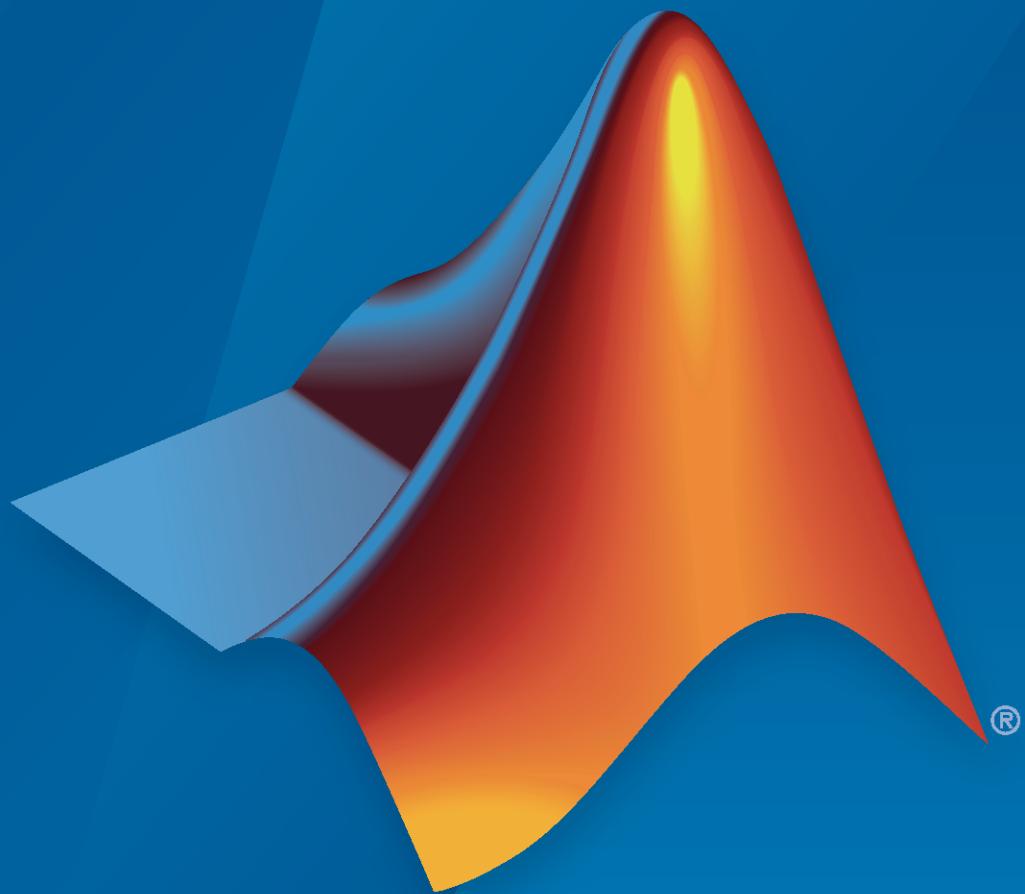


UAV Toolbox

Support Package for PX4® Reference



MATLAB® & SIMULINK®

R2025a

 MathWorks®

How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us
Phone: 508-647-7000



The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

UAV Toolbox Support Package for PX4® Autopilots Reference

© COPYRIGHT 2019–2025 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2019	Online only	New for Version 19.1.0 (R2019a)
April 2019	Online only	Revised for Version 19.1.1 (R2019a)
May 2019	Online only	Revised for Version 19.1.2 (R2019a)
September 2019	Online only	Revised for Version 19.2.0 (R2019b)
November 2019	Online only	Revised for Version 19.2.1 (R2019b)
March 2020	Online only	Revised for Version 20.1.0 (R2020a)
April 2020	Online only	Revised for Version 20.1.1 (R2020a)
September 2020	Online only	Revised for Version 20.2.0 (R2020b)
March 2021	Online only	Revised for Version 21.1.0 (R2021a)
September 2021	Online only	Revised for Version 21.2.0 (R2021b)
March 2022	Online only	Revised for Version 22.1.0 (R2022a)
September 2022	Online only	Revised for Version 22.2.0 (R2022b)
March 2023	Online only	Revised for Version 23.1.0 (R2023a)
September 2023	Online only	Revised for Version 23.2.0 (R2023b)
March 2024	Online only	Revised for Version 24.1.0 (R2024a)
September 2024	Online only	Revised for Version 24.2.0 (R2024b)
March 2025	Online only	Revised for Version 25.1.0 (R2025a)

Blocks

1

Getting Started with uORB Blocks for PX4 Autopilots Support Package	1-80
Reading GPS Data from PX4 Autopilot	1-91
Getting Started with PX4 Analog Input Block for ADC Channels	1-95
Send and Receive Serial Data Using PX4 Autopilots Support Package	1-101
Getting Started with PWM Blocks for PX4 Autopilots	1-107
Read PX4 System Parameters Using PX4 Autopilots Support Package	1-117
Code Verification and Validation with Processor-in-the-Loop (PIL) Simulation	1-121
Attitude Control for X-Configuration Quadcopter Using External Input	1-133
Position Tracking for X-Configuration Quadcopter	1-149
Position Tracking for X-Configuration Quadcopter Using Rate Controller	1-160
MAT-file Logging on SD Card for PX4 Autopilots	1-169
Reading Accelerometer Values from an I2C based Sensor Connected to a PX4 Autopilot	1-176
Getting Started with Connected IO for PX4 Host Target	1-184
Connecting to NSH Terminal for Debugging	1-189
Monitor and Tune PX4 Host Target Flight Controller with Simulink-Based Plant Model	1-191
Getting Started with Connected IO for PX4 Autopilot	1-199
Code Execution Profiling on PX4 Target in Monitor & Tune Simulation	1-204
PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics in Simulink	1-214

Scenario Simulation and Flight Visualization with PX4 Hardware-in-the-Loop (HITL) and UAV Dynamics in Simulink	1-224
PX4 Autopilot and NVIDIA Jetson in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics Modeled in Simulink	1-232
CAN Communication with Pixhawk Using Raspberry Pi	1-251
Transmit and Receive Data Using PX4 CAN Blocks	1-257
PX4 Hardware-in-the-Loop (HITL) Simulation with Fixed-Wing Plant in Simulink	1-261
Obstacle Avoidance in NVIDIA Jetson with PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics Modeled in Simulink	1-268
Reading GPS Data over UAVCAN	1-290
Log Simulink Signals Using PX4 ULog	1-293
UVify IFO-S Autopilot and Nvidia Jetson Hardware-in-the-Loop (HITL) Simulation in Simulink	1-303
Plant and Attitude Controller Model for Hexacopter	1-320
PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with Speedgoat in Simulink	1-323
PX4 Stock Autopilot in HITL Simulation with UAV Dynamics modeled in Simulink	1-330
PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Tilt-Rotor Plant in Simulink	1-334
Visualize PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Over Urban Environment	1-344
Getting Started with Actuator Control over PWM	1-362
Getting Started with Actuator Control over UAVCAN/DroneCAN	1-374
Simulate Manual Control for Fixed-Wing with PX4 Host Target	1-381
Visualize 3D Scenarios in Unreal Engine with PX4 Host Target Simulation	1-390
Getting Started with PX4 Write Parameter Block for PX4 Autopilots	1-394
Actuator Control Using Write Thrust & Torque Blocks for PX4 Autopilots	1-397
PX4 Hardware-in-the-Loop (HITL) Simulation and Visualization with VTOL UAV	1-407

Pin Mapping for Pixhawk Series Controller Boards

2

Index Numbers for Analog Channels on Pixhawk Series Controller Boards

2-2

Port Mapping for Serial Ports

3

Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards

3-2

Coder Target Context Sensitive Help

4

Model Configuration Parameters for PX4 Flight Controller	4-2
Hardware Implementation Pane Overview	4-3
Build Options	4-4
HITL	4-5
Connected I/O	4-5
External mode	4-6
Onboard Connectivity	4-7
Clocking	4-7
PIL	4-7
I2C	4-8
CAN	4-8
MAVLink	4-9
/dev/tty	4-9
Overrun Action	4-10
Build options	4-11
Build action	4-11
CMake configuration	4-11
Automatically determine serial port for firmware upload	4-11
Serial port for firmware upload	4-11
Serial port for NuttX (NSH) Terminal	4-12
Allow flashing FMUv3 CMake configuration on Pixhawk 1	4-12
Simulator	4-12
HITL	4-13
Enable HITL Mode	4-13
Simulator	4-13
Connected I/O	4-14
Hardware board Serial Port	4-14
Use the same host serial port for Connected I/O as used for firmware upload	4-14
Host Serial Port	4-14

Clocking	4-15
CPU Clock (MHz)	4-15
External Mode	4-16
Communication interface	4-16
Use the same host serial port for External mode as used for firmware upload	4-16
Host Serial Port	4-16
Hardware board Serial Port	4-16
Set logging buffer size automatically	4-17
Logging buffer size (in bytes)	4-17
Verbose	4-18
Onboard Connectivity	4-19
Onboard Computer IP Address	4-19
PIL	4-20
Hardware board serial port	4-20
Use the same host serial port for PIL as used for firmware upload (mentioned under 'Build Options')	4-20
Host Serial Port	4-20
I2C	4-21
Bus 1 speed (KHz)	4-21
Bus 2 speed (KHz)	4-21
Bus 3 speed (KHz)	4-21
Bus 4 speed (KHz)	4-21
CAN	4-22
CAN Port	4-22
Baud rate (in bits/s)	4-22
Test mode	4-22
MAVLink	4-24
Enable MAVLink on /dev/ttyACM0	4-24
/dev/tty	4-25
Baud rate	4-25
Parity	4-25
Stop bits	4-25
Enable hardware flow control	4-25
View port map	4-25
PWM Frequency	4-26
Enable Oneshot125 protocol for Main PWM channels	4-26
Configure Frequency for Main PWM channels (in Hz)	4-26
Enable Oneshot125 protocol for AUX PWM channels	4-26
Configure Frequency for AUX PWM channels (in Hz)	4-26
Main PWM	4-27
Main PWM channel x failsafe ON time value (in us)	4-27
Main PWM channel x disarmed ON time value (in us)	4-27
AUX PWM	4-28
AUX PWM channel x failsafe ON time value (in us)	4-28

AUX PWM channel x disarmed ON time value (in us)	4-28
PWM	4-29
PWM channel x disarmed ON time value (in us)	4-29
Overrun Action	4-30
Shutdown PX4 Autopilot upon overrun	4-30

Bus Mapping for I2C Blocks

5

I2C Bus Port Numbers for Labels on PX4 Autopilots	5-2
---	-----

Functions

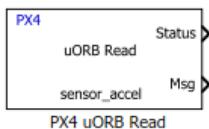
6

Blocks

PX4 uORB Read

Read uORB data for the specified uORB topic

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 uORB Read and Write Blocks

Description

The PX4 uORB Read block creates a Simulink® nonvirtual bus that corresponds to the specified uORB topic.

On each simulation step, the block checks if a new message is available on the specific topic. If a new message is available, the block retrieves the message and converts it to a Simulink bus signal. The **Msg** port outputs this new message. If a new message is not available, **Msg** outputs the last received uORB message. If a message has not been received since the start of the simulation, **Msg** outputs a blank message.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Msg — uORB message
nonvirtual bus

uORB message, returned as a nonvirtual bus. The type of uORB message is specified using the parameter, **Topic to subscribe to**.

Connect a Bus Selector block to the **Msg** output to extract the signals.

Data Types: bus

Status — Status of the received uORB data
0 | 1

The Status output indicates if the uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the **Msg** output is the latest, and a value of 1 indicates that the uORB data was received during the previous time step.

This output can be used to trigger subsystems for processing the new messages received in the uORB network.

Data Types: Boolean

Parameters

Main tab

Topic to subscribe to — Name of the uORB topic to which the block needs to subscribe
sensor_accel (default) | One of the uORB topic

Select the name of the uORB topic that the block needs to subscribe to read the uORB data. Click **Select** to browse the topic names available in the `Firmware/msg` folder of the `px4` directory that you downloaded.

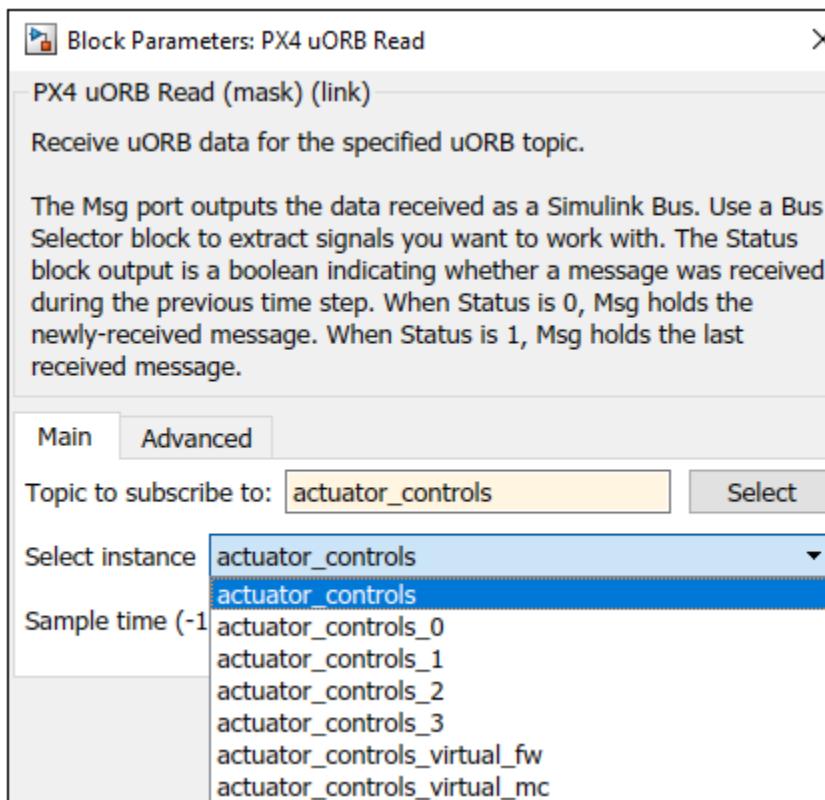
Select instance — Instance of the uORB topic to which the block needs to subscribe
instance of the uORB topic

Select the instance of the uORB topic that the block needs to subscribe to read the uORB data.

Dependencies

This parameter appears only if you select a topic that supports multiple instances, in the **Topic to subscribe to** parameter.

For example, the `actuator_controls` topic supports multiple instances and you can select one instance from the drop-down.



Sample time — Interval at which the block reads values

0.001 (default) | any non-negative value that is a multiple of 0.001 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab**Wait until data received** — Wait until requested data is available

off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When waiting for the data, the **Status** port displays same values from the previous time step. When the data become available, the **Data** port outputs data bytes, and the **Status** port changes to 0.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, Increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received

0.1 (default) | (0, 2^32/1000]

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

Version History

Introduced in R2018b

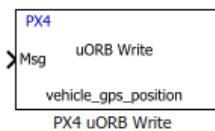
See Also

[PX4 uORB Write](#) | [PX4 uORB Message](#)

PX4 uORB Write

Write uORB data for the specified uORB topic

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 uORB Read and Write Blocks

Description

The PX4 uORB Write block accepts a Simulink nonvirtual bus that corresponds to the specified uORB topic and publishes the message to the uORB network.

On each sample hit, the block converts the **Msg** input from a Simulink bus signal to a uORB message and publishes it. The block does not distinguish whether the input is a new message; but, merely publishes it on every sample hit.

During Connected I/O simulation, this block writes data to the peripherals of the hardware.

Ports

Input

Msg — uORB message, specified as a nonvirtual bus
nonvirtual bus

To specify the type of uORB message, use the parameter, **Topic to Publish to**.

Data Types: bus

Parameters

Main Tab

Topic to publish to — Name of the uORB topic to which the message needs to be published
vehicle_gps_position (default) | one of the uORB topics

Select the name of the uORB topic that the block needs to publish to. Click **Select** to browse topic names available in the **Firmware/msg** folder of the **px4** directory that you downloaded.

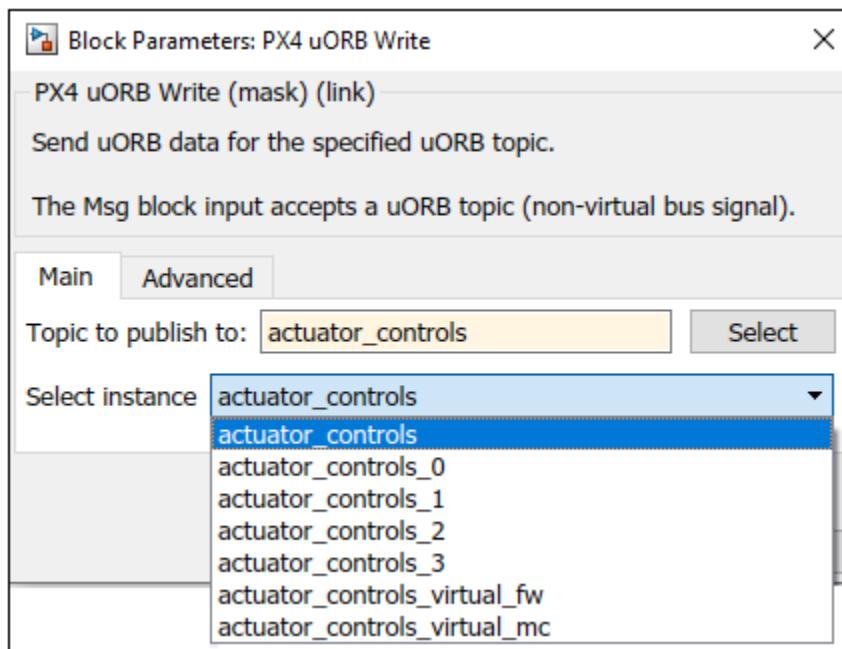
Select instance — Instance of the uORB topic to which the message needs to be published
instance of the uORB topic

Select the instance of the uORB topic that the block needs to publish to.

Dependencies

This parameter appears only if you select a topic that supports multiple instances, in the **Topic to publish to** parameter.

For example, the `actuator_controls` topic supports multiple instances and you can select one instance from the drop-down.



Advanced Tab

Length of publish queue — Maximum number of buffered elements
1 (default) | integer

Enter the maximum number of buffered elements in the publish queue. If the value is 1, no queuing is used to publish the data.

Version History

Introduced in R2018b

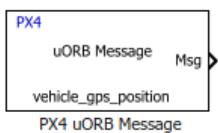
See Also

[PX4 uORB Read](#) | [PX4 uORB Message](#)

PX4 uORB Message

Create a blank message using specified uORB topic

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 uORB Read and Write Blocks

Description

The PX4 uORB Message block creates a Simulink nonvirtual bus with a blank message corresponding to the selected uORB topic. The block creates uORB message bus that work with **PX4 uORB Write** block.

Output

Msg — Blank uORB message

nonvirtual bus

Blank uORB message, returned as a nonvirtual bus. The type of uORB message is specified using the parameter, **Topic**. All elements of the bus are initialized to 0. Connect a Bus Assignment block to modify specific fields in the message.

Data Types: bus

Parameters

Topic — Name of the uORB topic for creating the blank uORB message
vehicle_gps_position (default) | One of the uORB topic

Select the name of the uORB topic for the block to create a blank message. Click **Select** to browse the topic names available in the px4/Firmware/msg directory that you downloaded.

Sample time — Interval between outputs
Inf (default) | numeric scalar

Enter the time interval at which the block provides blank uORB messages.

The default value indicates that the block output never changes. Using this value speeds the simulation and code generation by eliminating the need to recompute the block output. Otherwise, the block outputs a new blank message at each interval of **Sample time**.

Version History

Introduced in R2018b

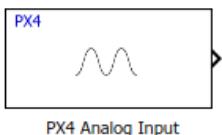
See Also

[PX4 uORB Read](#) | [PX4 uORB Write](#)

PX4 Analog Input

Measure analog voltage applied to an ADC channel

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The PX4 Analog Input block measure the analog voltage at the ADC channels on the PX4 flight controller hardware, and outputs the voltages as a 1-by-12 array. Each value in the array corresponds to the analog voltage at a particular channel number.

For more information about mapping the channel numbers and the voltage array, see “Index Numbers for Analog Channels on Pixhawk Series Controller Boards” on page 2-2.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Port_1 — Voltage values represented as a 1-by-12 array
analog

The voltage values captured at the different ADC channels on the flight controller board.

Connect this output to a Selector block with its `Input port size` parameter value set to 12. You can use the Selector block to specify the index values of the required channel numbers from which you want to read the voltage values.

Data Types: `vector`

Parameters

Sample time — Interval at which the block reads analog voltage values
0.001 (default) | any non-negative value that is a multiple of 0.001 | -1

Enter the time interval at which the block reads voltage values from the ADC channels on the flight controller board.

When you set this parameter to **-1**, Simulink determines the best sample time for the block based on the block context within the model.

Version History

Introduced in R2018b

See Also

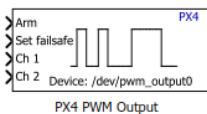
"Index Numbers for Analog Channels on Pixhawk Series Controller Boards" on page 2-2

PX4 PWM Output

Configure PWM outputs for servo motors and ESC control

Note PX4 PWM Output will be removed in a future release. For more information, see “Version History”.

Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The PX4 PWM Output block helps you to configure the PWM output from the PX4 flight controller board. The block accepts time value (in microseconds) that represents the ON period of PWM signal for a particular channel, and passes the same to the corresponding PWM channels on the board.

The PX4 PWM Output block also accepts signals for arming the flight controller.

During Connected IO simulation, this block writes data to the peripherals of the hardware.

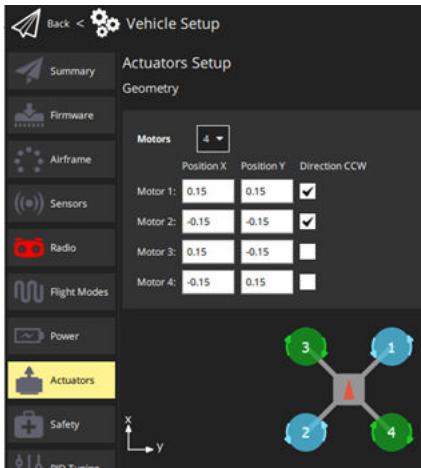
Configure Actuators

For Firmware version 1.14, the PWM & AUX pins need to be configured in QGroundControl before the PWM Output block can be used in Simulink. To do this, QGroundControl version 4.2.3 is needed which can be downloaded from this location.

Note Configuring the actuators is not required for Hardware-in-the-loop or PX4 Host Target simulations.

To configure the actuators, perform these steps.

- 1 Perform the Hardware setup process for your Autopilot.
- 2 Launch QGroundControl 4.2.8 (QGC) and allow it connect to your Autopilot.
- 3 Once QGC is connected, go to **Vehicle Setup -> Actuators** tab.



- 4 Assign the following sequence of Motors against each Main Channel.



It is recommended to change the Minimum PWM value of each channel to 1000 and Maximum PWM value to 2000.

- 5 Assign the following sequence of Motors against each AUX Channel.



