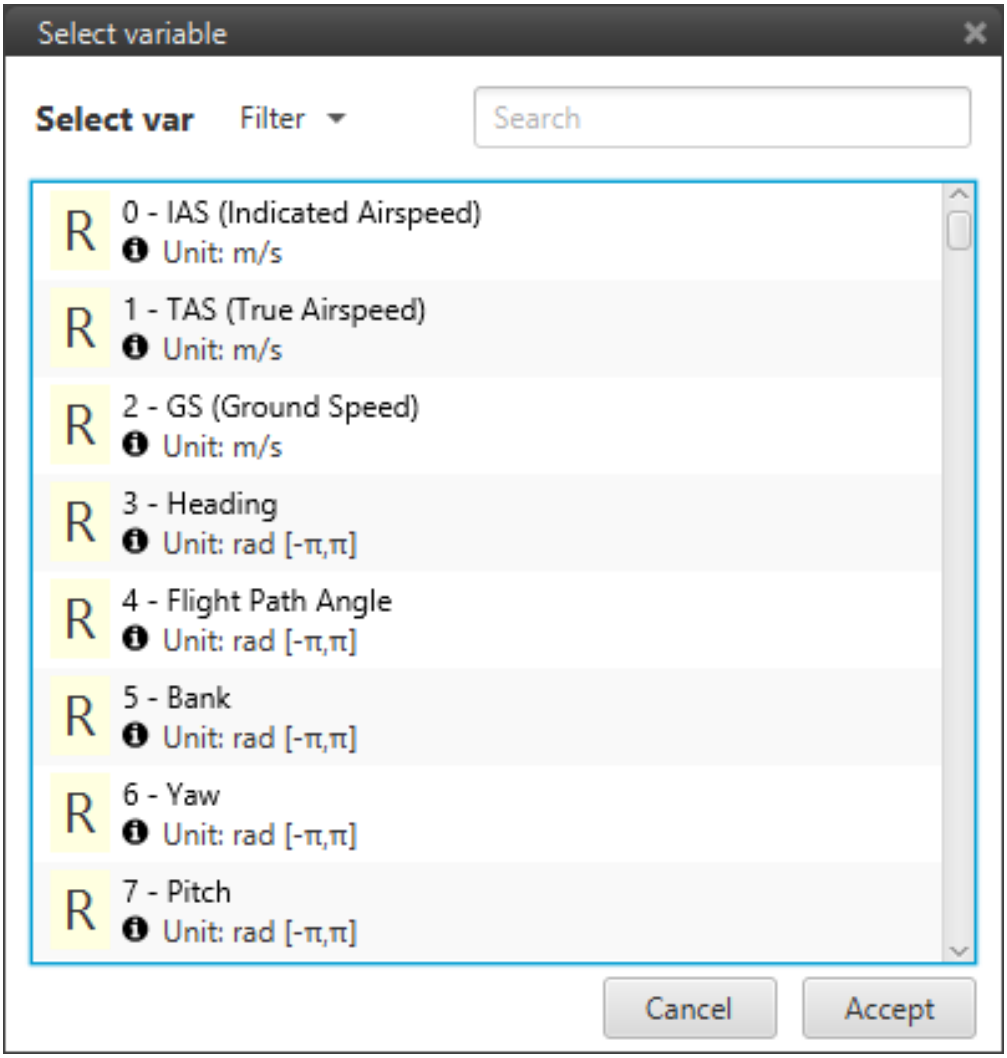
In the S-function there are three inputs specially dedicated to select custom temeletry (pin 22 for Bit variables, pin 23 for integers and pin 24 for reals). Each of this variables have an ID. The input structure of those is fixed and must be of size 50. User have to enter the corresponding Ids of the variables he is aiming to monitor. In the following example some BIT variables are requested:



Telemetry ID Mux

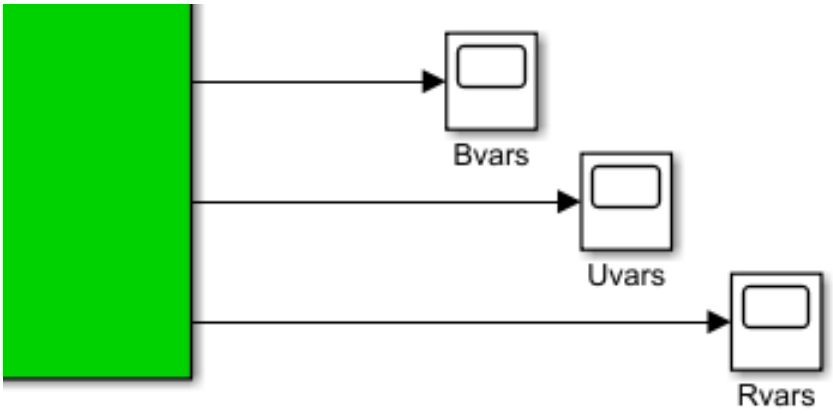
The ID of each variable in Veronte can be easily found in Veronte Pipe by adding a new workspace widget or in the Program window by adding a specific block (*Read Bit*, *Read Integer* or *Read Real*). The ID is labelled right before

variable name.



ID Indicator

Finally, to monitor or see their values, you can add a scope connected to the matching output (pin 32, 33 or 34), or use a demux block to separate the array in single values and connect them with a Display block.