Ports

Input

Arm — Arm the PX4 flight controller
0 (default) | 1

Enable signal for arming the PX4 flight controller.

Data Types: Boolean

Set failsafe — Enforce failsafe for PX4 flight controller
0 (default) | 1

Enforce failsafe values even though the failsafe conditions might or might not have occurred. You can specify the failsafe values in the Configuration Parameters dialog box in Simulink.

Note If you are using PX4 firmware 1.12.3, the value at Set failsafe input port is not valid if you use AUX channels for PWM.

Data Types: Boolean

Ch x — Inputs that accept PWM ON values
0 (default) | 1

Connect the Ch x inputs to signals that represent PWM ON values (in microseconds)

Dependencies

The required Ch inputs are selected using the **PWM Channels** parameter.

Data Types: uint16

Parameters

Main tab

Select PWM Device — Channel category to identify the channels that you are going to select
Main (default) | Aux

Select the category (Main or AUX) to identify and select the corresponding channels. After you select this category, you can select the channels in the parameter list, for Main and AUX, separately.

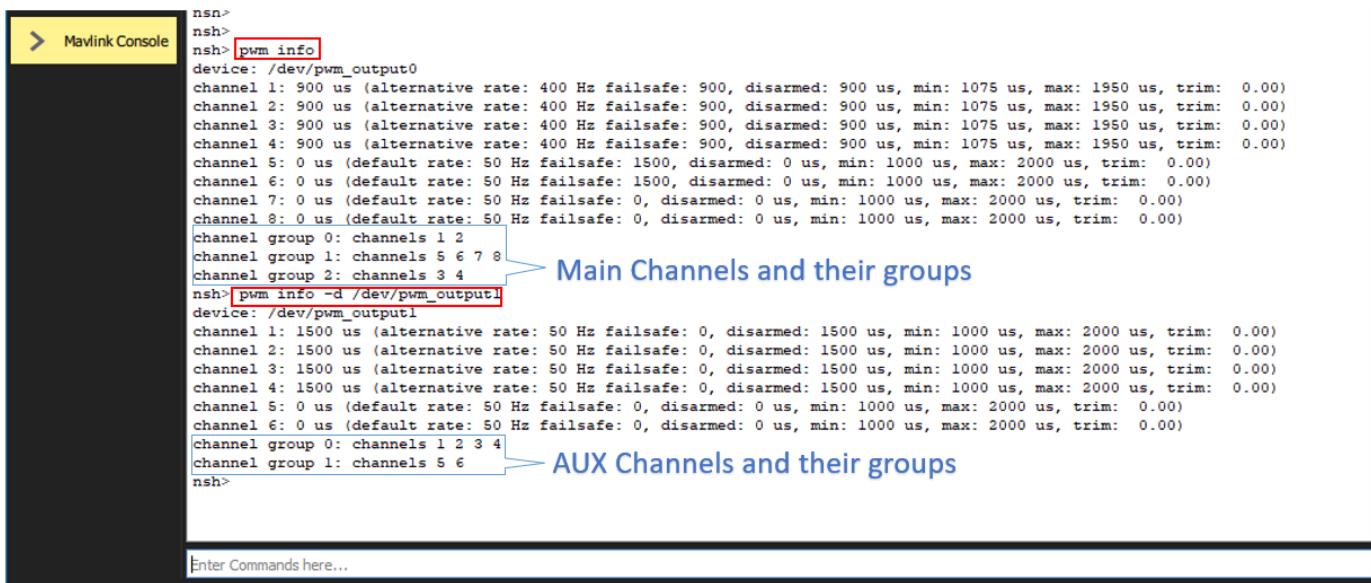
PWM Channels — Channels to which you want to send the PWM ON time values
Channel numbers

Select the channels to which you want to send the values for PWM ON time (the channels to which you connect servo motors or ESC). The Ch inputs to the block appear based on the selection in this parameter list.

Note Ensure that you select all the channels that belong to the same channel group. If you need to connect to PWM channels that are in different groups, select all channels in all the desired groups. Only one PX4 PWM Output block per channel category (Main or AUX) is allowed in the entire Simulink model.

Tip To identify the channel groups in the Pixhawk Series flight controller board, use QGroundControl application on your host computer:

- 1 Open QGroundControl.
- 2 Connect the Pixhawk Series flight controller board to the USB port of the host computer.
- 3 In QGroundControl, go to **Mavlink Console**, and do the following:
 - To identify the Main channels and their groups, enter the command `pwm info`.
 - To identify the Aux channels and their groups, enter the command `pwm info -d /dev/pwm_output1`



```

nsh>
nsh> pwm info
device: /dev/pwm_output0
channel 1: 900 us (alternative rate: 400 Hz failsafe: 900, disarmed: 900 us, min: 1075 us, max: 1950 us, trim: 0.00)
channel 2: 900 us (alternative rate: 400 Hz failsafe: 900, disarmed: 900 us, min: 1075 us, max: 1950 us, trim: 0.00)
channel 3: 900 us (alternative rate: 400 Hz failsafe: 900, disarmed: 900 us, min: 1075 us, max: 1950 us, trim: 0.00)
channel 4: 900 us (alternative rate: 400 Hz failsafe: 900, disarmed: 900 us, min: 1075 us, max: 1950 us, trim: 0.00)
channel 5: 0 us (default rate: 50 Hz failsafe: 1500, disarmed: 0 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 6: 0 us (default rate: 50 Hz failsafe: 1500, disarmed: 0 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 7: 0 us (default rate: 50 Hz failsafe: 0, disarmed: 0 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 8: 0 us (default rate: 50 Hz failsafe: 0, disarmed: 0 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel group 0: channels 1 2
channel group 1: channels 5 6 7 8
channel group 2: channels 3 4
nsh> pwm info -d /dev/pwm_output1
device: /dev/pwm_output1
channel 1: 1500 us (alternative rate: 50 Hz failsafe: 0, disarmed: 1500 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 2: 1500 us (alternative rate: 50 Hz failsafe: 0, disarmed: 1500 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 3: 1500 us (alternative rate: 50 Hz failsafe: 0, disarmed: 1500 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 4: 1500 us (alternative rate: 50 Hz failsafe: 0, disarmed: 1500 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 5: 0 us (default rate: 50 Hz failsafe: 0, disarmed: 0 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel 6: 0 us (default rate: 50 Hz failsafe: 0, disarmed: 0 us, min: 1000 us, max: 2000 us, trim: 0.00)
channel group 0: channels 1 2 3 4
channel group 1: channels 5 6
nsh>
```

Enter Commands here...

Version History

Introduced in R2018b

R2024b: To be removed in a future release

Not recommended starting in R2024b

The PX4 PWM Output function will be removed in a future release. Use the PX4 Actuator Write block instead.

For information to convert the PX4 PWM Output in your Simulink model to a PX4 Actuator Write block, see “Convert PX4 PWM Output Block to PX4 Actuator Write Block”.

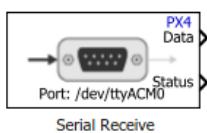
See Also

PX4 Actuator Write

Serial Receive

Read data from UART or USART port on PX4 flight controller

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The Serial Receive block reads data from the Universal Asynchronous Receiver Transmitter (UART) port or the Universal Synchronous and Asynchronous Receiver Transmitter (USART) port on the PX4 flight controller.

The block reads the values from the specified UART or USART port, and outputs the received values as a N-by-1 array. The properties for each port for serial communication are mentioned in the Configuration Parameters dialog box in Simulink (go to **Configuration Parameters > Hardware Implementation > /dev/tty*** pane).

During the external mode simulation, the block outputs the results from the executable running on the target hardware.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the specified serial port of the hardware.

You can choose to read data in blocking or non-blocking mode. For more information, see “Partially receive data from serial port” on page 1-18 and “Receive data from serial port” on page 1-20.

Note If you are using Cube Orange autopilot, then use GPS2 for serial data communication, as there might be issues using TELEM1 or TELEM2 ports.

Ports

Output

Data — Data received from the UART or USART port vector

At each sample time, the **Data** port outputs the values read from the UART or USART port.

The port outputs a data vector of the size that you specify in the **Data length (N)** parameter.

For more information, see “Receive data from serial port” on page 1-20

Data Types: int8 | uint8 | int16 | uint16 | int32 | uint32 | single | double

Status — Determine if the requested data is received at the given time step
scalar

The **Status** port outputs 0 when the length of data received is greater than or equal to the length specified in the **Data length (N)** parameter. A value of 0 indicates a successful read operation.

Otherwise, it outputs 1, indicating that no new data is available.

Dependencies

In non-blocking mode, if the **Output partially received data** option is checked, then the **Status** port is not available

Data Types: uint8

Length — Number of data bytes received
scalar

The **Length** port outputs the number of data bytes in the received message. For more information, see “Partially receive data from serial port” on page 1-18.

Dependencies

This output is available only if the **Output partially received data** option is checked.

Data Types: uint8

Parameters

Main Tab

Port — UART or USART port for serial communication

/dev/ttyACM0 (default) | /dev/ttyS1 | /dev/ttyS2 | /dev/ttyS3 | /dev/ttyS5 | /dev/ttyS6

Enter the name of the UART or USART port on the board from which the block reads data. To view the mapping between UART/USART ports and the labels on the Pixhawk flight controller, see “Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards” on page 3-2.

Data type — Data type of data read from the UART or USART port
uint8 (default) | int8 | uint16 | int16 | uint32 | int32 | single | double

Select the data type in which the block receives data from the UART or USART port.

Data length (N) — Length of data read from the UART or USART port
1 (default) | any integer greater than or equal to 0

Specify the length of data that you want to receive at each sample time.

Sample time — How often to read data from the UART or USART port
0 . 1 (default)

Specify how often the block reads data from the USART port. When you specify this parameter as -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab

Wait until data received — Wait until the requested data is available
off (default) | on

- **on** — When you select this parameter, the read operation runs in the **Blocking mode**. The read operation is blocked while waiting for the requested data to be available. If data is available, the Data port outputs data. If data is not available, the Data port waits for data.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns:

- Increase the time step by using the **Sample time** parameter.
- Reduce the length of data requested by using the **Data length (N)** parameter.
- **off** — When you clear this parameter, the read operation runs in the **Non-blocking mode**. When reading data, if data is not available, the Data port outputs zeroes. The Status port outputs 32, indicating that no new data is available.

Timeout in seconds — Time to wait until the data is received

5 (default) | (0, $2^{32}/1000$)

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

Output partially received data — Output the data even if the complete data is not available at the sample time

off (default) | on

- **on** — When you select this parameter, the Data port outputs partially received data at the sample time.
- **off** — When you clear this parameter, the Data port waits for the next sample time so that the data for the whole Data Length (N) is received, and then only provides the output.

Dependencies

This parameter is visible only if the **Wait until data received** option is unchecked (non-blocking mode). For more information, see “Partially receive data from serial port” on page 1-18.

Partially receive data from serial port

This example describes the values at the Length output port when the length of the messages received is less than, greater than, or equal to the length of requested data. This section explains the values at the output ports, with **Data type** set to uint8 and **Data size (N)** set to 4 in non-blocking mode with the **Output partially received data** option checked.

- Length of data received = Data length (N): The Data port outputs the message as a data vector of the size specified in the **Data size (N)** parameter.

Suppose that the **Data size (N)** parameter specified is 4 and the length of the message received is also 4.

In this case, the **Data** port outputs a data vector of size 4 filled with the data bytes of the message, and the **Length** port outputs 4.

$$\text{Length of message received} = \text{Data size (N)} = 4$$

Data	Data(Byte1)	Data(Byte2)	Data(Byte3)	Data(Byte4)
-------------	-------------	-------------	-------------	-------------

- Length of data received < Data length (N): The **Data** port outputs the message as a data vector of the size specified in the **Data size (N)** parameter. All the empty spaces in the vector are filled with zeroes.

Suppose that the **Data size (N)** parameter specified is 4 and the length of data received is 3 bytes.

In this case, the **Data** port outputs a data vector of size 4. The first three data bytes in the vector are the bytes from the received message; the remaining space is filled with zero. The **Length** port outputs 3.

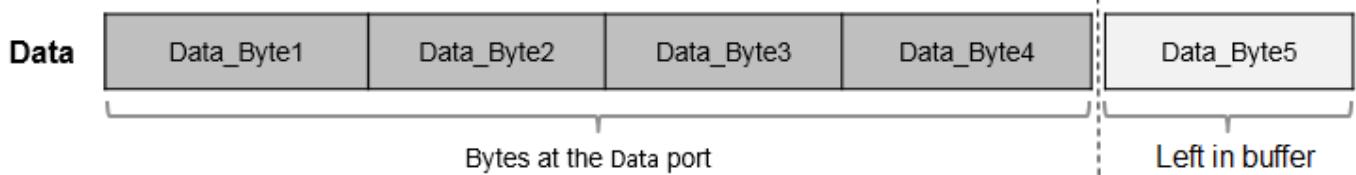
$$\text{Length of message received} < \text{Data size (N)}$$

Data	Data(Byte1)	Data(Byte2)	Data(Byte3)	0
-------------	-------------	-------------	-------------	---

- Length of data received > Data length (N): The **Data** port outputs a data vector of the size specified in the **Data size (N)** parameter. The vector contains only the first N data bytes from the message. The remaining data bytes are left in the buffer.

Suppose that the **Data length (N)** parameter specified is 4 and the length of data received is 5 bytes. In this case, the port outputs a data vector of size 4. The vector contains only the first 4 bytes from the received data. The remaining data byte is left in the buffer. The **Length** port outputs 4.

$$\text{Length of message received} > \text{Data size (N)}$$



Receive data from serial port

This example describes the values at the **Status** output port when the length of the messages received is less than, greater than, or equal to the length of requested data. This section explains the values at the output ports, with **Data type** set to `uint8` and **Data size (N)** set to 4 in blocking and non-blocking modes.

- Length of data received = Data length (N): The **Data** port outputs the message as a data vector of the size specified in the **Data size (N)** parameter.

Suppose that the **Data size (N)** parameter specified is 4 and the length of the message received is also 4.

In this case, for both blocking and non-blocking modes, the **Data** port outputs a data vector of size 4 filled with the data bytes of the message, and the **Status** port outputs 0.

Length of message received = Data size (N) = 4

Data	Data_Byte1	Data_Byte2	Data_Byte3	Data_Byte4
-------------	------------	------------	------------	------------

- Length of data received < Data length (N): The **Data** port outputs the message as a data vector of the size specified in the **Data size (N)** parameter. All the empty spaces in the vector are filled with zeroes.

Suppose that the **Data size (N)** parameter specified is 4 and the length of data received is 3 bytes.

For non-blocking mode, the **Data** port outputs a data vector of size 4. The first three data bytes in the vector are the bytes from the received message. The remaining space is filled with zero. The **Status** port outputs 1.

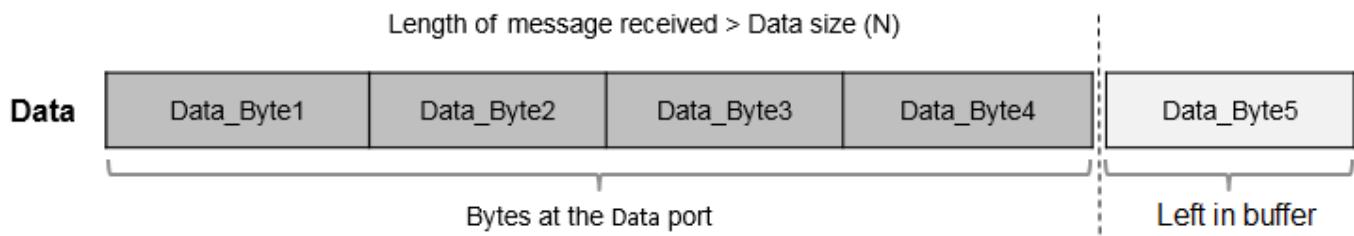
For blocking mode, the block waits till all the data bytes are received within the time specified in the **Timeout** parameter. If the fourth data byte is not received within the timeout, the **Data** outputs zeroes, and the **Status** port outputs 1. The three received data bytes are left in the buffer.

Length of message received < Data size (N)

Data	0	0	0	0
-------------	---	---	---	---

- Length of data received > Data length (N): The **Data** port outputs a data vector of the size specified in the **Data size (N)** parameter. The vector contains only the first N data bytes from the message. The remaining data bytes are left in the buffer.

Suppose that the **Data length (N)** parameter specified is 4 and the length of data received is 5 bytes. In this case, for both blocking and non-blocking modes, the port outputs a data vector of size 4. The vector contains only the first 4 bytes from the received data. The remaining data byte is left in the buffer. The **Status** port outputs zero.



Version History

Introduced in R2018b

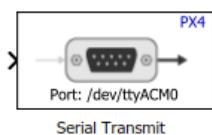
See Also

[Serial Transmit](#) | “Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards” on page 3-2

Serial Transmit

Send serial data to UART or USART port

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The Serial Transmit block sends serial data to the Universal Asynchronous Receiver Transmitter (UART) port or the Universal Synchronous and Asynchronous Receiver Transmitter (USART) port on the PX4 flight controller.

The block accepts data as an N-by-1 or 1-by-N array and sends the same to the specified UART or USART port. The properties for each port for serial communication are mentioned in the Configuration Parameters dialog box in Simulink (go to **Configuration Parameters > Hardware Implementation > /dev/tty*** pane).

During Connected I/O simulation, this block writes data to the specified serial port of the hardware.

The block inherits the data type from the signal at the input port.

Note If you are using Cube Orange autopilot, then use GPS2 for serial data communication, as there might be issues using TELE1 or TELE2 ports.

Ports

Input

Data — Data to be sent to the UART or USART port vector

The port accepts values as an N-by-1 or 1-by-N array.

Data Types: int8 | uint8 | int16 | uint16 | int32 | uint32 | single | double | Boolean

Parameters

Port — UART or USART port for serial communication

/dev/ttyACM0 (default) | /dev/ttyS1 | /dev/ttyS2 | /dev/ttyS3 | /dev/ttyS5 | /dev/ttyS6

Enter the name of the UART or USART port on the PX4 flight controller through which the block transmits the accepted data. To view the mapping between UART/USART ports and the labels on the Pixhawk flight controller, see “Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards” on page 3-2.

Version History

Introduced in R2018b

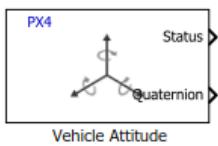
See Also

[Serial Receive](#) | “Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards” on page 3-2

Vehicle Attitude

Read vehicle_odometry uORB topic and obtain attitude measurements

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The Vehicle Attitude block reads the vehicle_odometry uORB topic and outputs the attitude measurements from the Pixhawk® hardware. The block outputs the vehicle attitude in both roll, pitch, and yaw, and quaternion (NED) form.

On each simulation step, the block checks if a new message is available on the vehicle_odometry topic. If a new message is available, the block retrieves the message and outputs it, and the timestamp is updated. If a new message is not available, the block outputs the last updated uORB message and the timestamp is not updated. If a message has not been received since the start of the simulation, the outputs are zeroes.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Status — Status of the received uORB data

0 | 1

The Status output indicates if the vehicle_odometry uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the outputs is the latest, and a value of 1 indicates that the uORB data was not updated during the previous time step.

This output can be used to trigger subsystems for processing new messages received in the uORB network.

Data Types: Boolean

Roll rate — Bias corrected angular velocity about x-axis

scalar | rad/s

The bias corrected angular velocity about x-axis, as read from the vehicle_odometry uORB topic.

Data Types: single

Pitch rate — Bias corrected angular velocity about y-axis
scalar | rad/s

The bias corrected angular velocity about y-axis, as read from the vehicle_odometry uORB topic.

Data Types: single

Yaw rate — Bias corrected angular velocity about z-axis
scalar | rad/s

The bias corrected angular velocity about z-axis, as read from the vehicle_odometry uORB topic.

Data Types: single

Quaternion — Attitude of the vehicle on which the PX4 flight controller is mounted, in quaternion form
1-by-4 array

The attitude of the vehicle on which the PX4 flight controller is mounted, in quaternion form, as read from the vehicle_odometry uORB topic.

Data Types: single

Timestamp — Timestamp of the received uORB data
time in microseconds

Represents the time (since the PX4 system start) when the vehicle_odometry topic was last updated.

Data Types: double

Parameters

Main tab

Roll — Enable roll value as one of the outputs
off (default) | on

Pitch — Enable pitch value as one of the outputs
off (default) | on

Yaw — Enable yaw value as one of the outputs
off (default) | on

Quaternion — Enable quaternion value as one of the outputs
on (default) | off

Timestamp — Enable timestamp value as one of the outputs
off (default) | on

Sample time — Interval at which the block reads values
1/250 (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab

Wait until data received — Wait until requested data is available
off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When the data is available, the block outputs the selected values, and the **Status** port is set to 0.

The board waits for the message in Blocking Mode for the time mentioned in the **Timeout in seconds**. If the data is not received before the timeout, then the Status port outputs 1 and the block outputs the last received value.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received
1e-3 (default) | (0, 2^32/1000]

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

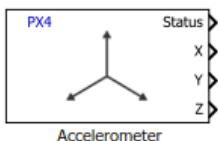
Version History

Introduced in R2018b

Accelerometer

Read sensor_accel uORB topic and obtain three dimensional linear acceleration

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The Accelerometer block reads the sensor_accel uORB topic and outputs the linear acceleration of the PX4 Autopilot board along the x, y and z axis. The block also outputs the timestamp.

On each simulation step, the block checks if a new message is available on the sensor_accel topic. If a new message is available, the block retrieves the message and outputs it, and the timestamp is updated. If a new message is not available, the block outputs the last updated uORB message and the timestamp is not updated. If a message has not been received since the start of the simulation, the outputs are zeroes.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Status — Status of the received uORB data

0 | 1

The Status output indicates if the sensor_accel uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the outputs is the latest, and a value of 1 indicates that the uORB data was received during the previous time step.

This output can be used to trigger subsystems for processing new messages received in the uORB network.

Data Types: Boolean

X — Linear acceleration along the x-axis
scalar (m/s^2)

The linear acceleration along the x-axis of the PX4 Autopilot board, as read from the sensor_accel uORB topic.

Data Types: **single**

Y — Linear acceleration along the y-axis
scalar (m/s^2)

The linear acceleration along the y-axis of the PX4 Autopilot board, as read from the sensor_accel uORB topic.

Data Types: **single**

Z — Linear acceleration along the z-axis
scalar (m/s^2)

The linear acceleration along the z-axis of the PX4 Autopilot board, as read from the sensor_accel uORB topic.

Data Types: **single**

Timestamp — Timestamp of the received uORB data
time in microseconds

Represents the time (since the PX4 system start) when the sensor_accel topic was last updated.

Data Types: **double**

To see more output options for the sensor_accel uorb topic, use the PX4 uORB Read block, subscribe to the topic, and select other output options.

Parameters

Main tab

Timestamp — Enable timestamp value as one of the outputs
off (default) | **on**

Select this parameter to output the timestamp.

Sample time — Interval at which the block reads values
1/250 (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab

Wait until data received — Wait until requested data is available
off (default) | **on**

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When the data is available, the block outputs the selected values, and the **Status** port is set to 0.

The board waits for the message in Blocking Mode for the time mentioned in the **Timeout in seconds**. If the data is not received before the timeout, then the Status port outputs 1 and the block outputs the last received value.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received

1e-3 (default) | (0, 2³²/1000]

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

Version History

Introduced in R2018b

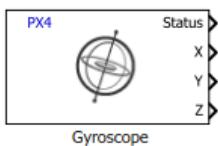
See Also

PX4 uORB Read | Gyroscope | Magnetometer

Gyroscope

Read sensor_gyro uORB topic and obtain three dimensional rate of rotation

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The Gyroscope block reads the sensor_gyro uORB topic and outputs the rate of rotation of the PX4 Autopilot board along the x, y and z axis. The block also outputs the timestamp.

On each simulation step, the block checks if a new message is available on the sensor_gyro topic. If a new message is available, the block retrieves the message and outputs it, and the timestamp is updated. If a new message is not available, the block outputs the last updated uORB message and the timestamp is not updated. If a message has not been received since the start of the simulation, the outputs are zeroes.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Status — Status of the received uORB data

0 | 1

The Status output indicates if the sensor_gyro uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the outputs is the latest, and a value of 1 indicates that the uORB data was received during the previous time step.

This output can be used to trigger subsystems for processing new messages received in the uORB network.

Data Types: Boolean

X — Angular velocity along the x-axis
scalar | rad/s

The angular velocity along the x-axis of the PX4 Autopilot board, as read from the sensor_gyro uORB topic.

Data Types: **single**

Y — Angular velocity along the y-axis
scalar | rad/s

The angular velocity along the y-axis of the PX4 Autopilot board, as read from the sensor_gyro uORB topic.

Data Types: **single**

Z — Angular velocity along the z-axis
scalar | rad/s

The angular velocity along the z-axis of the PX4 Autopilot board, as read from the sensor_gyro uORB topic.

Data Types: **single**

Timestamp — Timestamp of the received uORB data
time in microseconds

Represents the time (since the PX4 system start) when the sensor_gyro topic was last updated.

Data Types: **double**

To see more output options for the sensor_gyro uorb topic, use the PX4 uORB Read block, subscribe to the topic, and select other output options.

Parameters

Main tab

Timestamp — Enable timestamp value as one of the outputs
off (default) | on

Select this parameter to output the timestamp.

Sample time — Interval at which the block reads values
1/250 (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab

Wait until data received — Wait until requested data is available
off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When the data is available, the block outputs the selected values, and the **Status** port is set to 0.

The board waits for the message in Blocking Mode for the time mentioned in the **Timeout in seconds**. If the data is not received before the timeout, then the Status port outputs 1 and the block outputs the last received value.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received

1e-3 (default) | (0, 2³²/1000]

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

Version History

Introduced in R2018b

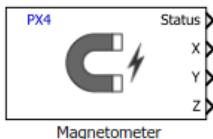
See Also

[PX4 uORB Read](#) | [Accelerometer](#) | [Magnetometer](#)

Magnetometer

Read sensor_mag uORB topic and obtain three dimensional magnetic field

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The Magnetometer block reads the sensor_mag uORB topic and outputs the magnetic field of the PX4 Autopilot board along the x, y and z axis. The block also outputs the timestamp.

On each simulation step, the block checks if a new message is available on the sensor_mag topic. If a new message is available, the block retrieves the message and outputs it, and the timestamp is updated. If a new message is not available, the block outputs the last updated uORB message and the timestamp is not updated. If a message has not been received since the start of the simulation, the outputs are zeroes.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Status — Status of the received uORB data

0 | 1

The Status output indicates if the sensor_mag uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the outputs is the latest, and a value of 1 indicates that the uORB data was received during the previous time step.

This output can be used to trigger subsystems for processing new messages received in the uORB network.

Data Types: Boolean

X — Magnetic field along the x-axis
scalar | gauss

The magnetic field along the x-axis of the PX4 Autopilot board, as read from the sensor_mag uORB topic.

Data Types: **single**

Y — Magnetic field along the y-axis
scalar | gauss

The magnetic field along the y-axis of the PX4 Autopilot board, as read from the sensor_mag uORB topic.

Data Types: **single**

Z — Magnetic field along the z-axis
scalar | gauss

The magnetic field along the z-axis of the PX4 Autopilot board, as read from the sensor_mag uORB topic.

Data Types: **single**

Timestamp — Timestamp of the received uORB data
time

Represents the time (since the PX4 system start) when the sensor_mag uORB topic was last updated.

Data Types: **double**

To see more output options for the sensor_mag uorb topic, use the PX4 uORB Read block, subscribe to the topic, and select other output options.

Parameters

Main tab

Timestamp — Enable timestamp value as one of the outputs
off (default) | on

Select this parameter to output the timestamp.

Sample time — Interval at which the block reads values
1/250 (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab

Wait until data received — Wait until requested data is available
off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When the data is available, the block outputs the selected values, and the **Status** port is set to 0.

The board waits for the message in Blocking Mode for the time mentioned in the **Timeout in seconds**. If the data is not received before the timeout, then the Status port outputs 1 and the block outputs the last received value.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received

1e-3 (default) | (0, 2³²/1000]

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

Version History

Introduced in R2018b

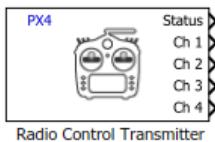
See Also

PX4 uORB Read | Accelerometer | Gyroscope

Radio Control Transmitter

Read input_rc uORB topic to obtain data from Radio Control Transmitter

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The Radio Control Transmitter block reads the input_rc uORB topic to obtain the signals sent from a Radio Control Transmitter at the specified channels.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Ch x — Measured pulse width value from the specified radio communication channel
scalar | time in the range [1, 2] ms

Pulse width values obtained from the selected channels of the Radio Control Transmitter. Connect these output values to the flight control logic in the Simulink model.

Data Types: uint16

Num Ch — Count of channels that are actually being seen
scalar

The count of channels that are actually being seen based on the communication with the Radio Control Transmitter

Dependencies

This output appears only if you have selected the **Number of channels actually being seen** parameter.

Data Types: uint32

RSSI — RSSI value from the RC Transmitter
negative | 0 | positive

RSSI (Receive Signal Strength Indicator) value obtained from the Radio Control Transmitter.

The value is negative for an undefined signal, 0 for no signal, and positive for full reception.

Dependencies

This output appears only if you have selected the **Receive Signal Strength Indicator (RSSI)** parameter.

Data Types: int32

RC Source — Input source of RC communication
scalar

Identify the input source of RC communication. The value ranges from 0 to 13 (these correspond to the same values as mentioned in the `input_rc.msg` file in `Firmware/msg` directory).

Dependencies

This output appears only if you have selected the **Radio control input source** parameter.

Data Types: uint8

Last valid time — Timestamp of the last valid reception
time

Timestamp of the last valid reception from the Radio Control Transmitter.

Dependencies

This output appears only if you have selected the **Last valid reception time** parameter.

Data Types: uint64

Timestamp — Current timestamp
time

Current timestamp of the value received from the Radio Control Transmitter.

Dependencies

This output appears only if you have selected the **Timestamp** parameter.

Data Types: double

Status — Status of the data read from `input_rc` uORB topic
integer in [0 , 7]

The Status output indicates if the data was received during a previous time step or not. A value of 0 indicates that the data at the Ch output is the latest and the communication with the Radio Control Transmitter is valid.

A value between 1 and 7 indicates the following:

- No value was read by the uORB topic during the current time step
- Communication loss with Radio Control Transmitter
- Radio Control Transmitter went to failsafe mode

This output can be used to trigger subsystems for processing the new data received from the RC Transmitter.

Data Types: uint8

Parameters

Main tab

Channel Selection — Radio channels from which you want to receive values
channels

Select the channels from which you want to receive values, as read from the Radio Control Transmitter. The Ch outputs of the block appear based on the selection in this parameter list.

Number of channels actually being seen — Count of channels that are actually being seen
off (default) | on

Select this option to obtain the count of channels that are actually being seen based on the communication with the Radio Control Transmitter.

Receive Signal Strength Indicator (RSSI) — RSSI value from the RC Transmitter
off (default) | on

Select this option to obtain the RSSI value as received from the Radio Control Transmitter

Radio control input source — Input source of RC communication
off (default) | on

Select this option to identify the input source of RC communication.

Last valid reception time — Timestamp of the last valid reception
off (default) | on

Select this option to obtain the timestamp of the last valid reception from the Radio Control Transmitter.

Timestamp — Current timestamp
off (default) | on

Select this option to obtain the current timestamp of the channel value from the Radio Control Transmitter.

Sample time — Interval at which the block reads values
1/250 seconds (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the RC Transmitter.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced tab

Wait until data received — Wait until requested data is available
off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received

5 (default) | (0, $2^{32}/1000$)

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

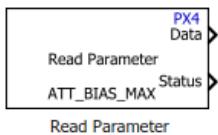
Version History

Introduced in R2018b

Read Parameter

Read PX4 system parameters

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots

Description

The Read Parameter block reads the specified PX4 system parameter, and provides its value as output.

The block supports all parameters listed in PX4 Parameter Reference page. You can also use custom parameters that you have developed using the guidelines.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Data — Value of the parameter
scalar

Value of the specified PX4 system parameter

Data Types: `int32` | `float`

Status — Status of the received parameter value
`0` | `1`

The Status output indicates if the parameter that you specified is valid or not. A value of `0` indicates that the parameter handle is valid, and a value of `1` indicates that the parameter handle is invalid.

Data Types: Boolean

Parameters

Parameter Name — Name of the PX4 parameter
`ATT_BIAS_MAX` (default) | One of the parameter

Enter the name of the PX4 system parameter.

Note The **Parameter Name** that you enter in this field must match one of the PX4 parameters; otherwise, the Status port outputs 1, indicating that the parameter handle is invalid.

Data Type — Data type of the PX4 parameter
FLOAT (default) | int32

Select the data type of the parameter that you entered in the **Parameter Name** field.

Note If the **Data type** that you select in this field does not match the actual data type of the PX4 parameter, the values read may lead to data type mismatch.

Sample time — Interval at which the block reads values
0.5 (default) | any non-negative value that is a multiple of 0.5 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

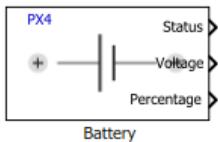
Version History

Introduced in R2018b

Battery

Read battery_status uORB topic and obtain details about the battery's state

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The Battery block reads the `battery_status` uORB topic and outputs the details about the state of the battery connected to PX4 flight controller. The block also outputs the timestamp.

On each simulation step, the block checks if a new message is available on the `battery_status` topic. If a new message is available, the block retrieves the message and converts it to the output values. If a new message is not available, the outputs are based on the last received uORB message. If a message has not been received since the start of the simulation, the outputs are zeroes.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Status — Status of the received uORB data

0 | 1

The Status output indicates if the `battery_status` uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the outputs is the latest, and a value of 1 indicates that the uORB data was received during the previous time step.

This output can be used to trigger subsystems for processing new messages received in the uORB network.

Data Types: Boolean

Voltage — Voltage of battery
scalar (volts)

The output voltage of battery, as read from the `battery_status` uORB topic.

Data Types: single

Current — The output current that the battery is delivering scalar (amperes)

The output current, as read from the battery_status uORB topic.

Data Types: single

Percentage — Percentage of battery life remaining scalar

The percentage of battery life remaining, as read from the battery_status uORB topic.

Data Types: single

Warning — Battery warning status scalar

The battery warning status, as read from the battery_status uORB topic.

Status	Description
0	No battery low voltage warning active
1	Warning of low voltage
2	Critical voltage, return / abort immediately
3	Immediate landing required
4	The battery has failed completely

Data Types: uint8

Timestamp — Timestamp of the received uORB data time

The timestamp of the battery_status uORB topic read from the uORB network.

Data Types: double

Parameters

Main tab

Voltage — Enable voltage value as one of the outputs
on (default) | off

Current — Enable current value as one of the outputs
off (default) | on

Battery percentage — Enable battery percentage as one of the outputs
on (default) | off

Battery status warning — Enable battery status warning as one of the outputs
off (default) | on

Timestamp — Enable timestamp value as one of the outputs
off (default) | on

Sample time — Interval at which the block reads values

1/250 (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab**Wait until data received** — Wait until requested data is available

off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When waiting for the data, the **Status** port displays same values from the previous time step. When the data become available, the block outputs the selected values, and the **Status** port changes to 0.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Timeout in seconds — Time to wait until the data is received

1e-3 (default) | (0, 2^32/1000]

Specify the amount of time that the block waits for the data during each time step, if the **Wait until data received** parameter is selected. If timeout occurs, the read operation is aborted.

Dependencies

This parameter appears only when you select the **Wait until data received** parameter.

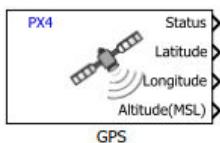
Version History

Introduced in R2018b

GPS

Read vehicle_gps_position uORB topic and obtain GPS coordinates

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Sensor Blocks

Description

The GPS block reads the vehicle_gps_position uORB topic and outputs the GPS coordinates of the GPS receiver connected to the PX4 Autopilot. The block outputs latitude, longitude and altitude, by default, and it also provides the option to add more outputs that are based on the GPS coordinates.

On each simulation step, the block checks if a new message is available on the vehicle_gps_position topic. If a new message is available, the block retrieves the message and outputs it, and the timestamp is updated. If a new message is not available, the block outputs the last updated uORB message and the timestamp is not updated. If a message has not been received since the start of the simulation, the outputs are zeroes.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Status — Status of the received uORB data

0 | 1

The Status output indicates if the vehicle_gps_position uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the outputs is the latest, and a value of 1 indicates that the uORB data was not updated during the previous time step.

This output can be used to trigger subsystems for processing new messages received in the uORB network.

Data Types: Boolean

Latitude — Latitude of the PX4 Autopilot

scalar | 1e-7 degrees

The latitude coordinate based on the current position of the GPS receiver connected to PX4 Autopilot, as read from the vehicle_gps_position uORB topic.

Data Types: int32

Longitude — Longitude of the PX4 Autopilot
scalar | 1e-7 degrees

The longitude coordinate based on the current position of the GPS receiver connected to PX4 Autopilot, as read from the vehicle_gps_position uORB topic.

Data Types: int32

Altitude(MSL) — Altitude of the PX4 Autopilot above Mean Sea Level (MSL)
scalar | millimeters

The altitude coordinate based on the current position of the GPS receiver above Mean Sea Level, as read from the vehicle_gps_position uORB topic.

Data Types: int32

Speed — Ground speed of the PX4 Autopilot
scalar | meters/sec

The ground speed coordinate from the GPS receiver connected to PX4 Autopilot, as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **Ground speed (m/s)** parameter.

Data Types: float32

Course — Course of the PX4 Autopilot over ground
scalar | radians

The course over ground (not the heading, but the direction of movement) value within (- π to π), based on the direction of movement of the GPS receiver connected to PX4 Autopilot, as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **Course (rad)** parameter.

Data Types: float32

Fix Type — Fix type of the GPS receiver
scalar

The type of signal or technique being used by the GPS receiver to determine the location, as read from the vehicle_gps_position uORB topic. The following table represents the values for this signal:

Value	Type of Fix
0-1	No fix
2	2D fix
3	3D fix
4	RTCM code differential
5	Real-Time Kinematic, float

Value	Type of Fix
6	Real-Time Kinematic, fixed
8	Extrapolated. Some applications will not use the value of this field unless it is at least two, so always correctly fill in the fix.

Dependencies

This output appears only if you have selected the **GPS fix type** parameter.

Data Types: uint8

Num Sats — Number of satellites used for global positioning
scalar

The number of satellites used by the GPS receiver to determine the GPS coordinates, as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **Number of satellites** parameter.

Data Types: uint8

UTC Time — UTC timestamp in Unix time obtained from the GPS receiver
UTC (in microseconds)

UTC timestamp in Unix time (in microseconds) obtained from the GPS receiver, as read from the vehicle_gps_position uORB topic. The value might be unavailable right after cold start, indicated by a value of 0.

Dependencies

This output appears only if you have selected the **Unix timestamp UTC (us)** parameter.

Data Types: uint64

DOP — Horizontal and vertical dilution of precisions used by the GPS receiver
2-by-1 vector

The horizontal and vertical dilution of precisions (DOP) used by the GPS receiver, as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **Dilution of precision** parameter.

Data Types: float32

NED velocity — North, east, and down velocity of the PX4 Autopilot
3-by-1 vector | meters/sec

The north, east, and down velocity measured by the GPS receiver connected to the PX4 Autopilot, as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **NED velocity** parameter.

Data Types: float32

Altitude(Ellipsoid) — Altitude of the PX4 Autopilot above Ellipsoid
scalar | millimeters

The altitude coordinate based on the current position of the GPS receiver based on ellipsoidal model (1e-3 meters above Ellipsoid), as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **Altitude above Ellipsoid** parameter.

Data Types: int32

Jamming ind — Jamming indicator of the GPS receiver
scalar

The jamming indicator value of the GPS receiver, which indicates if jamming of signals is occurring or not, as read from the vehicle_gps_position uORB topic.

Dependencies

This output appears only if you have selected the **Jamming indicator** parameter.

Data Types: int32

Timestamp — Timestamp of the received uORB data since system start
microseconds

Represents the time since the PX4 system start when the vehicle_gps_position uORB topic was last updated.

Dependencies

This output appears only if you have selected the **PX4 timestamp (us)** parameter.

Data Types: uint64

Parameters

Main tab

Latitude (1e-7 degrees) — Enable latitude value as one of the outputs
on (default) | off

Longitude (1e-7 degrees) — Enable longitude value as one of the outputs
on (default) | off

Altitude above MSL (mm) — Enable altitude (above MSL) as one of the outputs
off (default) | on

Ground speed (m/s) — Enable ground speed value as one of the outputs
off (default) | on

Course (rad) — Enable course value as one of the outputs
off (default) | on

GPS fix type — Enable GPS fix type value as one of the outputs
off (default) | on

Number of satellites — Enable number of satellites as one of the outputs
off (default) | on

Unix timestamp UTC (us) — Enable UTC timestamp value as one of the outputs
off (default) | on

Sample time — Interval at which the block reads values
1/250 (default) | any non-negative value that is a multiple of 1/250 | -1

Enter the time interval at which the block reads values from the uORB network.

When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Advanced Tab

Wait until data received — Wait until requested data is available
off (default) | on

- When you select this option, the read operation runs in Blocking Mode. The read operation is blocked while waiting for the requested data to be available. When the data is available, the block outputs the selected values, and the **Status** port is set to 0.

The board waits for the message in Blocking Mode for the time mentioned in the **Timeout in seconds**. If the data is not received before the timeout, then the Status port outputs 1 and the block outputs the last received value.

A task overrun occurs if the target hardware is still waiting for the data to be available when the next read operation is scheduled to begin.

To fix overruns, increase the time step by using the **Sample time** parameter.

- When you clear this option, the read operation runs in Non-Blocking Mode. In this mode, the block does not wait for the requested data to be available.

Dilution of precision — Enable dilution of precision value as one of the outputs
off (default) | on

NED velocity — Enable NED velocity value as one of the outputs
off (default) | on

Altitude above Ellipsoid — Enable altitude above ellipsoid value as one of the outputs
off (default) | on

Jamming indicator — Enable jamming indicator value as one of the outputs
off (default) | on

PX4 timestamp (us) — Enable PX4 timestamp value as one of the outputs
off (default) | on

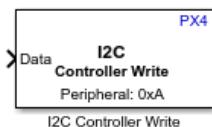
Version History

Introduced in R2020a

I2C Controller Write

Write data to I2C peripheral device or I2C peripheral device register

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots

Description

The I2C Controller Write block writes data to an I2C peripheral device that is connected to the board. Using this block, you can write data to a specific register on the I2C peripheral device.

To view the mapping between the Bus number and the port label on hardware board, click **View I2C Bus map**. For more information on the Bus mapping, see “I2C Bus Port Numbers for Labels on PX4 Autopilots” on page 5-2.

During Connected I/O simulation, this block writes data to the peripherals of the hardware.

Ports

Input

Data — Data to write to I2C peripheral device
scalar | vector

Data Types: int8 | uint8 | int16 | uint16 | int32 | uint32 | single | double

Output

Status — Status of write operation
scalar

When you select the **Output error status** parameter, an output port, labeled as **Status**, becomes available.

The port outputs one of these values:

- 0 for a successful write operation.
- 1 for a failure in opening the bus.
- 2 for a failure in write operation.

Data Types: uint8

Parameters

I2C Bus — I2C bus on the board to which the I2C peripheral device is connected to
1 (default) | nonnegative integer

Specify the I2C bus on the board to write data to the I2C peripheral device.

Peripheral address — I2C peripheral address to write data
10 (default)

Specify the I2C peripheral address to write the data. The I2C peripheral device address is a 7-bit address.

Specify this address as an integer or in hexadecimal format by using `hex2dec()`, for example, `0x61`.

Peripheral byte order — Byte ordering that your I2C peripheral device supports
BigEndian (default) | LittleEndian

The 2-byte ordering options are:

- **BigEndian** - The most significant byte is sent first over the I2C bus.
- **LittleEndian** - The least significant byte is sent first over the I2C bus.

Enable register access — Option to write to specific I2C peripheral register
on (default) | off

When you select this parameter, the block writes data to the I2C peripheral register that you specify in the **Peripheral register address** parameter.

Peripheral register address — I2C peripheral register address to write data
uint8(1) (default)

Specify the I2C peripheral register address to write the data.

Specify this address in 8-bit unsigned integer or in hexadecimal format. To specify the address in 8-bit unsigned integer, use `uint8()`, for example, '`uint8(20)`'. To specify the address in hexadecimal format, use `hex2dec()`, for example, `0x20`.

Dependencies

This parameter appears only when you select **Enable register access** parameter.

Output error status — Option to display the error status during data transmission
off (default) | on

When you select this parameter, an output port, labeled as **Status**, becomes available.

The port outputs the status of the write operation.

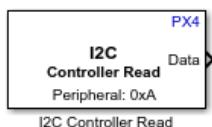
See Also

I2C Controller Read

I2C Controller Read

Read data from I2C peripheral device or I2C peripheral device register

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots

Description

The I2C Controller Read block reads serial data from an I2C peripheral device that is connected to the board. Using this block, you can read data from a specific register on the I2C peripheral device.

To view the mapping between the Bus number and the port label on hardware board, click **View I2C Bus map**. For more information on the Bus mapping, see “I2C Bus Port Numbers for Labels on PX4 Autopilots” on page 5-2.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Data — Data from I2C peripheral device vector

The port reads vector data from an I2C device that is connected to the board. The size of the data is the value that you specify in the **Data size** parameter.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `int64` | `uint64` | `single` | `double`

Status — Status of read operation scalar

When you select the **Output error status** parameter, an output port, labeled as **Status**, becomes available.

The port outputs one of these values:

- 0 for a successful read operation.
- 1 for a failure in opening the bus.

- 2 for a failure in read operation.

Data Types: uint8

Parameters

I2C Bus — I2C bus to read data

1 (default)

Specify the I2C bus on the board to read data from the I2C peripheral device.