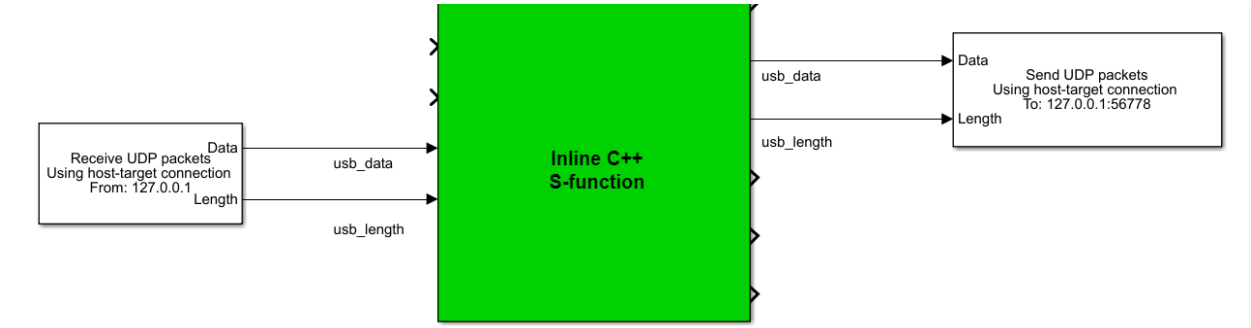
Display variables

CHAPTER

SEVEN

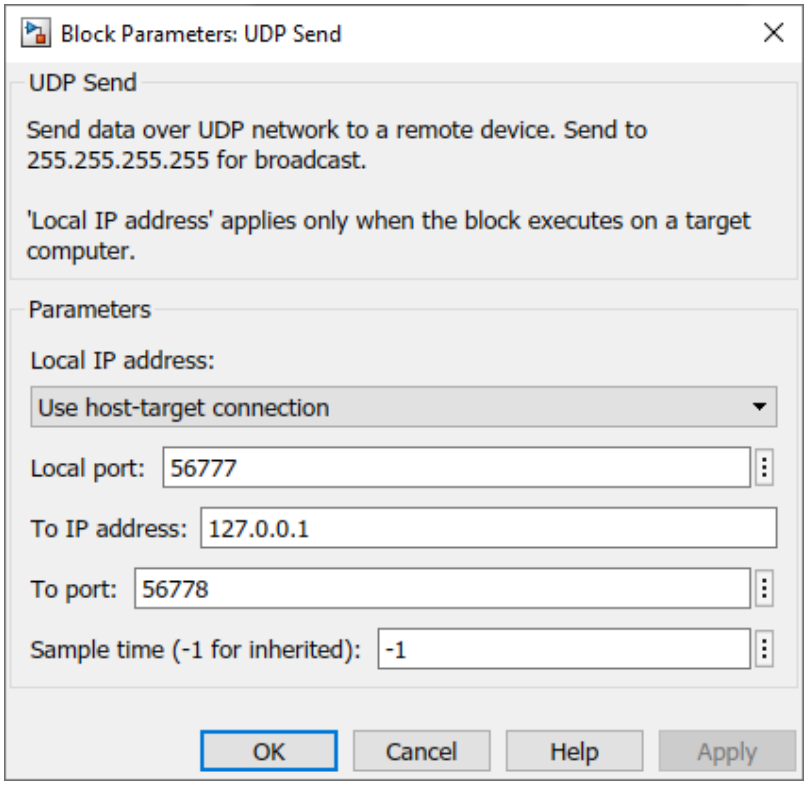
CONNECTING SIL & VERONTE PIPE

1. Add a UDP serial communication block and connect it to USB data and length.
2. Add a second UDP serial communication block and connect it to the USB output of veronte.