Peripheral address — I2C register peripheral address from which to read data

10 (default) | nonnegative integer

Specify the I2C peripheral address to read the data. You can specify this address in hexadecimal format `hex2dec()`, for example, `0x20`.

Peripheral byte order — Byte ordering that your I2C peripheral device supports

BigEndian (default) | LittleEndian

The 2-byte ordering options are:

- BigEndian -The most significant byte is sent first over the I2C bus.
- LittleEndian - The least significant byte is sent first over the I2C bus.

Enable register access — Option to read from a specific I2C peripheral register

on (default) | off

When you select this parameter, the block reads data from the I2C peripheral register that you specify in the **Peripheral register address** parameter.

Peripheral register address — The I2C peripheral register address to read data

2 (default) | integer between 0 and 255

Specify the I2C register address to read the data.

Specify this address as an integer or in hexadecimal format by using `hex2dec()`, for example, `0x20`.

Dependencies

To enable this parameter, select the **Enable register access** parameter.

Data type — Data type of data from I2C peripheral device

uint8 (default) | int8 | int16 | uint16 | int32 | uint32 | single | double

Select the data type in which you want to read from the I2C peripheral device.

Data size (N) — Data size to read from I2C peripheral device

1 (default) | nonnegative integer

Specify the data size that you want to read from the I2C peripheral device for the selected data type.

Output error status — Option to display the error status during data transmission

off (default) | on

When you select this parameter, an output port, labeled as **Status**, becomes available.

The port outputs one of these values:

- 0 for a successful read operation.
- 1 for a failure in opening bus.
- 0 for a failure in read or write operation.

Sample time — Frequency to read data from the I2C peripheral device
0.01 (default)

Specify how often the I2C Controller Read block reads data from the I2C peripheral device. When you specify this parameter as -1, the block inherits its sample time based on the context of the block within the model.

See Also

I2C Controller Write

PX4 Read Position Setpoint

Read the position setpoints published by the PX4 Navigator module in the uORB topic `position_setpoint_triplet`

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The PX4 Read Position Setpoint block outputs the position setpoints published by the PX4 navigator module in the `position_setpoint_triplet` uORB topic. The block outputs the coordinates of the current destination setpoint (Current), the next setpoint after the current destination (Next) and the most recent setpoint that was traversed (Previous). The block outputs the Previous, Current and Next position setpoints as coordinates in either the World Geodetic System of 1984 (WGS 84), local north-east-down (NED), or local east-north-up (ENU) coordinate systems.

You can create an autonomous mission in the plan view of the QGroundControl (QGC) and upload the mission to the PX4 Autopilot over MAVLink. For information on creating missions, see Plan View. The mission consists of waypoints such as TakeOff, Position (also known as Waypoint), Land and so on. The MAVLink module on the PX4 Autopilot decodes the mission data and stores it in the memory. The navigator module in PX4 retrieves the waypoints from the onboard memory and publishes them in the `position_setpoint_triplet` uORB topic. The Previous, Current, and Next setpoints are automatically updated by the navigator module when the vehicle reaches those waypoints.

Note If the onboard companion computer is enabled with PX4 (PX4 parameter `COM_OBS_AVOID` is set to 1), the current waypoint will be updated using the waypoint data received from the onboard computer. The updated waypoint from the onboard computer is communicated to PX4 using the Path Planning Interface over MAVLink. The PX4 MAVLink module receives the waypoint and publishes the waypoint in `vehicle_trajectory_waypoint` uORB topic. When onboard computer workflow is enabled, this block reads the waypoint from `vehicle_trajectory_waypoint` uORB topic and outputs it as current waypoint.

The block also outputs the waypoint mode for the current position setpoint which indicates the corresponding waypoint type that has been set in the mission uploaded from QGroundControl (QGC). This helps in distinguishing the Current Waypoint types (TakeOff, Waypoint, Land) in Simulink.

Feed the coordinates that the block outputs directly to the controller algorithm designed in Simulink as input commands. Use the **Type** output to distinguish between waypoints and take corresponding appropriate action in the controller.

This block also reads the setpoint data for missions created in the QGC, for which the mission type is Flight plan. However, you cannot use this block to read setpoint data for geofencing and rally missions.

Among the flight plan missions, the supported mission commands whose waypoint data can be retrieved by the block are Takeoff, Land and Waypoint. Return to Launch (RTL) is not supported.

Note The PX4 Read Position Setpoint block is supported only when you select **Use default startup script (rcS)** as the startup script option for the PX4 Autopilot during the **Select System Startup Script in PX4** step of the Hardware Setup process. For more information, see “Selecting Startup Script for PX4 Autopilot”.

Note The PX4 Read Position Setpoint block requires you to connect an SD card to the PX4 Autopilot. Ensure that the connected SD card does not contain any custom startup script (rc.txt) in the `etc` folder.

Note The PX4 Read Position Setpoint block cannot determine if the ongoing mission in QGroundControl is paused. You need to find a separate logic to determine if the mission is paused in QGroundControl.

During Normal mode simulation, the block outputs zeroes.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Ports

Output

Previous — Coordinates of last traversed setpoint by vehicle
Vector

This port outputs the coordinates of the last traversed setpoint by the vehicle either in the WGS84 (latitude, longitude, and altitude) coordinate system or as local coordinates (x, y, and z) in the NED and ENU coordinate systems.

Data Types: double

Current — Coordinates of current destination setpoint of vehicle
Vector

This port outputs the coordinates of the current destination setpoint of the vehicle either in the WGS84 (latitude, longitude, and altitude) coordinate system or as local coordinates (x, y, and z) in NED and ENU coordinate systems.

Data Types: double

Next — Coordinates of next setpoint of vehicle
Vector

This port outputs the coordinates of the next setpoint after the current destination of the vehicle either in the WGS84 (latitude, longitude, and altitude) coordinate system or as local coordinates (x, y, and z) in NED and ENU coordinate systems.

Data Types: double

Type — Waypoint mode for the Current setpoint
Scalar

The **Type** output port provides information on the waypoint mode or mission command for the current setpoint that is set in the mission created and uploaded from QGroundControl. The port outputs one of the following types.

Waypoint Type or Mission Command	Type Output
Takeoff	3
Land	4
Waypoint	0
Loiter	2
Idle	5

Data Types: uint8

Position Setpoint — Simulink bus for the position_setpoint_triplet uORB topic
Bus

This port outputs the Simulink bus for the `position_setpoint_triplet` uORB topic. Use this port to avail other information that is published as part of the `position_setpoint_triplet` topic by the navigator module.

Dependencies

To enable this port, select the **Output Simulink bus for position_setpoint_triplet** parameter.

Data Types: Bus

Trajectory Waypoint — Simulink bus for the vehicle_trajectory_waypoint uORB topic
Bus

This port outputs the Simulink bus for the `vehicle_trajectory_waypoint` uORB topic. Use this port to avail other information that is published as part of the `vehicle_trajectory_waypoint` topic such as yaw setpoint, velocity setpoint and so on.

Dependencies

To enable this port, select the **Output Simulink bus for vehicle_trajectory_waypoint** parameter.

Data Types: Bus

Onboard Failsafe — failsafe flag for onboard computer
Boolean

The port returns value 1 when there is no new waypoint received from the onboard computer during the last 0.5 seconds. Use this flag to enable any specific onboard failsafe action that is modelled with the controller.

Dependencies

To enable this port, select the **Output onboard computer failsafe flag** parameter.

Data Types: Boolean

Onboard Status — Onboard computer enabled status
Boolean

This port outputs the status flag to indicate if the onboard computer workflow is enabled or not.

The port returns value 1 when the onboard computer is enabled with PX4 and 0 otherwise.

Dependencies

To enable this port, select the **Output onboard computer enabled status** parameter.

Data Types: Boolean

Parameters

Main

coordinate Format — Coordinate system for setpoints

Local Coordinates (NED) (default) | WGS84 | Local Coordinates (ENU)

Use this port to specify the desired coordinate format for the previous, current and next waypoints. The format can be specified as one of the following:

- **Local Coordinates (NED)**: Use this option to output the previous, current, and next setpoints as local coordinates (x, y, and z) in the NED frame. The transformation method that is used to convert the Geodetic coordinates provided by the PX4 to local coordinates is flat.
- **Local Coordinates (ENU)**: Use this option to output the previous, current, and next setpoints as local coordinates (x, y, and z) in the ENU frame. The transformation method that is used to convert the Geodetic coordinates provided by PX4 to local coordinates is flat.
- **WGS84**: Use this option to output the previous, current, and next setpoints as the original Geodetic coordinates (latitude, longitude, and altitude) returned by PX4.

Sample time — Interval between outputs

0 . 01 (default) | any nonnegative integer that is a multiple of 0,001 | -1

Specify the time interval at which the block reads values from the uORB network. When you specify this parameter as -1, Simulink determines the best sample time for the block based on the context of the block within the model.

Advanced

Output Simulink bus for position_setpoint_triplet — Bus for position_setpoint_triplet uORB topic
off (default) | on

Select this parameter to enable **Position Setpoint** port. This port contains the full Simulink bus for the **position_setpoint_triplet** uORB topic. Use this port to avail other information that is published as part of the **position_setpoint_triplet** topic by the navigator module.

Output Simulink bus for vehicle_trajectory_waypoint — Bus for vehicle trajectory waypoint
off (default) | on

Select this parameter to enable **Trajectory Waypoint** port.

Output onboard computer failsafe flag — Onboard computer failsafe flag status
off (default) | on

Select this parameter to enable **Onboard Failsafe** port. Use this port to model your own failsafe condition.

Output onboard computer enabled status — Onboard computer enable status
off (default) | on

Select this parameter to enable **Onboard Status** port.

Version History

Introduced in R2021b

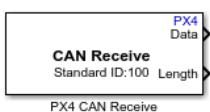
See Also

“Deployment and Verification Using PX4 Host Target and jMAVSim/Simulink”

PX4 CAN Receive

Receive message from CAN network

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots

Description

The PX4 CAN Receive block receives messages from a Controller Area Network (CAN) network by using the PX4 Autopilot CAN port connected to the hardware.

Specify the message type and its properties using the block parameters dialog box.

If you simulate a model that contains the PX4 CAN Receive block without connecting the hardware, the block outputs zeros. For more information, see “Block Produces Zeros or Does Nothing in Simulation” (Simulink).

Also, if there are no messages on the CAN bus, the block outputs zeros.

During Connected I/O simulation, this block reads data from the peripherals of the hardware.

Note To avoid conflict with PX4 UAVCAN, disable UAVCAN before working with PX4 CAN blocks. To disable, set the PX4 parameter, `UAVCAN_ENABLE` to 0. For more information, see [Finding/Updating Parameters](#).

Limitations

1 All PX4 CAN Receive blocks must have the same “**Data is input as** on page 1-0” parameter.

Ports

Output

Data — Message data
vector | bus

The block outputs messages in `uint8` and `CAN Msg` format.

- `uint8` – To output the message as a `uint8` vector array, select **Data is output as** to `Raw data`.
- `CAN Msg` – To output the message in CAN message format, select **Data is output as** to `CAN Msg`. For more information, see “**Data is input as** on page 1-0”.

Data Types: uint8 | CAN Msg

Length — Length of message
scalar

The length of the message, in bytes.

Data Types: uint8 | CAN Msg

Status — Status of received message
scalar

The port outputs the message received status.

The port outputs one of these values:

- 0 for a successful CAN receive.
- 1 for a failure in opening the CAN.
- 2 for a failure in setting Baud rate.
- 3 for a failure in setting CAN Test mode.
- 4 for a failure in CAN receive.

Dependencies

This port appears only when you select the **Output Status** parameter.

Data Types: uint8

Remote — Enable remote message
scalar

The port sends request for remote frames.

Dependencies

This port appears only when you select the **Output Remote** parameter.

Parameters

Data is input as — Data input type
Raw data (default) | CAN Msg

Select a type to receive message.

- Raw data – To receive message as a 1-by- N uint8 array, select **Data is input as** to Raw data.
- CAN Msg – To receive message in CAN message format, select **Data is input as** to CAN Msg and then perform these steps:
 - 1 Add a CAN Unpack block from the Embedded Coder® / Vehicle Network Toolbox™ to your model.
 - 2 Connect the output of the CAN Receive block to the input of the CAN Unpack block.
 - 3 Using the options in the **Data is input as** list of the CAN Unpack block, specify if you want to create your messages or you want to upload a CAN database file. If you choose to upload a CAN database file, the CAN Unpack inherits the message properties from the uploaded file.

Note To use the CAN Unpack block, you must have a Embedded Coder / Vehicle Network Toolbox license.

Message ID — Message identifier

100 (default) | numeric identifier of length 11 or 29 bits

Message identifier, which is 11 bits long for the standard frame size or 29 bits long for the extended frame size, specified in decimal, binary, or hex. For binary and hex formats, use `bin2dec('')` and `hex2dec('')`, respectively, to convert the entry. The message identifier is coded into a message that is sent to the CAN bus.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data. To output the identifier of the CAN Msg, select **Output identifier** of the CAN Unpack block.

Identifier Type — Message identifier type

Standard (11-bit identifier) (default) | Extended (29-bit identifier)

The type of message identifier.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data.

Sample Time — Time interval to read message

0.1 (default) | -1 | nonnegative real value

Specify how often the block receives message, in seconds. When you specify this parameter as -1, Simulink determines the best sample time for the block based on the block context within the model.

Output Status — Enable remote messages

off (default) | on

When you select the **Output Status** parameter, the block configures an output port, **Status**. The port outputs the message remote frame status.

Output Remote Flag — Enable remote messages

off (default) | on

When you select the **Output Remote Flag** parameter, the block configures an output port, **Remote**. The port outputs the message remote frame status.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data.

Version History

Introduced in R2022b

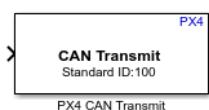
See Also

PX4 CAN Transmit

PX4 CAN Transmit

Transmit message to CAN network

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots

Description

The PX4 CAN Transmit block transmits message to a Controller Area Network (CAN) network by using the PX4 Autopilot CAN port.

Specify the message type and its properties using the block parameters dialog box.

If you simulate a model that contains the PX4 CAN Transmit block without connecting the hardware, the block does nothing. For more information, see “Block Produces Zeros or Does Nothing in Simulation” (Simulink).

During Connected I/O simulation, this block writes data to the peripherals of the hardware.

Note To avoid conflict with PX4 UAVCAN, disable UAVCAN before working with PX4 CAN blocks. To disable, set the PX4 parameter, `UAVCAN_ENABLE` to 0. For more information, see [Finding/Updating Parameters](#).

Ports

Input

Port1 — Message data
vector | scalar

The block accepts messages in `Raw data` and `CAN Msg` formats.

- `Raw data` – To accept the message as a `uint8` vector array, set **Data is input as** to `Raw data`.
- `CAN Msg` – To accept the message in CAN message format, set **Data is input as** to `CAN Msg`. You can create a message or upload a CAN database file. For more information, see “Data is input as” on page 1-0 .

Dependencies

This port appears only when you select the **Remote Frame** parameter.

Data Types: `uint8` | `CAN Msg`

Output

Port1 — Status of data transmission
scalar

Output port to display status of data transmission.

The port outputs one of these values:

- 0 for a successful CAN transmission.
- 1 for a failure in opening the CAN.
- 2 for a failure in setting Baud rate.
- 3 for a failure in setting CAN Test mode.
- 4 for a failure in CAN transmission.

For more information, see “Output Status” on page 1-0 .

Dependencies

This port appears only when you select the **Output Status** parameter.

Parameters

Data is input as — Data input type
Raw data (default) | CAN Msg

Select the type of the message at the input port.

- Raw data - To accept the message as a uint8 vector array, set **Data is input as** to Raw data.
- CAN Msg - To accept the message in CAN message format, set **Data is input as** to CAN Msg and then perform these steps:
 - 1 Add a CAN Pack block from the Embedded Coder / Vehicle Network Toolbox to your model.
 - 2 Connect the output of CAN Pack block to the input port of the CAN Transmit block.
 - 3 Using the options in the **Data is input as** list of the CAN Pack block, specify if you want to create your messages or you want to upload a CAN database file. If you choose to upload a CAN database file, the CAN Pack inherits the message properties from the uploaded file.

Note To use CAN Pack block, you must have an Embedded Coder / Vehicle Network Toolbox license.

Identifier Type — Message identifier type

Standard (11-bit identifier) (default) | Extended (29-bit identifier)

The type of message identifier.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data. For CAN Msg, **Identifier Type** is inherited from the type specified in the CAN Pack block.

Message ID — Message identifier

100 (default) | numeric identifier of length 11 or 29 bits

Message identifier, which is 11 bits long for the standard frame or 29 bits long for the extended frame, specified in decimal, binary, or hex. For binary and hex formats, use `bin2dec('')` and `hex2dec('')`, respectively, to convert the entry. The message identifier is coded into a message that is sent to the CAN bus.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data. For CAN Msg, **Message ID** is inherited from the identifier specified in the CAN Pack block.

Data Field Length — Length of message

3 (default) | positive integer

The length of the message, in bytes.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data. For CAN Msg, **Message Length** is inherited from the length of the CAN Pack block.

Output Status — Status of data transmission

off (default) | on

When you select the **Output Status** parameter, the block configures an output port. The port outputs the status of the data transmission as a `uint8` integer. Each bit in the integer corresponds to the type of error occurred during data transmission. An output of 0 indicates successful transmission.

Remote Frame — Enable remote message

off (default) | on

When you select the **Remote Frame** parameter, the remote frame of the message is set to 1 indicating that the block is requesting the transmission of a specific identifier.

Dependencies

This parameter appears only when you select **Data is input as** to Raw data. For CAN Msg, the **Remote Frame** is inherited from the request specified in the CAN Pack block.

Version History

Introduced in R2022b

See Also

PX4 CAN Receive

MAVLink Bridge Source

Read MAVLink data from the Pixhawk board

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:
px4MAVLinkBridgelib

Description

The MAVLink Bridge Source block reads MAVLink data stream from the Pixhawk board connected to the serial port and routes it to the Simulink plant model and the configured UDP/TCP objects. This block allows you to configure Serial, UDP, and TCP connections.

To access the block, open the block library by entering the following command in the MATLAB® Command window.

`px4MAVLinkBridgelib`

The MAVLink Bridge Source and MAVLink Bridge Sink blocks provides a way to route MAVLink data stream from Pixhawk hardware to the mission planning software QGroundControl(QGC), On-board computer hardware, Simulink Plant, and vice versa in HITL workflow. For more information, see “MAVLink Connectivity for QGC, On-board Computer and Simulink Plant”.

Ports

Output

Data — MAVLink data stream
vector

This port outputs the MAVLink data stream.

Data Types: `uint8`

Length — Length of the data stream
scalar

This port outputs the length of the data stream for the current time setup.

Data Types: `uint16`

Parameters

Serial Port — Port for connecting the Pixhawk board
Select a port (default) | COM1 | COM3 | Specify manually

Select a serial port to connect to the Pixhawk board. If the required ports are not available in the list, select **Specify manually** and enter the serial port in the **Serial Port (specify manually)** field.

Max Data Length — Maximum output data stream length
1024 (default) | any non-negative value

Enter the maximum output data stream length.

Destination Ports — Port for enabling data routing
'UDP', '127.0.0.1', '14550' (default)

Click **Add** to add the destination UDP/TCP connections for data routing. The existing connections ('UDP', '127.0.0.1', '14550') are used for connecting to QGC.

Note The MAVLink bridge block utilizes the following local port settings:

- Local Port for Ground Control Stations: 14570
 - Local Port for Onboard Connection: 14580
-

Sample time — Interval between outputs
0.01 (default) | any nonnegative integer that is a multiple of 0,001 | -1

Specify the time interval at which the block reads values from the serial connections and routes it to the added UDP/TCP connections.

Version History

Introduced in R2022a

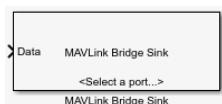
See Also

MAVLink Bridge Sink

MAVLink Bridge Sink

Write MAVLink data to the Pixhawk board

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:
px4MAVLinkBridgelib

Description

The MAVLink Bridge Sink block reads the MAVLink data from the source UDP/TCP ports, merges it with input data and writes to Pixhawk hardware connected to Serial port.

To access the block, open the block library by entering the following command in the MATLAB Command window.

```
px4MAVLinkBridgelib
```

The MAVLink Bridge Sink and MAVLink Bridge Source blocks provides a way to route MAVLink data stream from Pixhawk hardware to the mission planning software QGroundControl(QGC), On-board computer hardware, Simulink Plant, and vice versa in HITL workflow. For more information, see “MAVLink Connectivity for QGC, On-board Computer and Simulink Plant”.

Ports

Input

Data — MAVLink data stream
vector

This port takes MAVLink data stream (uint8) as input.

Data Types: uint8

Parameters

Serial Port — Port for connecting the Pixhawk board
Select a port (default) | COM1 | COM3 | Specify manually

Select a serial port for connecting to the Pixhawk board. If the required ports are not available in the list, select **Specify manually** and enter the serial port in the **Serial Port (specify manually)** field.

Note You must select / specify the same serial port in MAVLink Bridge Source and MAVLink Bridge Sink blocks.

Source Ports — Port for enabling data routing

'UDP', '127.0.0.1', '14550' (default)

Click **Add** to add the source UDP/TCP connections for data routing. The existing connections ('UDP', '127.0.0.1', '14550') are used for connecting to QGC.

Note The MAVLink bridge block utilizes the following local port settings:

- Local Port for Ground Control Stations: 14570
 - Local Port for Onboard Connection: 14580
-

Version History

Introduced in R2022a

See Also

MAVLink Bridge Source

PX4 ULog

Log Simulink signals to an SD card in ULog format

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The PX4 ULog block logs the Simulink signals to an SD Card in the ULog format.

In PX4 Autopilots, the system architecture uses logger module that logs topics in ULog file formats. PX4 uses ULog file format for uORB logging messages. These messages are commonly known as micro-ORB (Object Request Broker) and form the basis for subscribe, publish architecture in PX4.

The ULog format is efficient for post-flight analysis. Using ulogreader, flight logger app or online tools you can analyze, visualize, and convert ULog files to gain insights and improve the performance of PX4-based autonomous systems.

The number of ports in the block updates as per the uORB topic selected. To create your own topic, use `createPX4uORBMessage` API. The block is currently configured with default uORB topic `SL_CustomPX4uORB` that is build during setup screens.

Note This block does not support PX4 Host target and Connected I/O Simulation.

Ports

Input

double_a — First input signal, specified as double scalar

Input signal to be logged by the block, specified as double.

Data Types: double

double_b — Second input signal, specified as double scalar

Input signal to be logged by the block, specified as double.

Data Types: double

single_a — First input signal, specified as single scalar

Input signal to be logged by the block, specified as single.

Data Types: single

single_b — Second input signal, specified as single scalar

Input signal to be logged by the block, specified as single.

Data Types: single

Parameters

Topic — Name of the uORB topic to which the Simulink signals are to be logged
SL_CustomPX4uORB (default) | one of the uORB topics

Select the name of the uORB topic that the block needs to log Simulink signals. Click **Select** to browse topic names available in the **Firmware/msg** folder of the px4 directory that you downloaded.

Version History

Introduced in R2023b

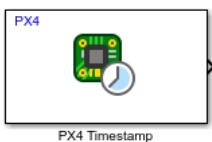
See Also

PX4 Timestamp

PX4 Timestamp

Output absolute timestamp from the PX4 Autopilot to a Simulink model

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The PX4 timestamp block gets the time since start in microseconds from the PX4 Autopilot into the Simulink Model. This timestamp information is absolute, ensuring it never wraps or goes backwards. The output datatype of timestamp is `uint64`.

Ports

Output

Timestamp — Absolute timestamp
`uint64`

Absolute timestamp from the PX4 Autopilot, returned as `uint64`.

Data Types: `uint64`

Parameters

Sample time — Interval at which the block reads values
`0.01` (default) | any non-negative value that is a multiple of `0.01` | `-1`

Enter the time interval at which the block reads values from the Pixhawk Autopilot.

When you set this parameter to `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Version History

Introduced in R2023b

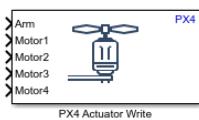
See Also

PX4 ULog

PX4 Actuator Write

Set actuator values for motors and servos

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Peripheral Blocks

Description

The PX4 Actuator Write block writes motor and servo actuator values over the PWM and UAVCAN interfaces. The PX4 Actuator Write block accepts single scalar values between -1 to 1 as input and writes those values to the selected motors or servos using `ActuatorTest` uORB topic.

An input value of 1 indicates maximum output, a value of 0 centers the servos or sets the motors to their lowest possible thrust, and an input of -1 indicates the maximum output but in the reverse direction. However, for motors that do not support reverse operation, any value less than zero stops the motor.

Setting the **Arm** input to `true` value arms the actuators, while `false` serves as a kill switch and disarms the actuators.

Configure Actuators

For firmware version 1.14, you must configure the PWM & AUX pins in QGroundControl before using the PX4 Actuator Write block in Simulink. To configure the pins, you must use QGroundControl version 4.3.0, which can be downloaded from this location. For information on configuring the actuators, see “Configure and Assign Actuators in QGC”.

Ports

Input

Arm — Arm PX4 flight controller

`0 | 1`

Enable signal for arming the flight actuators.

A value of 0 indicates that the flight actuator is not armed, whereas a value of 1 indicates that the flight actuator is armed.

Data Types: Boolean

Motor x — Inputs that accept motor values

`[-1, 1]`

Connect the Motor x inputs to signals that represent single scalar values.

If you provide an input greater than 1 or less than -1, the block internally clips it to 1 and -1, respectively.

Dependencies

To enable these ports, select the required number of motors (1-12) from the **Motors** parameter.

Data Types: `single`

Servo x — Input that accept servo values

`[-1, 1]`

Connect the Servo x inputs to signals that represent single scalar values.

If you provide an input greater than 1 or less than -1, the block internally clips it to 1 and -1, respectively.

Dependencies

To enable these ports, select the required number of servos (1-8) from the **Servos** parameter.

Data Types: `single`

Motor Values — Input that accept motor values as an array

`-1 | 1`

Connect the Motor Values inputs to signals that represent values as an array.

An input value of 1 indicates maximum output, a value of 0 sets the motors to their lowest possible thrust, and an input of -1 stops the motor.

Dependencies

To enable this port, select **Input Motor Values as an array** parameter.

Data Types: `single`

Servo Values — Input that accept servo values as an array

`-1 | 1`

Connect the Servo Values inputs to signals that represent values as an array.

An input value of 1 indicates maximum output, a value of 0 centers the servos, and an input of -1 indicates the maximum output in the reverse direction.

Dependencies

To enable this port, select **Input Servo Values as an array** parameter.

Data Types: `single`

Parameters

Motors tab

Motors — Motors to write values to

Motor numbers

Select the motor numbers to which you want to send the single scalar values. The number of **Motor** input ports on the block depends on the number of motors you select in this parameter.

Input Motor Values as an array — Option to specify motor values as an array
off (default) | on

Select this parameter to enable the **Motor Values** input port.

You can input the motor values as an array of size 1-by-12.

Servos tab

Servos — Servos to write values to
Servo numbers

Select the servo numbers to which you want to send the single scalar values. The number of **Servo** input ports on the block depends on the number of servos you select in this parameter.

Input Servo Values as an array — Option to specify servo values as an array
off (default) | on

Select this parameter to enable the **Servo Values** input port.

You can input the servo values as an array of size 1-by-8.

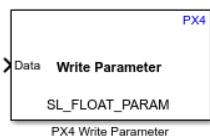
Version History

Introduced in R2024b

PX4 Write Parameter

Write PX4 system parameters

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The Write Parameter block writes to the PX4 system parameter you specify, allowing easy parameter modification.

The block supports the parameters listed in PX4 Parameter Reference. You can also use your own custom parameters that align with the PX4 guidelines. For more information, see PX4 guidelines.

During Connected IO simulation, this block writes data to the peripherals of the hardware.

Ports

Input

Data — Parameter data vector

Specify the data to write to the PX4 parameter.

Data Types: int32 | float

Parameters

Parameter Name — Name of the PX4 parameter
SL_FLOAT_PARAM (default) | one of the parameter name

Select the required PX4 system parameter from the drop-down list. The **Parameter Name** selected in this field must be one of the PX4 parameters; otherwise, the **Status** port outputs a value of 1, indicating that the parameter handle is invalid. Additionally, parameters requiring a system reboot are not supported.

Data Type — Data type of the PX4 parameter
FLOAT (default)

The data type is autopopulated based on the parameter you selected in the **Parameter Name** field.

Note If the **Data type** that you select in this field does not match the actual data type of the PX4 parameter, the values written might lead to data type mismatch.

Output Status — Status of data transmission

off (default) | on

When you select the **Output Status** parameter, the block configures an output port. The output status indicates if the parameter that you specified is valid or not. A value of 0 indicates that the parameter handle is valid, and a value of 1 indicates that the parameter handle is invalid.

Version History

Introduced in R2025a

PX4 Write Thrust & Torque Setpoint

Publish the thrust and torque setpoints to uORB topics

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.



Libraries:

UAV Toolbox Support Package for PX4 Autopilots / PX4 Utility Blocks

Description

The block writes motor and servo actuator values by directly publishing thrust and torque setpoints to the `vehicle_thrust_setpoint` and `vehicle_torque_setpoint` uORB topics. This block accepts two inputs, thrust and torque, each as a 1-by-3 vector. The thrust setpoint is along, and the torque setpoint is about, the x-, y-, and z- body axes. Both inputs are normalized to the range [-1, 1] in the vehicle body FRD frame.

Ports

Input

Thrust — Thrust setpoint
vector

Specify the desired thrust setpoint along x-, y-, and z-axes of the vehicle body. The specified inputs are normalized to the range [-1, 1] in the vehicle body FRD frame.

Data Types: `single`

Torque — Torque setpoint
vector

Specify the desired torque setpoint about the x-, y-, and z-axes of the vehicle body. The specified inputs are normalized to the range [-1, 1] in the vehicle body FRD frame.

Data Types: `single`

Version History

Introduced in R2025a

Getting Started with uORB Blocks for PX4 Autopilots Support Package

This example shows you how to use the PX4 uORB Read and the PX4 uORB Write blocks in UAV Toolbox Support Package for PX4® Autopilots to read and write uORB messages in the uORB network.

Introduction

UAV Toolbox Support Package for PX4 Autopilots enables you to create Simulink® models that work with the uORB middleware. The uORB is an asynchronous publish() / subscribe() messaging API used for inter-thread/inter-process communication. A component sends a message by publishing it to a particular topic, such as `sensor_accel`. Other components receive the message by subscribing to that topic.

UAV Toolbox Support Package for PX4 Autopilots includes a library of Simulink blocks for sending and receiving messages for a specified uORB topic. When you generate code for a Simulink model containing PX4 uORB Read and PX4 uORB Write blocks and deploy it to a Pixhawk® series flight controller, the Simulink app in the controller reads data from the uORB network and also writes data to the network for the corresponding uORB topic.

In this example, you will learn how to:

- Create Simulink models to send and receive uORB messages
- Observe the real time uORB data by running the model in External mode

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

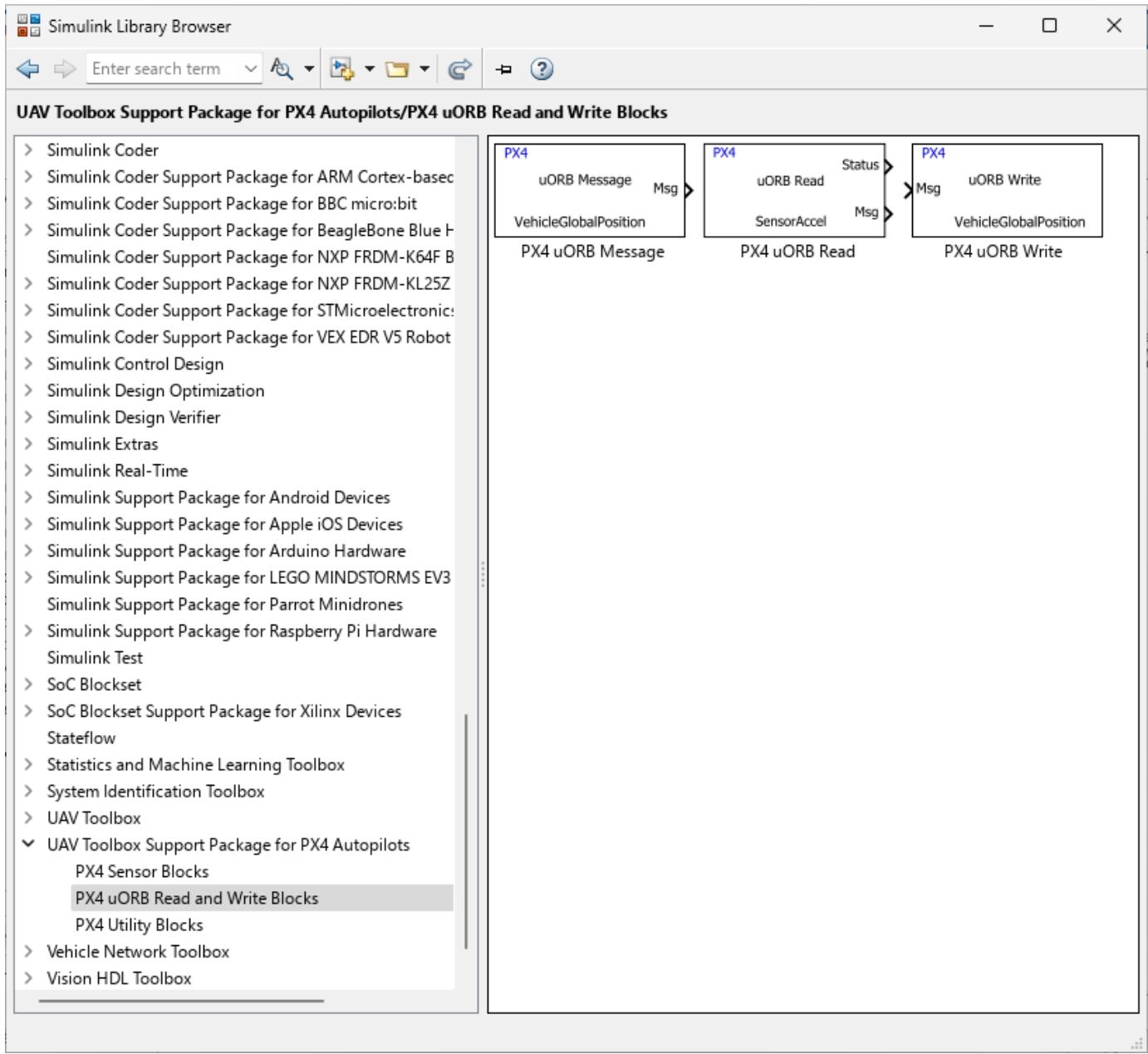
To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)

Task 1 - Configure and Run a Model to Read Sensor Data Using uORB Topics

In this task, you will configure a Simulink model to read sensor data from PX4 flight controller using a PX4 uORB Read block and observe the data over External mode.

1. From the MATLAB® toolbar, select **HOME > New > Simulink Model** to open the Simulink Start Page. Click **Blank Model** to launch a new Simulink model.
2. On the Simulink toolbar, from the **Simulation** tab select **Library Browser** to open the Simulink Library Browser. Click the UAV Toolbox Support Package for PX4 Autopilots tab (you can also type `px4lib` in MATLAB command window). Select **PX4 uORB Read and Write Blocks** category.



3. Drag a PX4 uORB Read block to the model.

4. Double-click the block to open the Block Parameters dialog box. Click **Select** next to the "Topic to subscribe to" field. A pop-up window appears containing the *msg* folder of the downloaded PX4 firmware. Select *SensorAccel.msg*, and click **Open**. Click **OK** to close the dialog box.

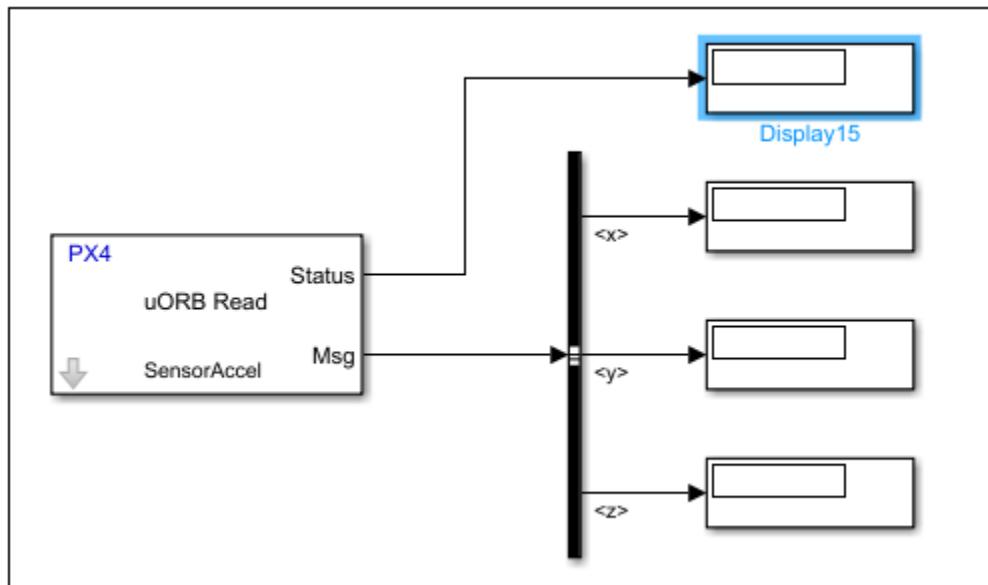
5. From the Simulink > Signal Routing tab in the Library Browser, drag a Bus Selector block to the model.

6. Connect the **Msg** output of the PX4 uORB Read block to the input port of the Bus Selector block.

7. Double-click the Bus Selector block. Add the *x*, *y*, and *z* signals to the block output. Remove *signal1* and *signal2* from the block output.

Note: If the x, y, and z signals are not listed as elements of the input bus, close the dialog box. To ensure that the bus information is propagated, on the **Modeling** tab, click **Update Model**. Then, reopen the dialog box.

8. From the Simulink > Sinks tab in the Library Browser, drag four Display blocks to the model. Connect the three outputs of the Bus Selector block to the three Display blocks. Connect the Status output to the fourth Display block. Your entire model should look like this.

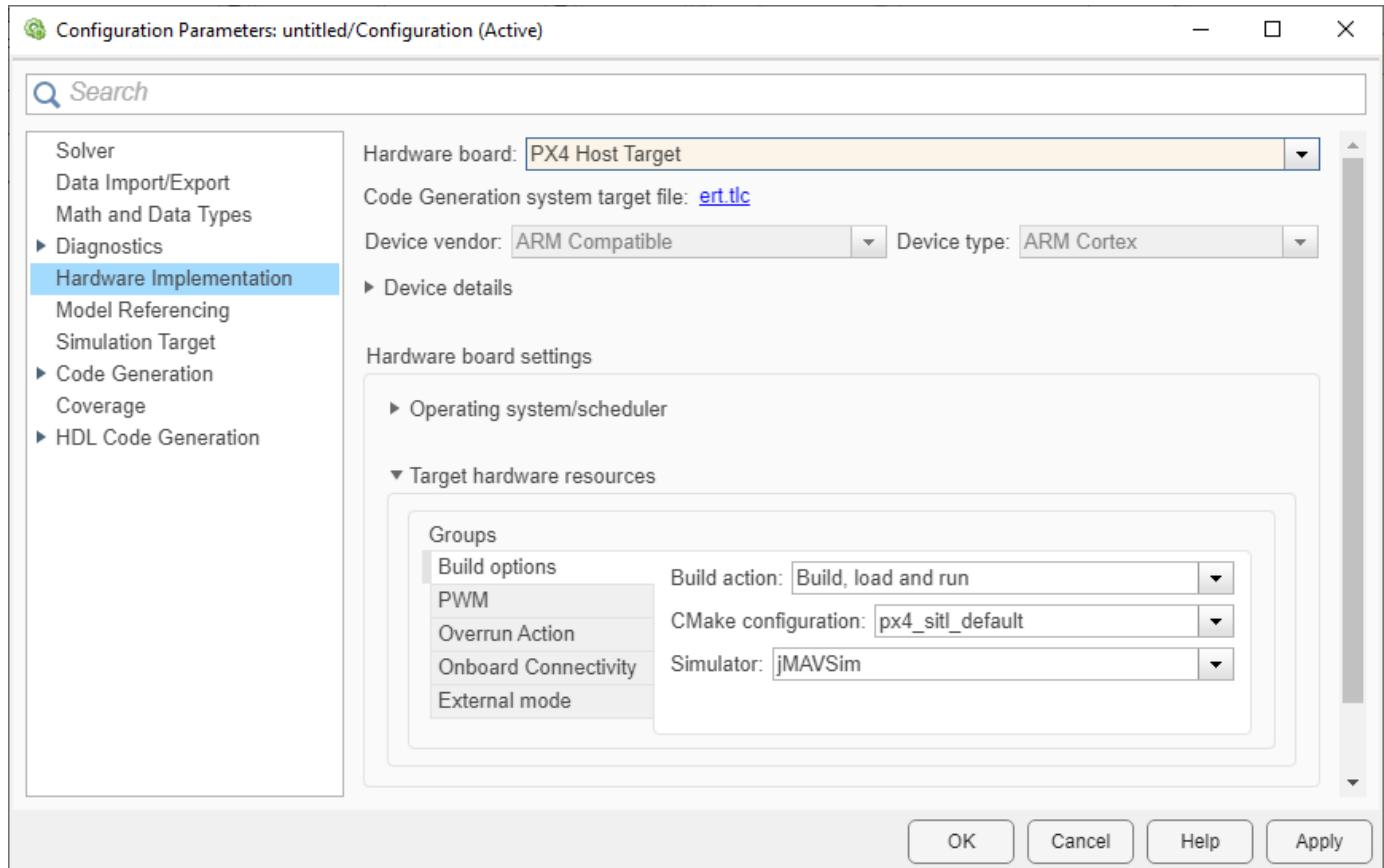


9. Connect the Pixhawk series controller to the host computer using the USB cable.

10. In the **Modeling** tab, click **Model Settings**.

11. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane:

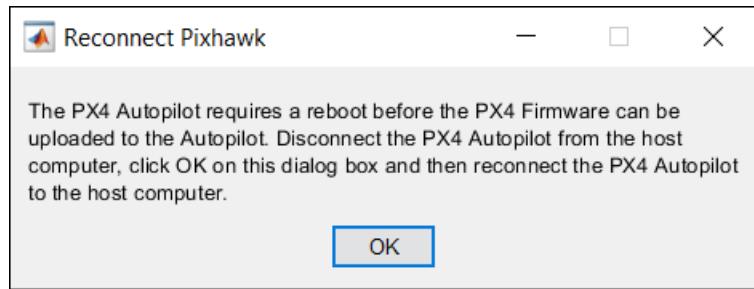
- Set the **Hardware board** to the same Pixhawk series controller that you selected during Hardware Setup screens.
- In the **Target Hardware Resources** section, enter the serial port of the host computer to which the Pixhawk series controller is connected, in the *Serial port for firmware upload* field. The *Serial port for Nuttx (NSH) Terminal* is optional.



12. In the **Simulation** tab, set the simulation **Stop Time** to inf.

13. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

Wait for the code generation to be completed. Whenever the dialog box appears instructing you to reconnect the flight controller to the serial port, ensure that you click **OK** on the dialog box within **5 seconds** after reconnecting the flight controller.



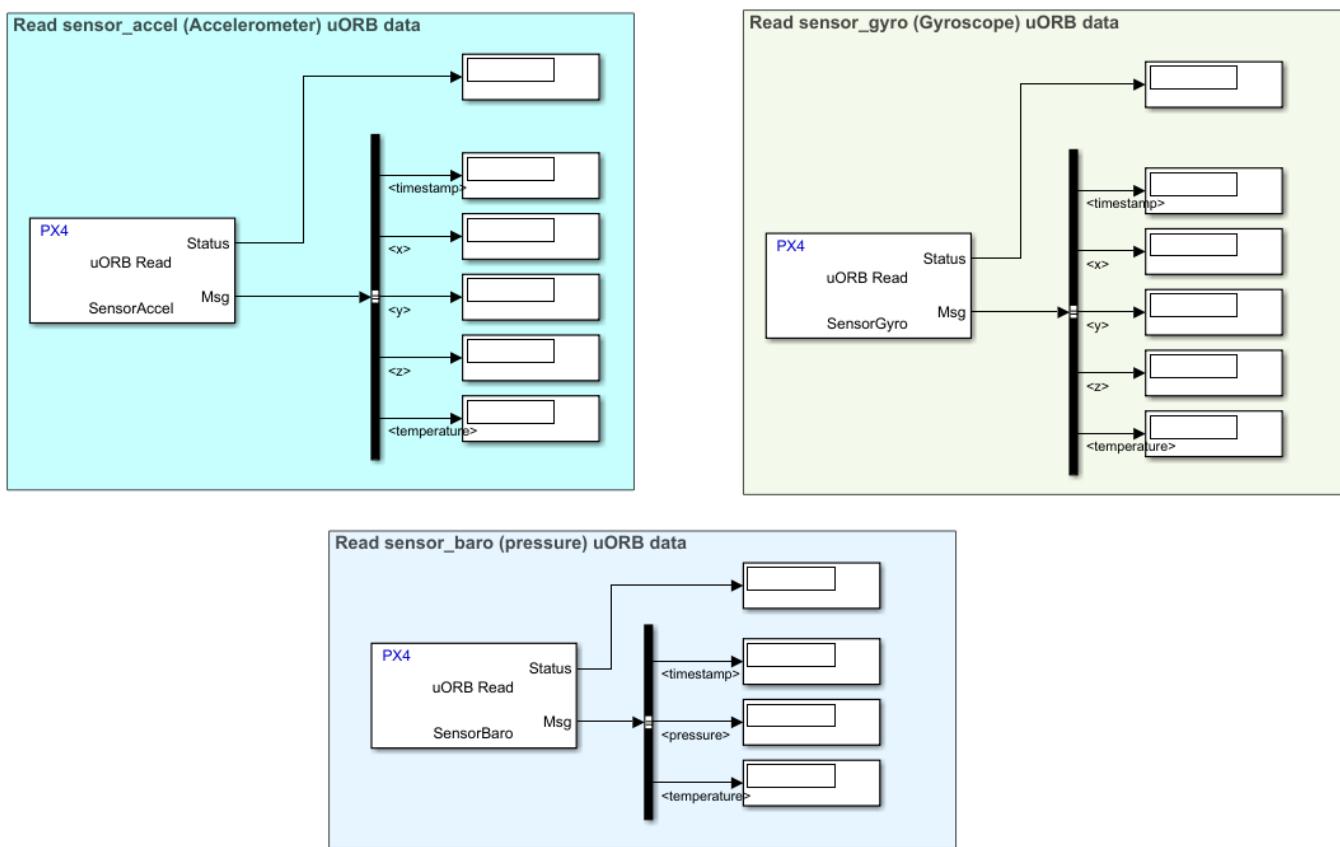
After the Pixhawk series controller is flashed, wait for the signal monitoring to start.