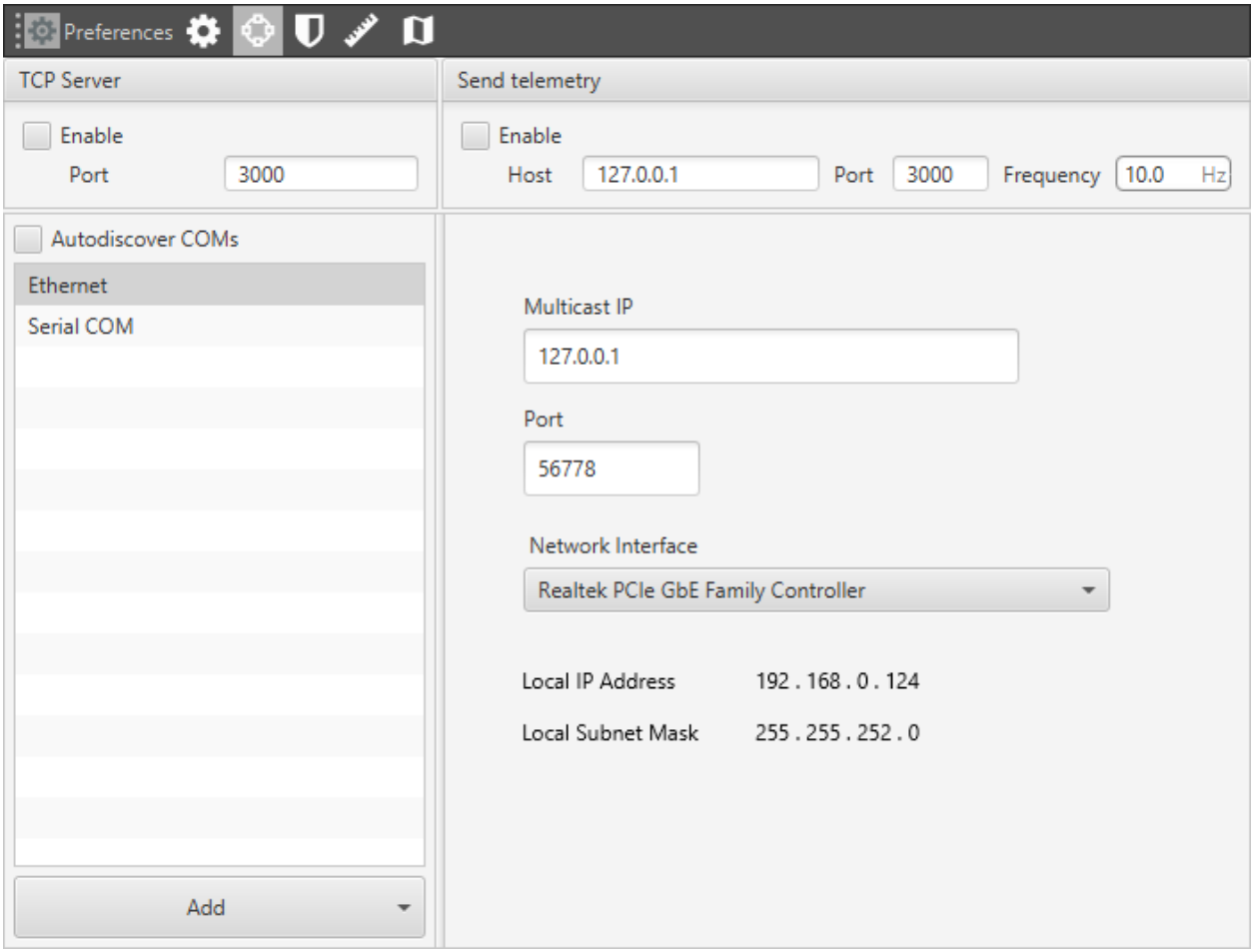
UDP Blocks

3. Configure your destination port.



Destination UDP Port

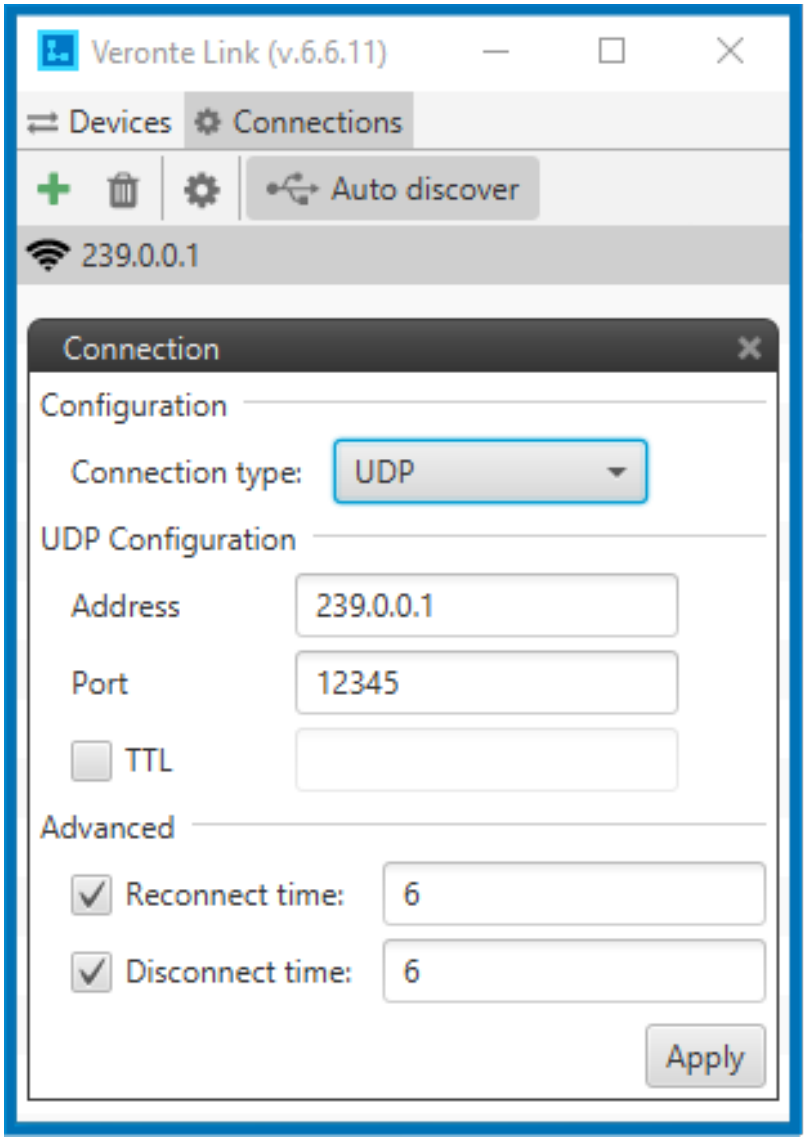
4. Set an ethernet network in Preferences as shown using the destination port selected before. Check that *Local IP Address* and *Local Subnet Mask* have non-zero values.