Observe the accelerometer values while the simulation is running. Change the position of the board along all the axes and observe the corresponding accelerometer values along those axes.

A pre-configured Simulink model (*px4demo_uORBRead*) is also available, which helps you to read the accelerometer, gyroscope, and barometer values using the `sensor_accel`, `sensor_gyro` and `sensor_baro` uORB topics respectively. This model is configured with PX4 Host Target as the default board. When PX4 Host Target is set as the hardware board, you can perform Connected I/O simulation. When you simulate a model in Connected I/O, the model communicates with the PX4 Host Target, enabling you to read and write data to the PX4 Host Target from Simulink.

Open the *px4demo_uORBRead* model.

PX4 uORB Read



Task 2 - Create a Publisher

In this task and the two subsequent tasks (Task 3 and Task 4), you will create a complete model that consists of blocks to publish data (PX4 uORB Write) and subscribe data (PX4 uORB Read), using the `VehicleGlobalPosition` uORB topic.

1. From the MATLAB toolbar, select HOME > New > Simulink Model to open a new Simulink model.
2. On the Simulink toolbar, from the **Simulation** tab select **Library Browser** to open the Simulink Library Browser. Click the UAV Toolbox Support Package for PX4 Autopilots tab (you can also type **px4lib** in MATLAB command window). Select **PX4 uORB Read and Write Blocks** category.
3. Drag a PX4 uORB Write block to the model.

4. Double-click the block. Click **Select** next to the "Topic to publish to" field. A pop-up window appears containing the *msg* folder of the selected PX4 firmware. Select **vehicle_global_position.msg**, and click **Open**.

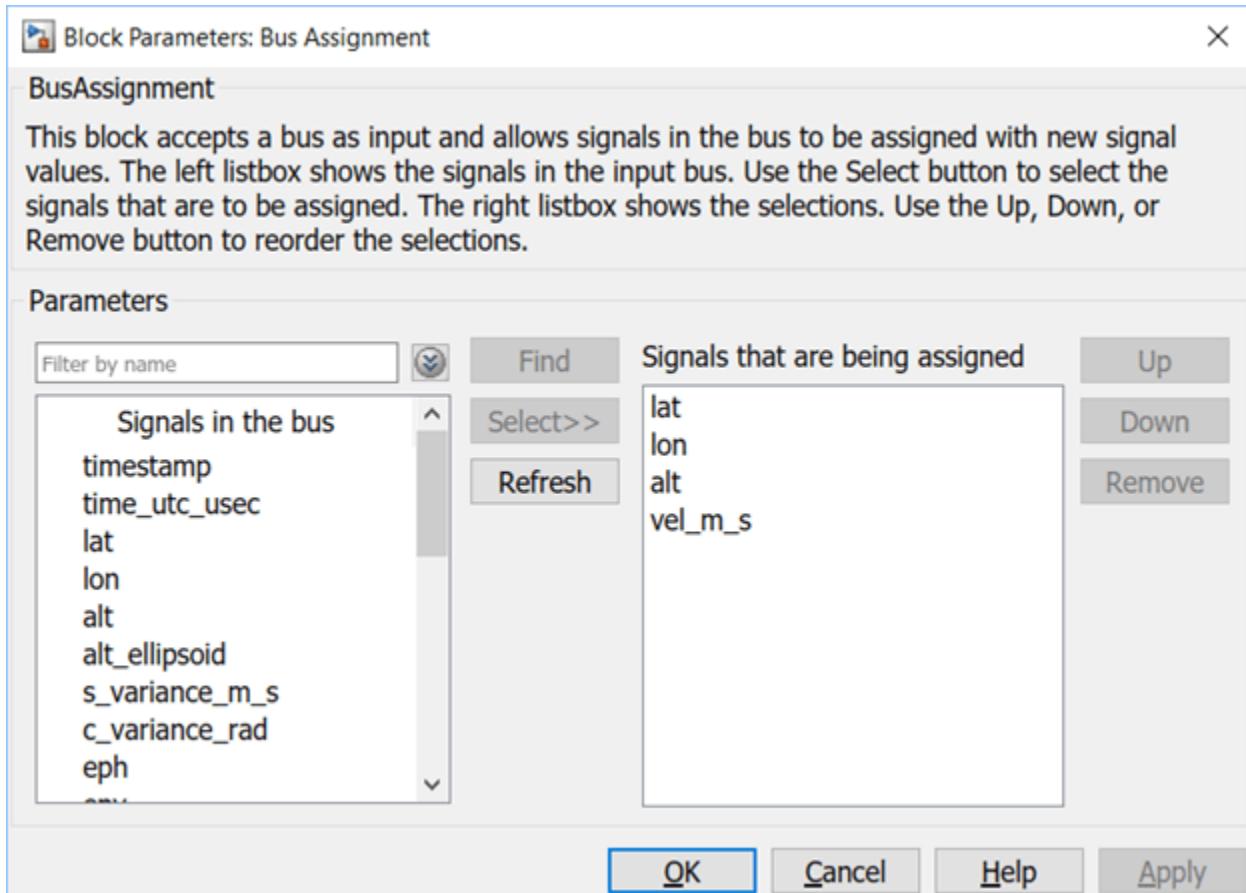
Click **OK** to close the dialog box.

Task 3 - Create a Blank uORB Message

In this task, you will create a blank uORB message in the model created in Task 2, and populate the latitude (lat), longitude (lon), altitude (alt), and velocity (terrain_alt) fields for uORB topic **VehicleGlobalPosition**.

A uORB message is represented as a bus in Simulink. A bus is a bundle of Simulink signals, and can also include the other buses (see “Simulink Bus Capabilities” (Simulink) for an overview). The **PX4 uORB Message** block outputs a Simulink bus corresponding to a uORB message.

1. On the Simulink toolbar, from the **Simulation** tab select **Library Browser** to open the Simulink Library Browser. Click the UAV Toolbox Support Package for PX4 Autopilots tab (you can also enter the command **px4lib** in MATLAB command window). Select the *PX4 uORB Read and Write Blocks* category.
2. Drag a PX4 uORB Message block to the model. Double-click the block to open the Block Parameters dialog box.
3. Click **Select** next to the **Topic** field. A pop-up window appears containing the *msg* folder of the selected PX4 firmware. Select **vehicle_global_position.msg**, and click **Open**. Click **OK** to close the dialog box.
4. From the Simulink > Signal Routing tab in the Library Browser, drag and drop a Bus Assignment block.
5. Connect the output port of the **PX4 uORB Message** block to the **Bus** input port of the Bus Assignment block. Connect the output port of the Bus Assignment block to the input port of **PX4 uORB Write** block.
6. Double-click the Bus Assignment block. Select **???signal1** in the list on the right, and click **Remove**. Select the **lat**, **lon**, **alt**, and **terrain_alt** signals in the list on the left, and click **Select** to move them to the right. Click **OK** to close the dialog box.

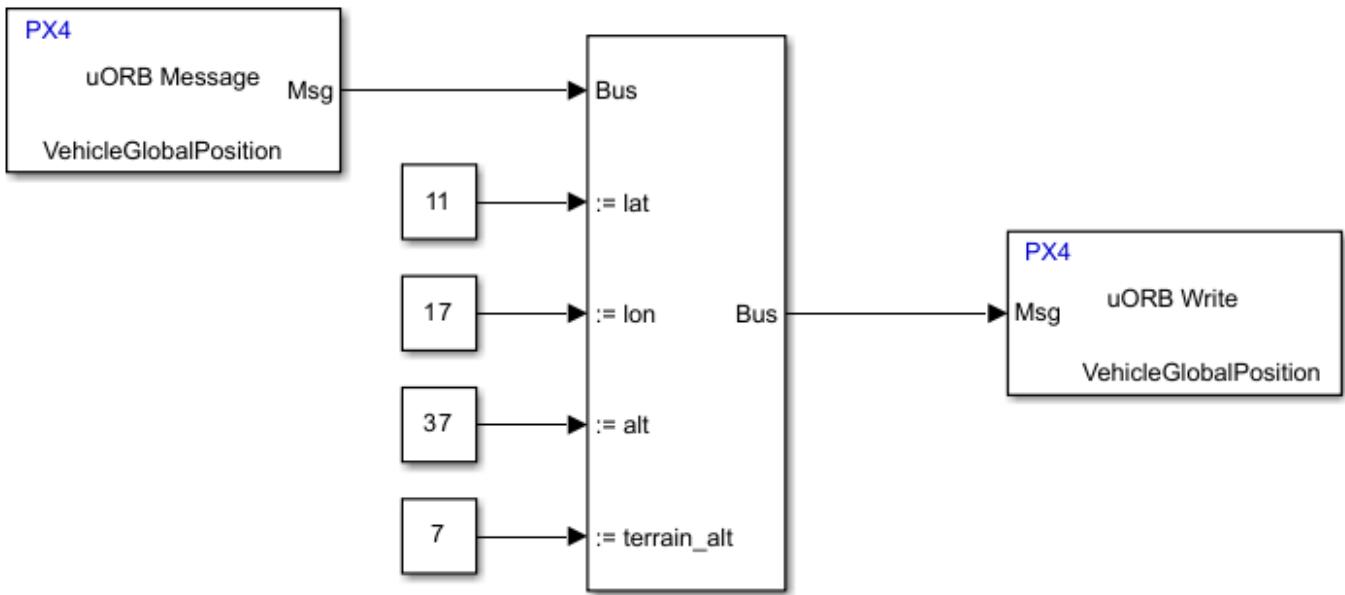


Note: If the `lat`, `lon`, `alt`, and `terrain_alt` signals are not listed as elements of the input bus, close the dialog box. To ensure that the bus information is propagated, on the **Modeling** tab, click **Update Model**. Then, reopen the dialog box.

You can now populate the bus with random values that represent the GPS location.

7. From the Simulink > Sources tab in the Library Browser, drag four Constant blocks into the model.
8. Connect the output ports of each Constant block to the Bus Assignment input ports.
9. Double-click each of the Constant blocks that is connected to the Bus Assignment block, set the **Constant** values (as indicated in the below figure), and set the **Output data type** as `Inherit:Inherit via back propagation`.

The model should now look like this:



Task 4 - Create a Subscriber

In this task, you will add a subscriber (PX4 uORB Read block) to the model created in Task 3, to receive messages sent to the `VehicleGlobalPosition` topic. You will extract the `lat`, `lon`, `alt` and `terrain_alt` signals from the message.

1. From the UAV Toolbox Support Package for PX4 Autopilots tab in the Library Browser, drag a PX4 uORB Read block to the model.
2. Double-click the block. Click **Select** next to the `Topic to subscribe to` field. A pop-up window appears containing the `msg` folder of the selected PX4 firmware. Select `vehicle_global_position.msg`, and click **Open**. Click **OK** to close the dialog box.

The PX4 uORB Read block outputs a Simulink bus. To extract the `lat`, `lon`, `alt`, and `terrain_alt` signals from the bus, use a Bus Selector block.

3. From the Simulink > Signal Routing tab in the Library Browser, drag and drop a Bus Selector block to the model.
4. Connect the **Msg** output of the PX4 uORB Read block to the input port of the Bus Selector block.
5. Double-click the Bus Selector block. Add `lat`, `lon`, `alt`, and `terrain_alt` signals to the block output. Then, remove `signal1` and `signal2` from the block output.

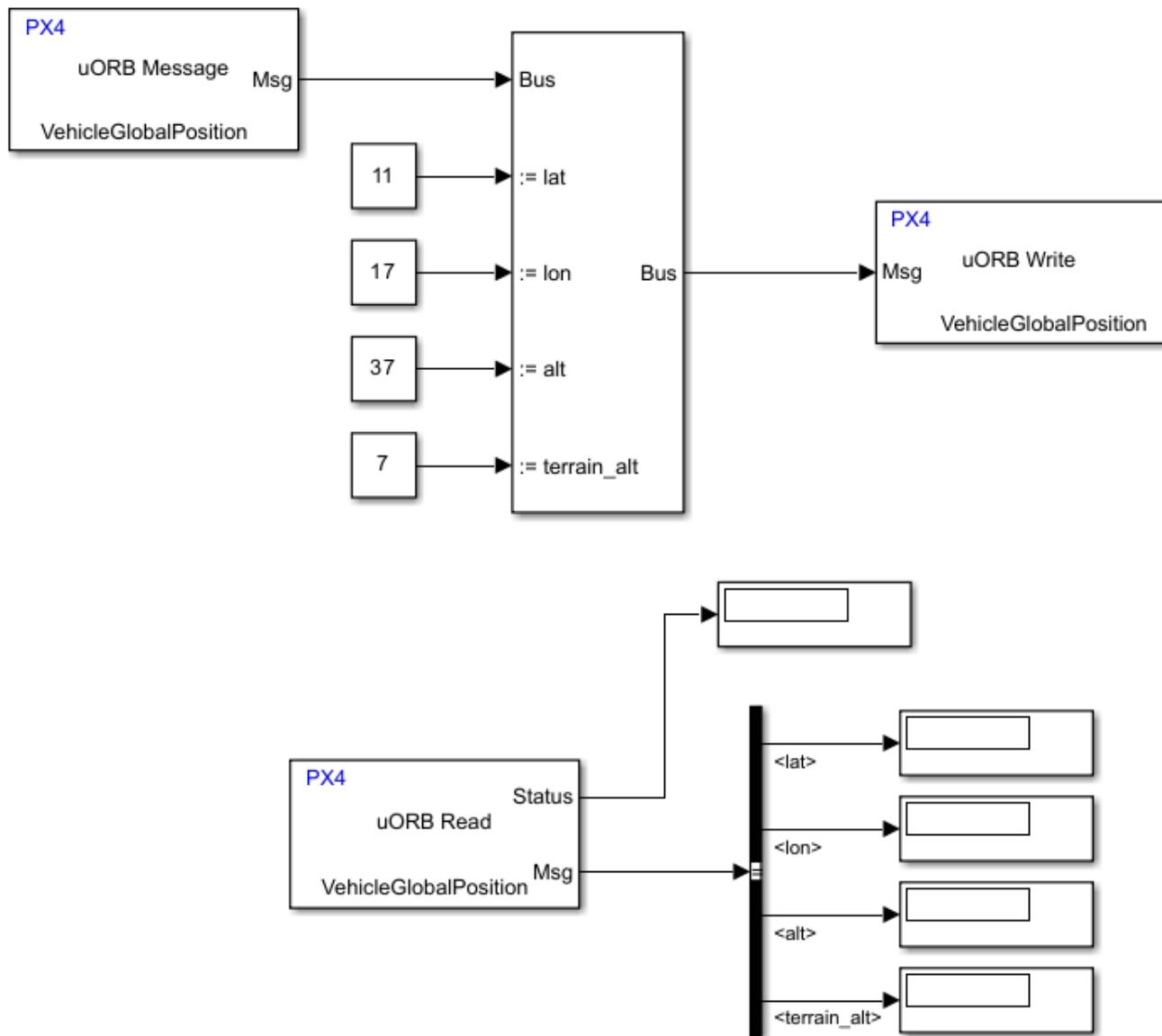
The PX4 uORB Read block outputs the most-recently received message for the topic on every time step. The Status output indicates whether the message has been received during the prior time step. For the current task, the Status output is not needed, and you can terminate it.

6. Click Simulink > Sinks tab in the Library Browser. Drag a Terminator block to the model.
7. Connect the Status output of the PX4 uORB Read block to the input of the Terminator block.

The next step will help you to configure the display of the extracted `lat`, `lon`, `alt` and `terrain_alt` signals.

- 8.** From the Simulink > Sinks tab in the Library Browser, drag four Display blocks to the model. Connect the each output of the Bus Selector block to each Display block.

Your entire model should look like this:



- 9.** Save the model.

Task 5 - Configure and Run the Model

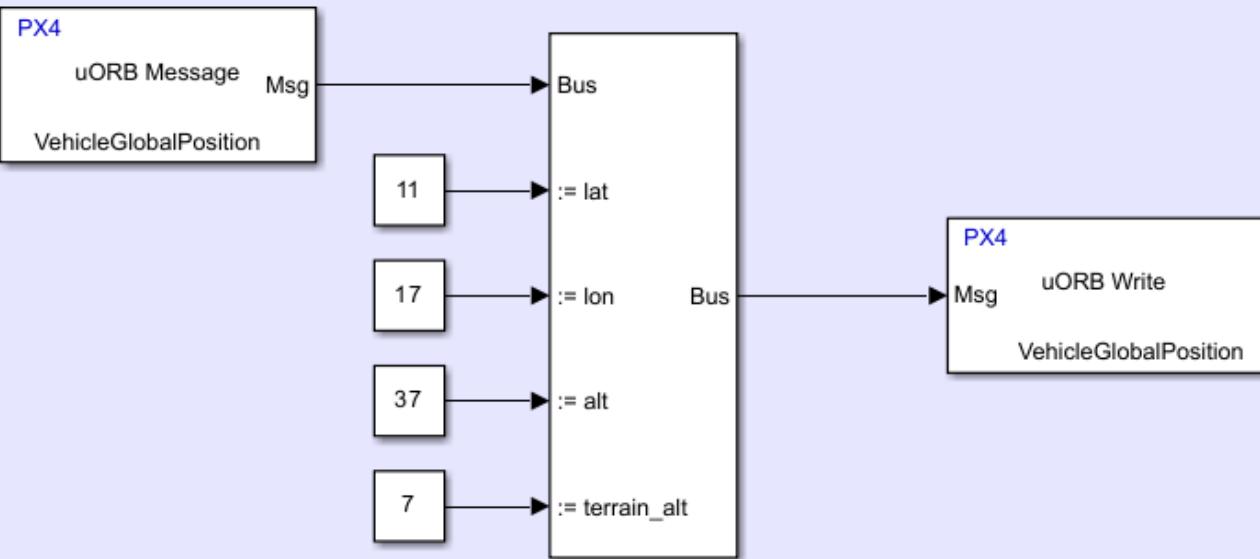
In this task, you will define the configuration parameters and run the complete model (created through Tasks 2 to 4) for signal monitoring and parameter tuning.

1. Open Configuration Parameters dialog box, and set the Target Hardware as the Pixhawk Series controller that you have selected during Hardware Setup screens.
2. Enter the serial port to which the Pixhawk Series controller is connected to host computer. Click **Apply** and **OK**.
3. In the **Simulation** tab, set the simulation **Stop Time** to **inf**.
4. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.
6. While the simulation is running, change the value of the constant block connected to the **alt** field of the Bus Assignment block, from 37 to 44. You can observe the corresponding change in Display block connected to **alt** port of the Bus Selector block.
7. Observe that the base MATLAB workspace has one or more variables whose name starts with **px4_Bus_**. These are temporary bus objects created by Simulink and should not be modified. However, it is safe to clear them from the workspace, as they will be re-created if needed.
8. In the **Hardware** tab, click **Stop** to stop the simulation.

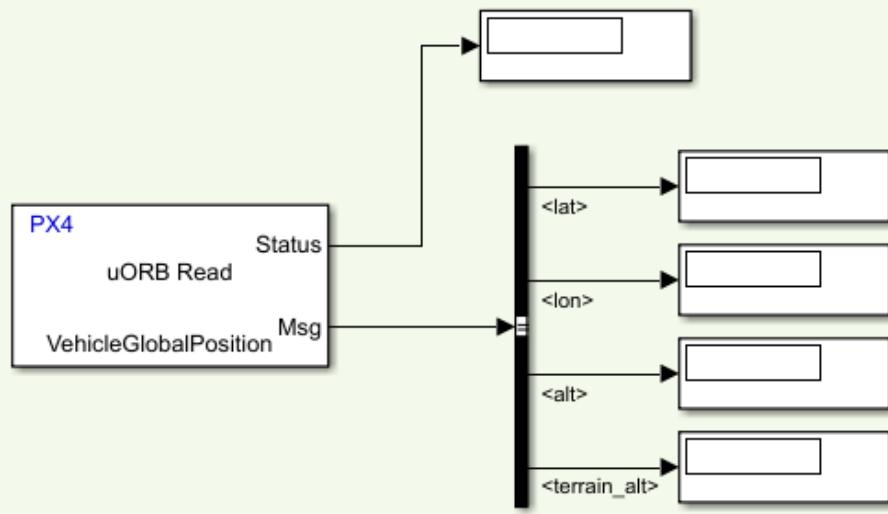
A pre-configured model (*px4demo_uORBReadWrite*) with publish/subscribe blocks is available for your reference. This model is configured with PX4 Host Target as the default board. When PX4 Host Target is set as the hardware board, you can perform Connected I/O simulation. Open the *px4demo_uORBReadWrite* model.

PX4 uORB Read and Write

Write uORB data



Read back uORB data



Reading GPS Data from PX4 Autopilot

This example shows how to obtain the data from a GPS device connected to PX4 flight controller, using UAV Toolbox Support Package for PX4® Autopilots.

The example uses a pre-defined Simulink model (*px4demo_readGPS*) that contains the GPS block. The GPS block gets the GPS data by reading the `sensor_gps` uORB message.

Prerequisites

- If you are new to Simulink®, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

To run this example, you will need the following hardware:

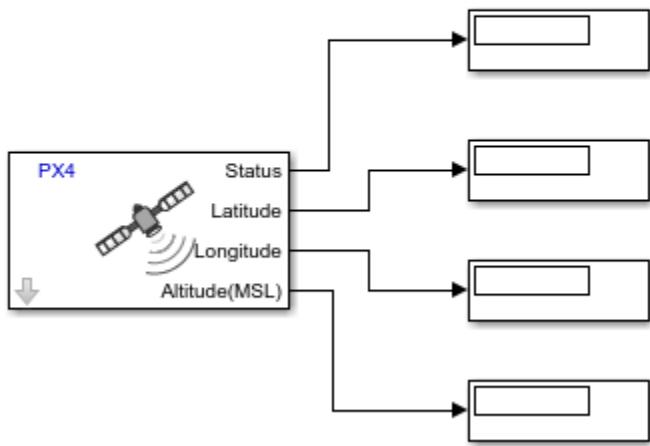
- Pixhawk® Series flight controller
- Micro USB type-B cable
- GPS device
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)
- Micro-SD card reader

Configure and Deploy the Model

In this task, you will configure and run the pre-defined Simulink model (*px4demo_readGPS*) to verify the GPS data. While running the model, you can select either Monitor and Tune option or the Connected IO option. This model contains the GPS block that reads `sensor_gps` uORB topic. After you verify the GPS data, you deploy the model to the PX4 hardware board.

1. Open the *px4demo_readGPS* model.

Read Vehicle GPS Data



In the example model, the GPS block is used to read GPS values. This block accepts signals from the `sensor_gps` uORB message, and outputs the required values.

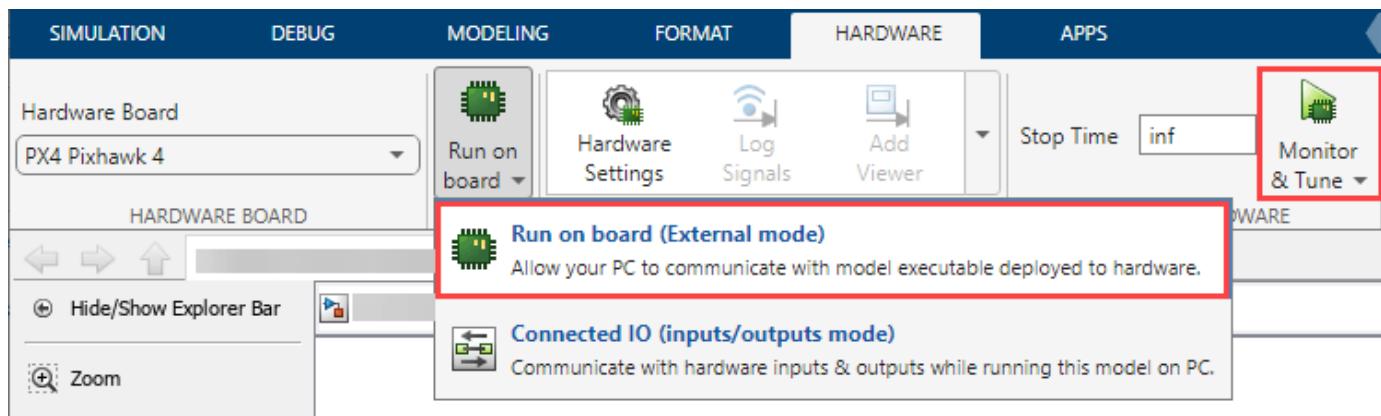
2. Double-click the GPS block. By default, signals corresponding to **Latitude**, **Longitude**, and **Altitude(MSL)** are selected as outputs. You can add the other signals as output by selecting the corresponding checkboxes.

Note: The **Status** output indicates if the uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the output is the latest, and a value of 1 indicates that the uORB data was received during the previous time step. This output can be used to trigger subsystems for processing new messages received in the uORB network.

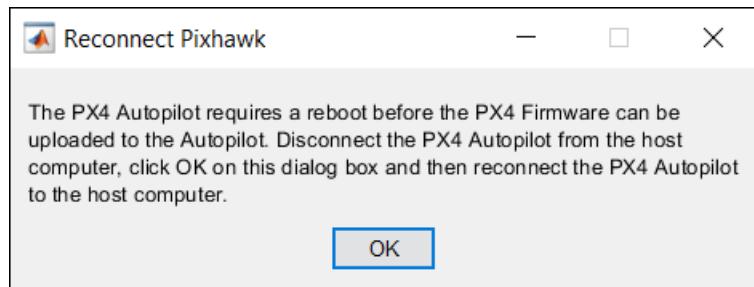
3. In the **Modeling** tab, click **Model Settings**.
4. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane:
 - Set the **Hardware board** to the same PX4 hardware board that you selected during Hardware Setup screens.
 - In the **Target Hardware Resources** section, set the **Build options** to **Build, load and run** to automatically download the generated binary file on to the connected Pixhawk Series flight controller.
 - Enter the serial port of the host computer to which the Pixhawk Series flight controller is connected, in the *Serial port for firmware upload* field.
 - In the **External mode** pane, select the option **Use the same host serial port for External mode as used for firmware upload**.
5. Navigate to **Solver** pane and select the option **Treat each discrete rate as a separate task**. Click **OK**.
6. In the **Simulation** tab, specify the stop time for parameter tuning simulation. The default value for the **Stop time** parameter is 10.0 seconds. To run the model for an indefinite period, enter **inf**.
7. Connect the USB cable from the PX4 flight controller to the host computer.

8. Run the model using one of the following options.

- **Monitor & Tune:** To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

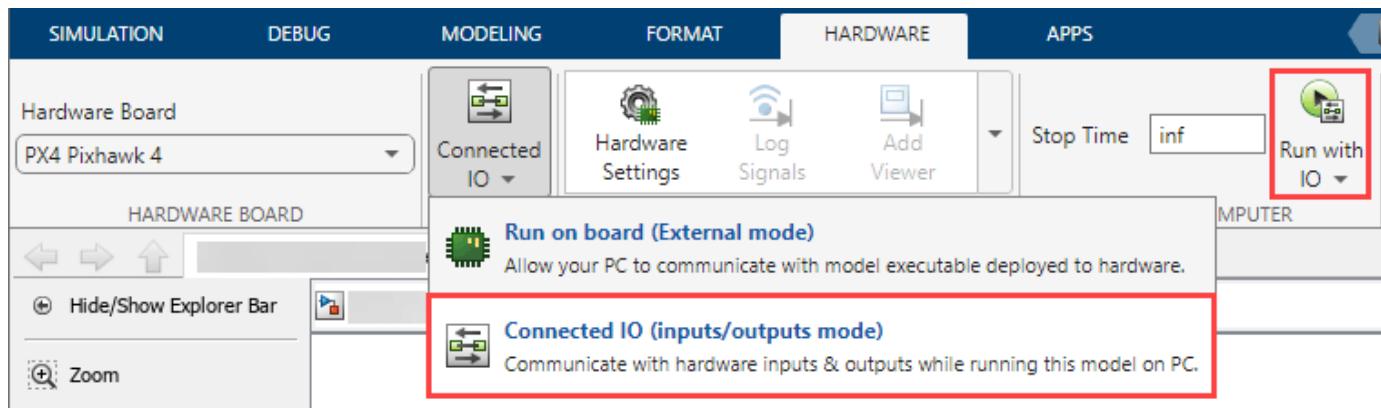


Wait for the code generation to be completed. Whenever the dialog box appears instructing you to reconnect the flight controller to the serial port, click **OK** and then reconnect the PX4 Autopilot on the host computer.



The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

- **Connected IO:** To run this model in the Connected I/O mode, on the **Hardware** tab, in the **Mode** section, select **Connected IO** and then click **Run with IO**.



If the Connected IO firmware is not deployed on the hardware, then the dialog box instructing you to reconnect the flight controller to the serial port appears otherwise the dialog box is not displayed. If the dialog box appears, click **OK** and then reconnect the PX4 Autopilot on the host computer.

9. Verify the output GPS values in the Scope display. The values correspond to the current GPS position of the PX4 flight controller.

10. Stop the Monitor and Tune operation by clicking **Stop** in the **Hardware** tab.

11. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. The model is deployed to the PX4 hardware board.

Getting Started with PX4 Analog Input Block for ADC Channels

This example shows you how to use the PX4 Analog Input block to read the voltage applied at the ADC pins on the Pixhawk® Series controller.

Introduction

UAV Toolbox Support Package for PX4 Autopilots enables you to create Simulink® models that read the voltage applied at the ADC pins on a Pixhawk Series controller.

A single PX4 Analog Input block measures the analog voltage at all the ADC channels on the Pixhawk Series controller, based on the supported FMU version of the hardware. The block outputs the voltages as a 1-by-12 array. Each value in the array corresponds to the analog voltage at a particular channel number.

In this example, you will learn how to use the PX4 Analog Input block to:

- Observe the analog voltage at all channels
- Detect which of the ADC channels of the FMU are exposed on the connected PX4 flight controller, and use the voltages from those channels only in the Simulink model.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

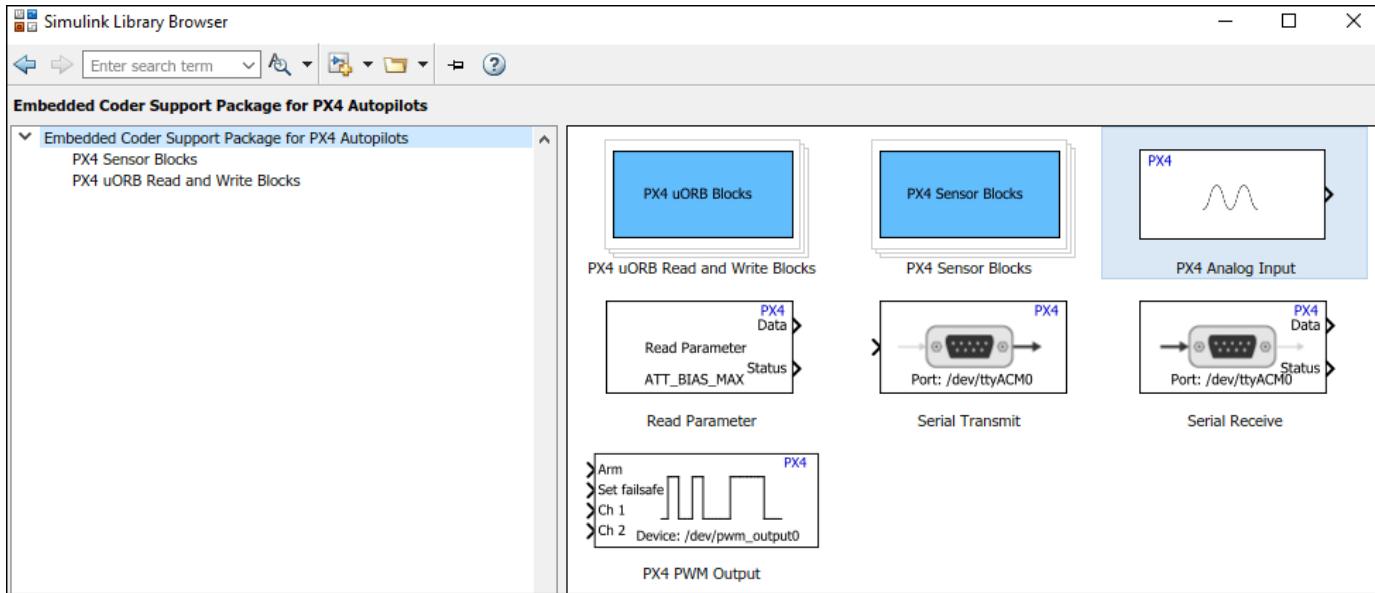
To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- 4-pin cable or any cable compatible with the ADC pins on the Pixhawk Series controller
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)

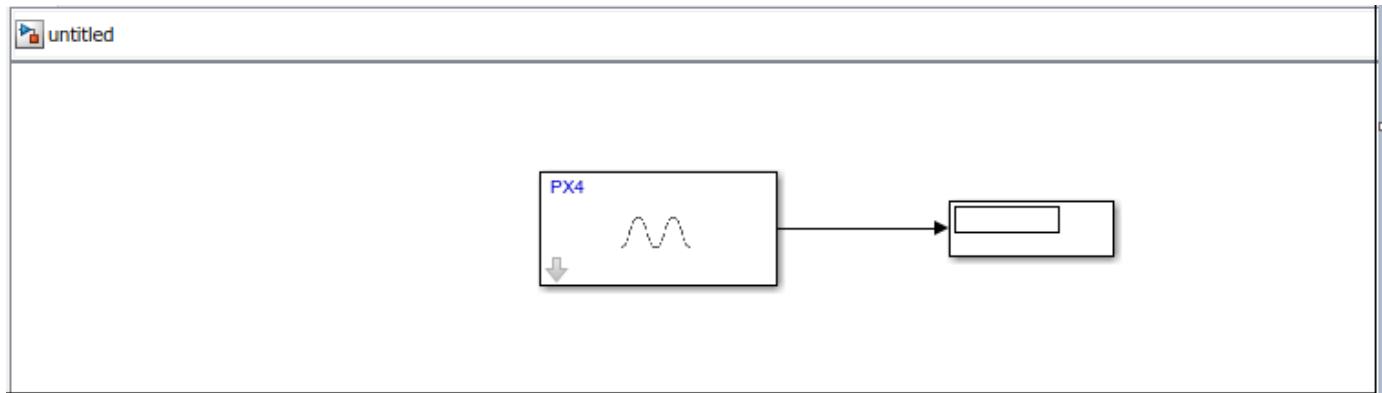
Task 1 - Observe ADC Channel Values

In this task, you will configure a PX4 Analog Input block to output analog data for all the ADC channels that are present on the FMU. All the ADC channels that are available on the FMU might not be exposed on the Pixhawk Series controller.

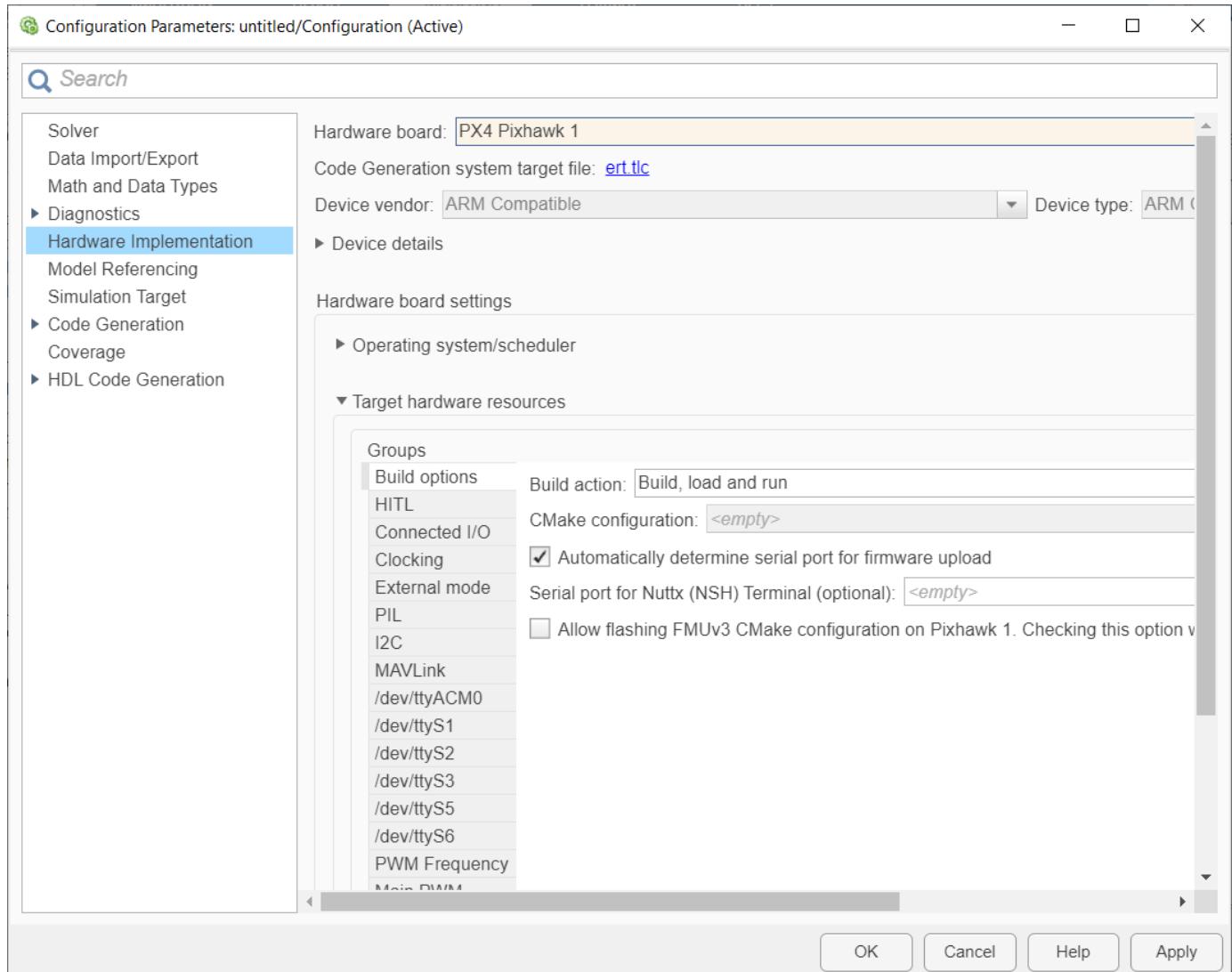
1. From the MATLAB® toolstrip, select HOME > New > Simulink Model to open the Simulink Start Page. Click **Blank Model** to launch a new Simulink model.
2. On the Simulink toolbar, select View > Library Browser to open the Simulink Library Browser. Click the UAV Toolbox Support Package for PX4® Autopilots tab (you can also type px4lib in MATLAB command window).



3. Drag and drop a **PX4 Analog Input** block to the model.
4. From the Simulink > Sinks tab in the Library Browser, drag and drop a **Display** block to the model. Connect the output of the PX4 Analog Input block to the Display block.



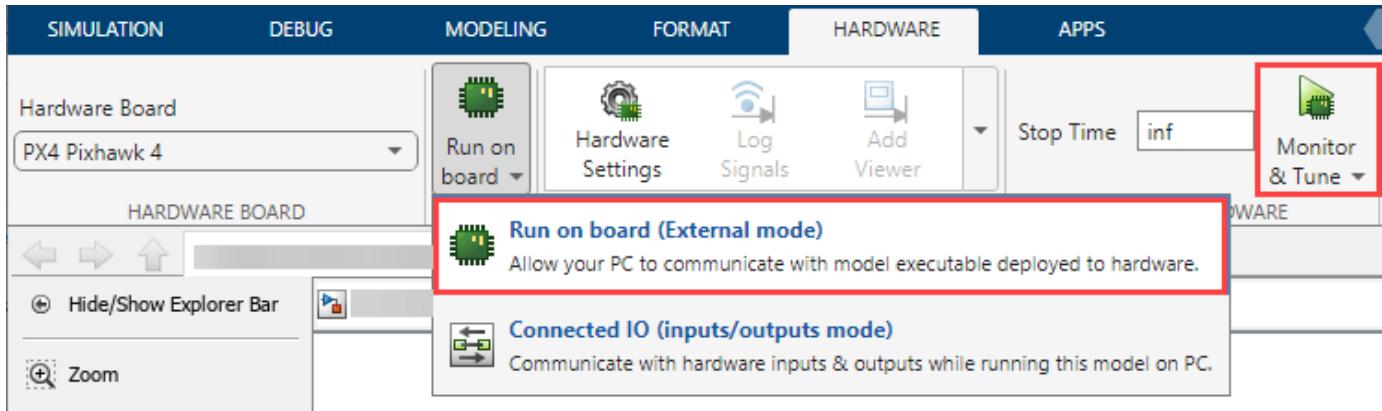
5. Connect the Pixhawk Series controller to the host computer using the USB cable.
6. In the **Modeling** tab of the Simulink toolbar, click **Model Settings**.
7. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane:
 - Set the **Hardware board** to the same Pixhawk series controller that you selected during Hardware Setup screens.
 - In the **Target Hardware Resources** section, enter the serial port of the host computer to which the Pixhawk Series controller is connected, in the *Serial port for firmware upload* field.
 - Click **Apply** and then **OK**.



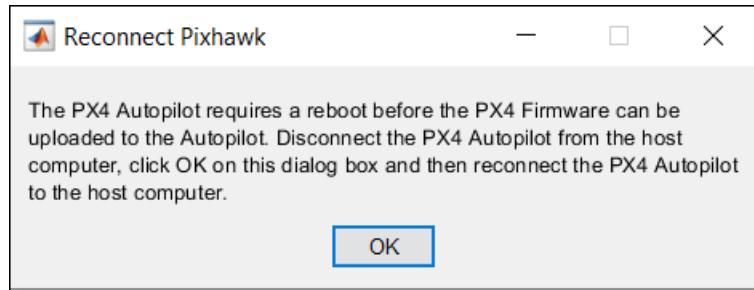
8. In the **Simulation** tab, specify the stop time for parameter tuning simulation. The default value for the **Stop time** parameter is 10.0 seconds. To run the model for an indefinite period, enter **inf**.

9. Run the model using one of the following options.

- **Monitor & Tune:** To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

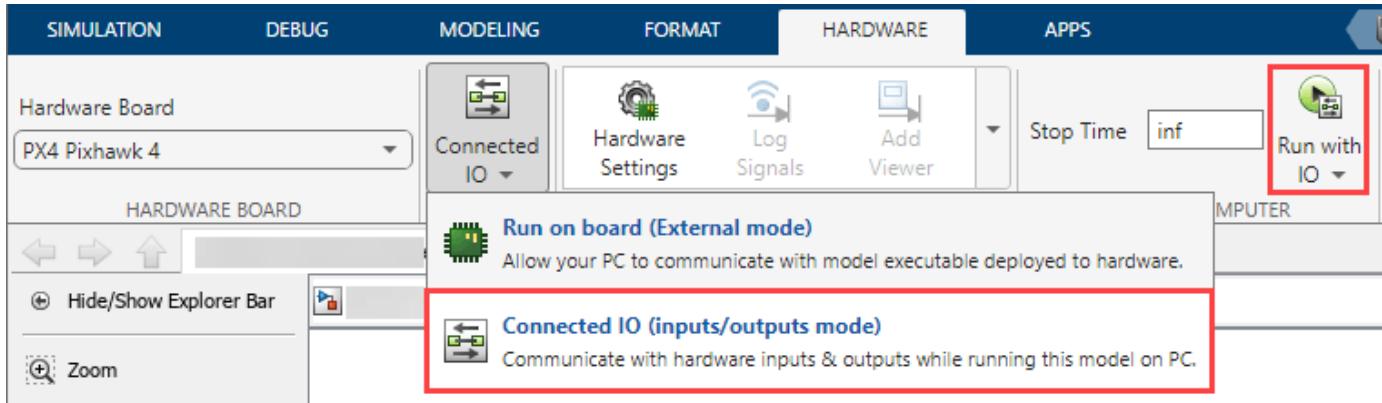


Wait for the code generation to be completed. Whenever the dialog box appears instructing you to reconnect the flight controller to the serial port, click **OK** and then reconnect the PX4 Autopilot on the host computer.



The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

- **Connected IO:** To run this model in the Connected I/O mode, on the **Hardware** tab, in the Mode section, select Connected IO and then click Run with IO.



If the Connected IO firmware is not deployed on the hardware, then the dialog box instructing you to reconnect the flight controller to the serial port appears otherwise the dialog box is not displayed. If the dialog box appears, click **OK** and then reconnect the PX4 Autopilot on the host computer.

10. Observe the voltage array values in the Display block. It is a 1-by-12 array that shows all the ADC channels based on the FMU version of the PX4 flight controller.

Task 2 - Detect ADC Channels That Are Exposed on Pixhawk Series Controller

In this task, you will determine which of the ADC channels that are provided by the FMU version are exposed on the Pixhawk Series controller, and use those index values for further processing of signals.