Destination UDP Port (Pipe)

7.1 Pipe v6.6 and higher

In case of using VeronteLink (comunication with Pipe v6.6 or higher) you have to configure the connections tab selecting UPD as connection type and set the configuration as in fourth step.



Destination UDP Port (Veronte Link)

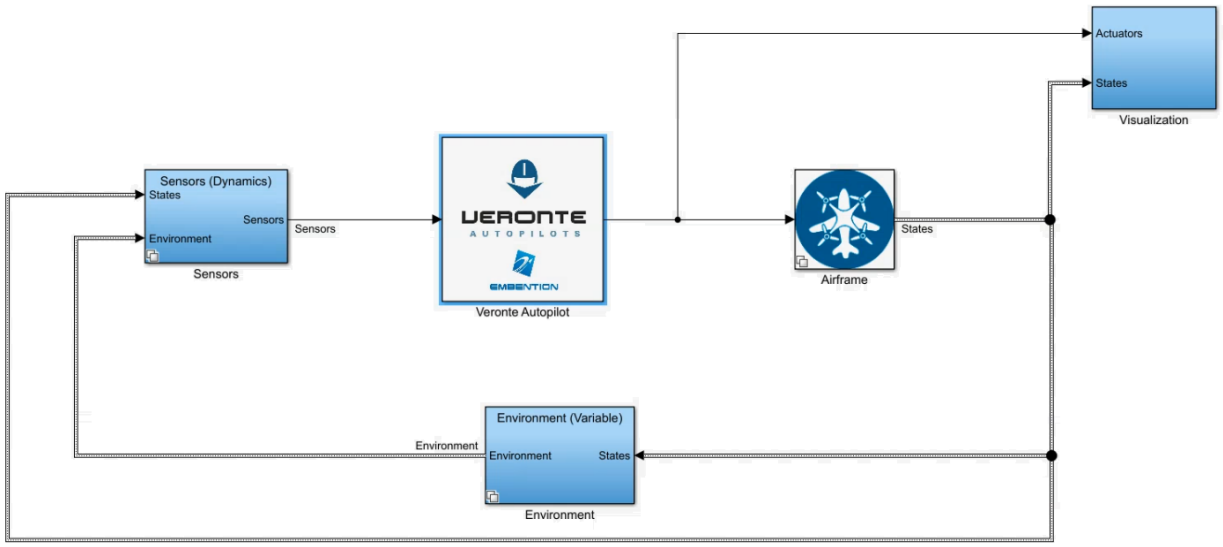
CHAPTER

EIGHT

SIMULATION

8.1 Complete Simulation

After setting the main blocks, the result should look like this:

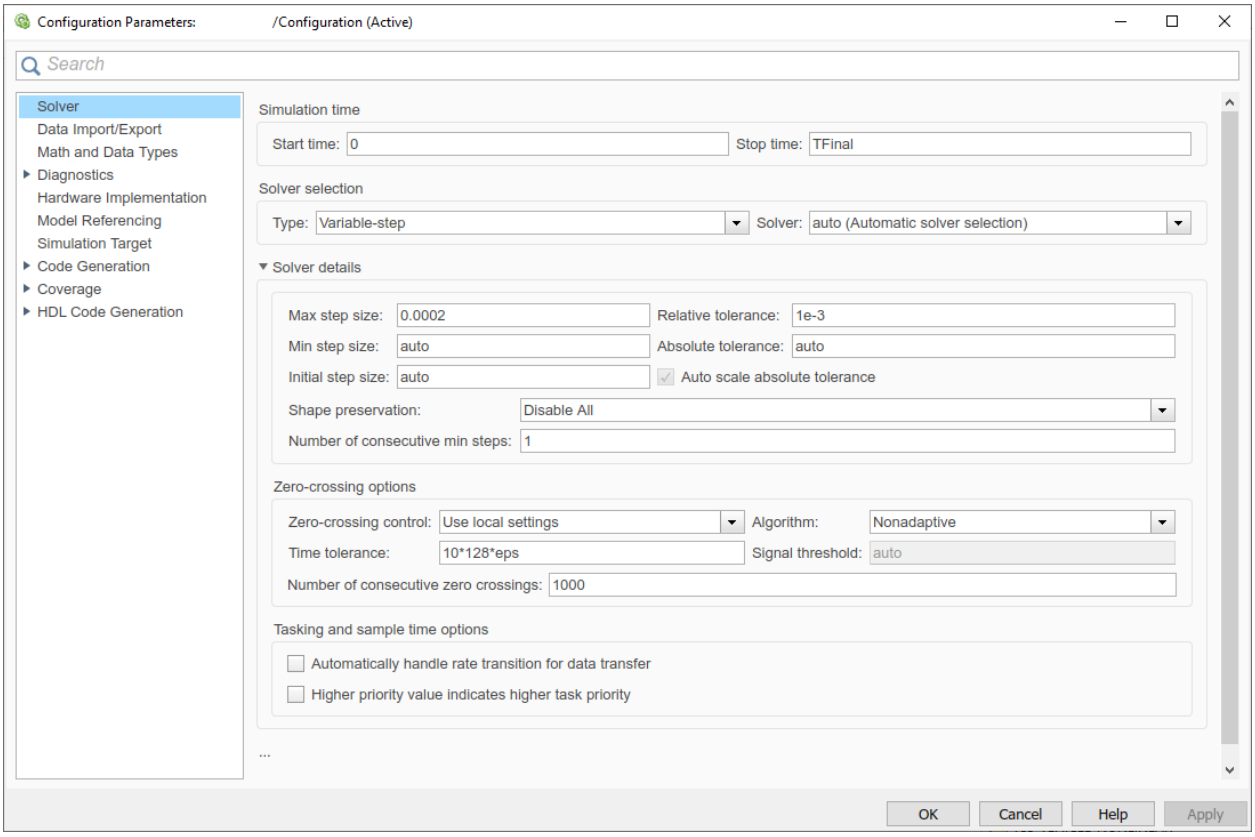


Complete Setup Example

The main systems are:

- **Veronte Autopilot**: It contains our flight control software. It basically consists of the S-function and their link with the rest of the blocks (sensors, outputs, etc.)
- **Airframe**: a model of the flight dynamics.The inputs of this systems are the output of the Veronte autopilot (nominal value for servos). For example, for a quacopter, the input of this block consists of the values of the PWM signal (one for each motor). Then with this value the airframe system updates the platform's state. The state vector is used for predict the new environment conditions and the sensors readings.
- **Environment**: a model of the atmosphere, magnetic field, WGS84…
- **Sensors**: it contains individual blocks or subgroup of all the sensors that veronte needs as input.
- **Visualization**: It contains Display blocks, scopes, flight instruments…

The time step should be set to 0.0002 as shown in the next figure in order to guarantee a good GNC/Adquisition frequency:

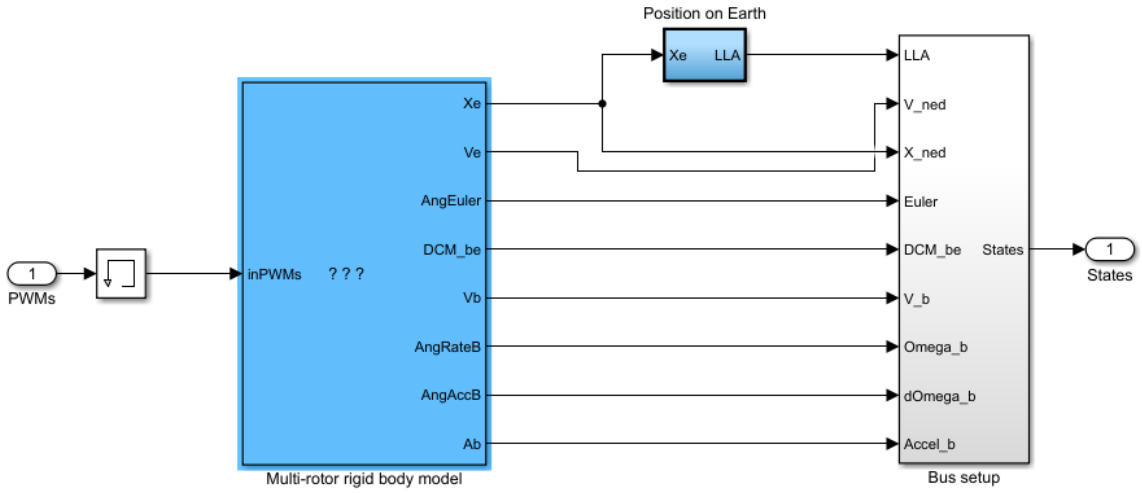


Time step settings

8.2 Quadcopter Example

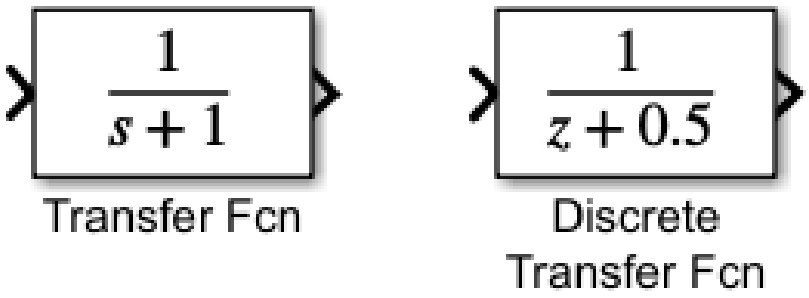
In this section a basic example about how make an airframe model is shown.

In the picture below this is represented. Once Veronte receives all the sensor information, the autopilot computes the guidance and control algorithms. As a result, the autopilot computes the necessary value for servos. The inputs of the system are the values of the PWM, it means, servos output (pin 2). User also can use the control output value directly. These values have to be entered in a user function that computes the airframe model. However, these values are the current ones. To perform a properly simulation the input values must be those from the previous step. It can be solved with the memory block (stored previous step input).



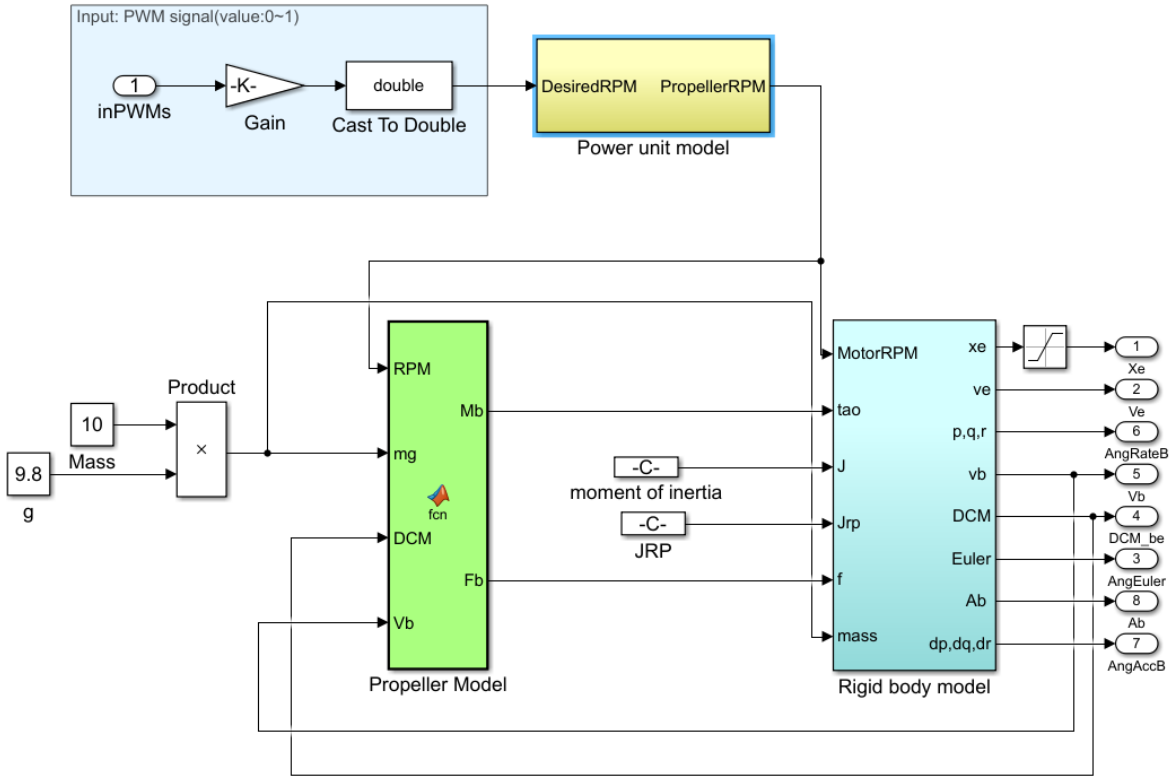
Complete Setup Example

In this example, the value of PWM is transformed to RPM. For this, it is necessary to implement an engine/rotor model, for example, by using a Transfer function (power unit model).



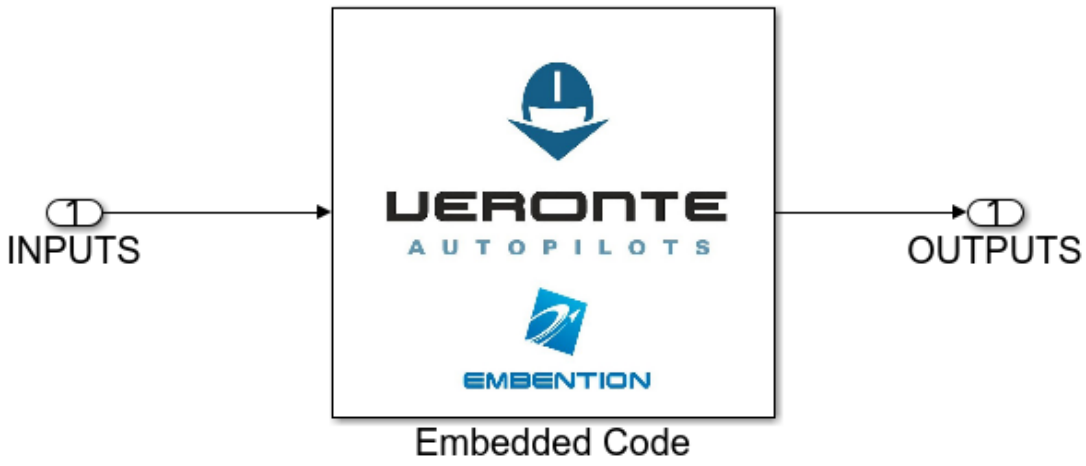
Transfer function

Once the RPM are calculated, the aerodynamic forces and moments can be computed with a properly model. Then this forces are entered to the Rigid Body model to integrate the vehicule state.



Airframe model

A complete simulation is composed by many systems or blocks. In this manual the sensor the environment and the autopilot subsystem have been already introduced. All theses blocks must be combined with others such as Airframe block (a brief example will be included in this manual).



Veronte autopilot

This manual contains the information required for the user to run a Software in the loop simulation using Matlab and Simulink. This document includes a basic description of how our autopilot works with Simulink, and some examples to allow the user to create a complete model from scratch.

Version UM.306.6.2.38

Date 2023-01-30