Ensure that the Monitor and Tune process is still running (as done in Task 1) with the PX4 Analog Input block and the Display block.

1. Use the cable to connect an ADC pin on the Pixhawk Series controller to +3.3V Vcc.
2. Observe if the value for any cell in the Display block changes to +3.3V approximately.

The index of this cell in the array is the index of ADC channel that is exposed on the flight controller.

3. Repeat steps 1 and 2 for all the other ADC channels that might be remaining in the Pixhawk Series controller.

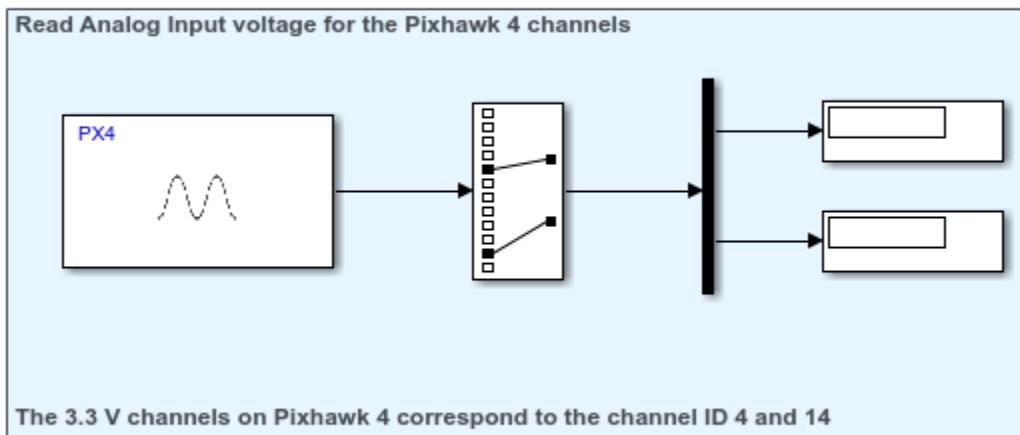
After you identify the index numbers of all exposed ADC channels, you can stop Monitor and Tune process and remove the Display block connected at the output of the PX4 Analog Input block. You can now proceed with extracting the signal values from the block.

4. From the Simulink > Signal Routing tab in the Library Browser, drag and drop a **Selector** block to the model.
5. Double-click the block, and set the *Input port size* to **12**. Set the value for **Index** based on the index values that you identified. For example, you can enter the **Index** values as [6 7 8].
6. Connect the input of the Selector block to the output of the PX4 Analog Input block.
7. Connect the output of the Selector block to a **Demux** block. The **Number of ports** in the Demux block can be set to the total number of ADC channels identified.

The output of the Demux block can be used for further processing of logic.

The complete model will look like the pre-configured model (px4demo_ADC) available for your convenience.

PX4 Analog Input



Copyright 2018-2022 The MathWorks, Inc.

See Also

PX4 Analog Input

Send and Receive Serial Data Using PX4 Autopilots Support Package

This example shows how to use UAV Toolbox Support Package for PX4® Autopilots to send and receive serial data with a Pixhawk® Series flight controller.

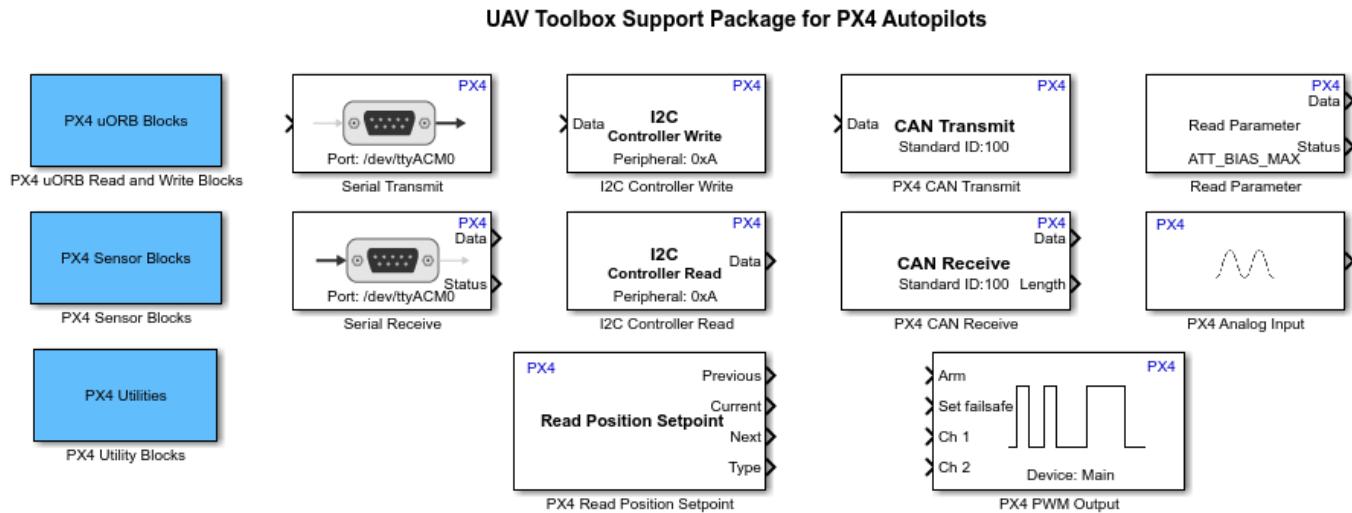
This example uses a simple protocol to send a data request from the host computer to the Pixhawk Series flight controller, and receives the requested data from the flight controller.

Introduction

The UAV Toolbox Support Package for PX4 Autopilots contain a Serial Receive and a Serial Transmit block that helps you to receive and send serial data over UART or USART port on the Pixhawk Series flight controller.

To view the blocks, enter `px4lib` at the MATLAB® prompt.

```
px4lib;
```

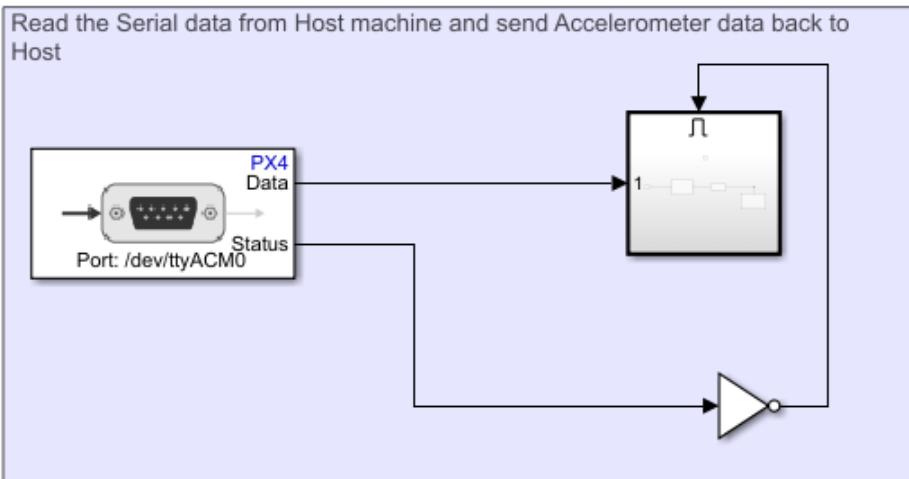


Copyright 2018-2024 The MathWorks, Inc.

In this example, we use the pre-configured PX4 Serial Transmit and Receive model(`px4demo_serial`), along with a custom script `getAccelerometerData.m` to send and receive serial data from the host computer. This model uses the Serial Transmit and Serial Receive blocks to exchange data.

- **PX4 Serial Transmit and Receive model:** In this model, the Pixhawk Series flight controller sends the accelerometer data to the host computer using Serial USB cable. The data is sent only upon receipt of a request from the host computer (which runs the custom script, `getAccelerometerData.m`). Open the `px4demo_serial` model.

PX4 Serial Transmit and Receive



This model works along with a [MATLAB Script](#) that is to be executed on the Host machine after deploying this model onto the Pixhawk board.

The [MATLAB Script](#) requests the board to send the accelerometer data back to the host.

Copyright 2019-2021 The MathWorks, Inc.

- `getAccelerometerData.m`: Using this script, the host computer requests the Pixhawk Series flight controller to send its accelerometer data, and it displays the accelerometer values upon receiving them. Run the following command in MATLAB to open the file:

```
edit(pwd, 'getAccelerometerData.m');
```

In this example, you will learn how to:

- Create and deploy a Simulink® model that can send and receive serial data.
- Create a simple protocol to exchange serial data.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)

Task 1 - Configure the Model for Pixhawk Hardware

1. Connect your Pixhawk® board to the host computer using the USB cable.
2. Open the PX4 Serial Transmit and Receive model(`px4demo_serial`). This model is configured to use the PX4 Pixhawk Series boards.

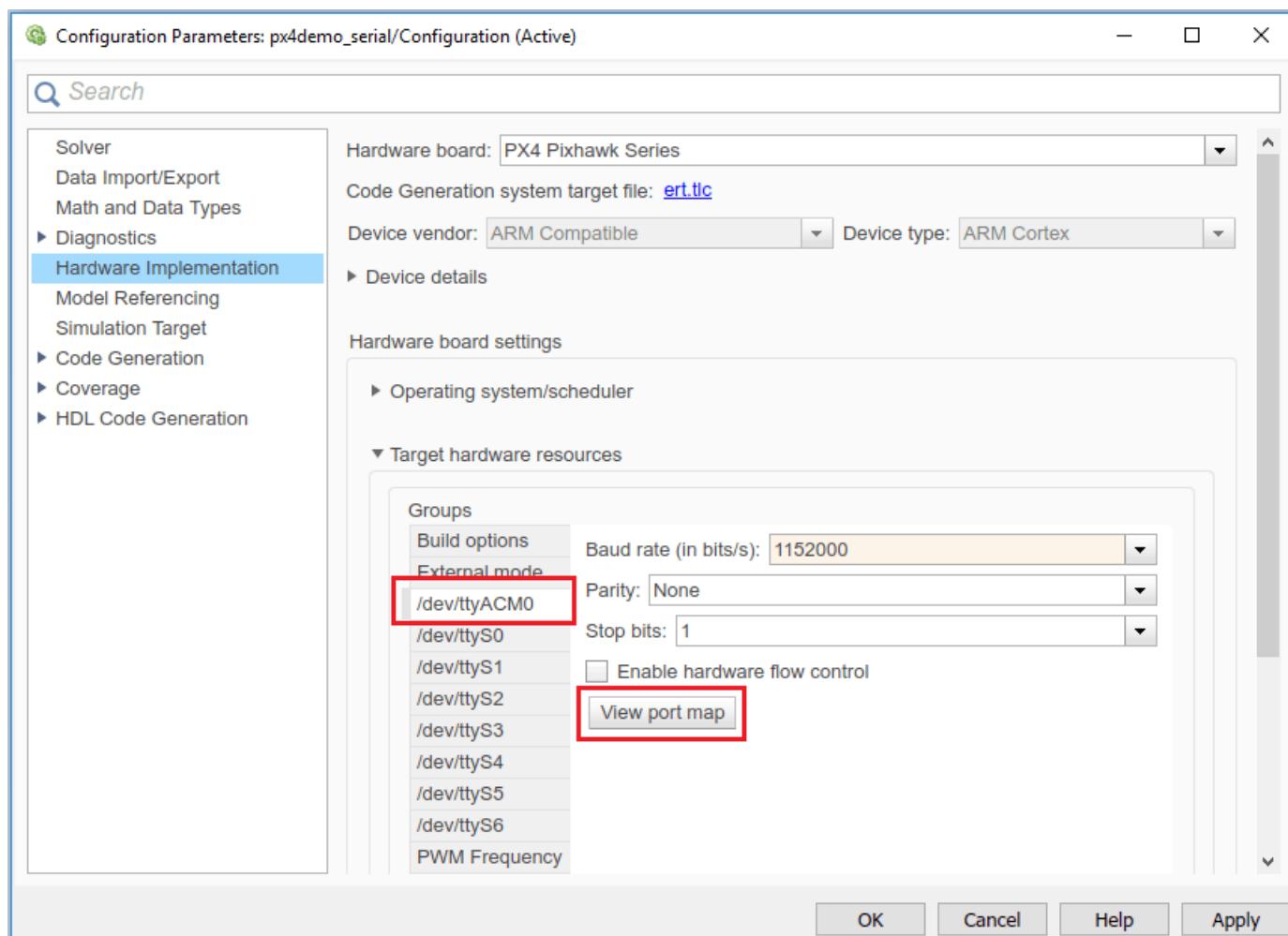
3. In the **Modeling** tab, click **Model Settings**.

4. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane:

- Set the **Hardware board** to the same Pixhawk series controller that you selected during Hardware Setup screens.
- In the **Target Hardware Resources** section, open the **Build options** pane, and enter the serial port of the host computer to which Pixhawk Series flight controller is connected, in the *Serial port for firmware upload* field.

5. From the Groups list under Target hardware resources, select /dev/ttyACM0.

6. To know the mapping between the serial ports mentioned on the board, click **View port map**.



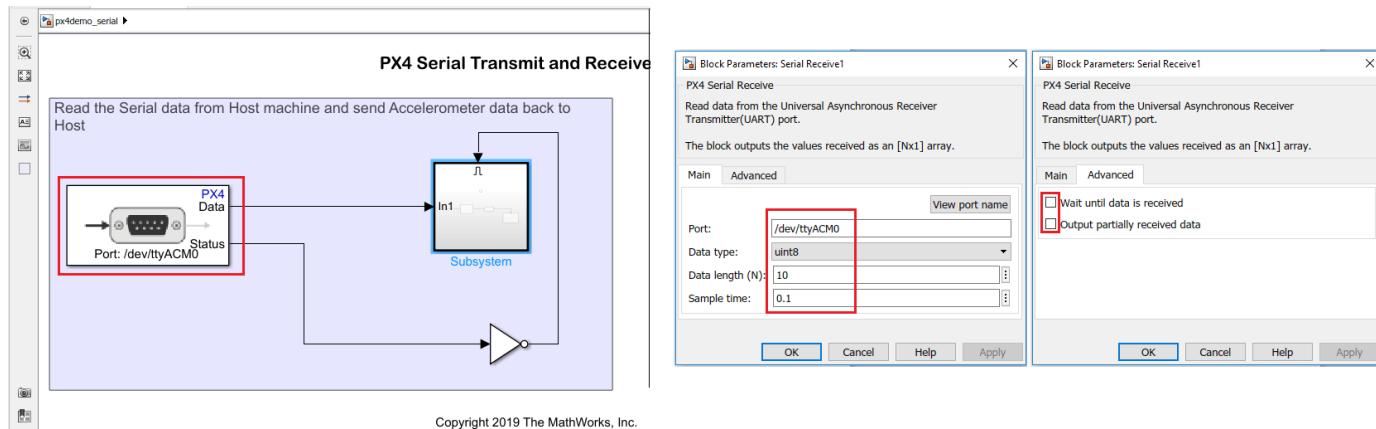
7. Set the Baud rate to **115200**, Parity to **None** and Stop bits to **1**.

8. Click **Apply**, and then **OK** to close the dialog box.

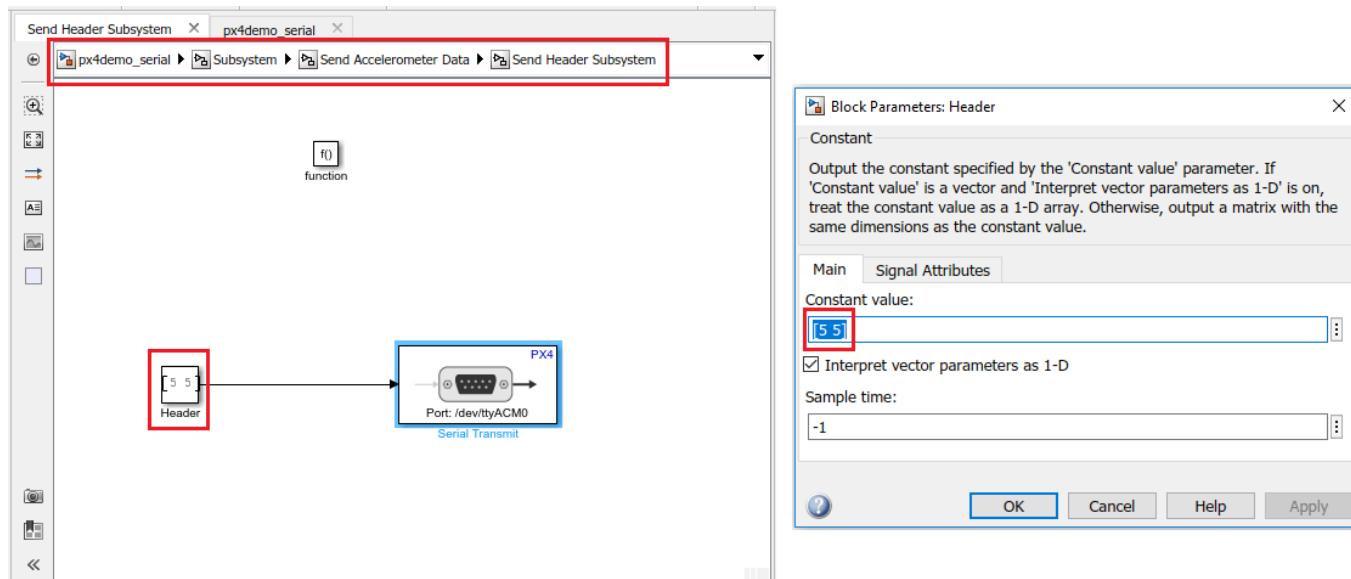
Task 2 - Configure the Blocks in the Simulink Model

In the Simulink model, double-click the following blocks and verify that the parameter values specified are the same as shown below:

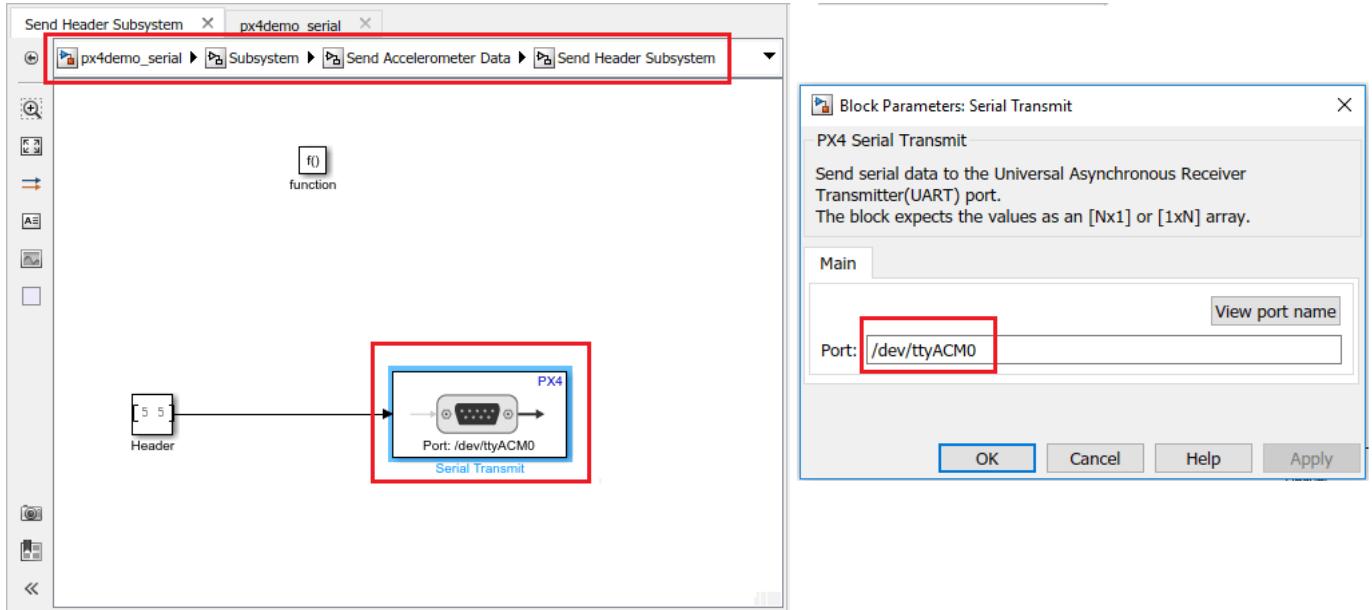
- Serial Receive block in the main model



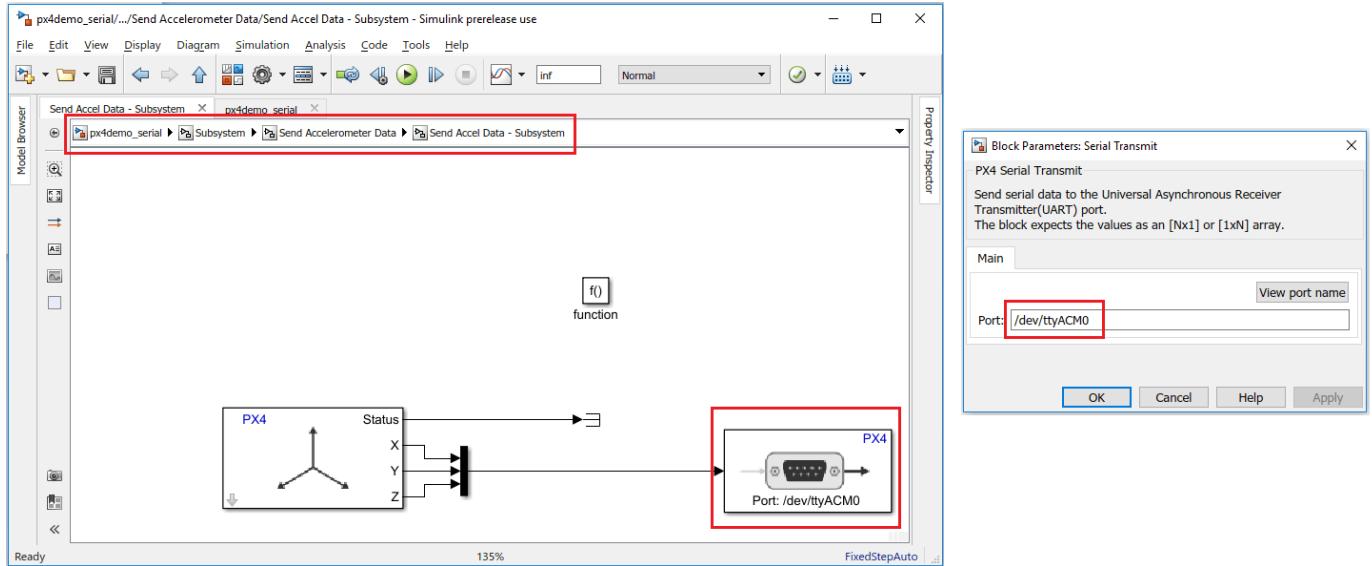
- Header block (Constant) block in the Send Header subsystem



- Serial Transmit block in the Send Header subsystem



*Serial Transmit block in the Send Accel Data subsystem



Note: The Header (Constant block inside Send Header Subsystem) must be set to **[5 5]** of *uint8* data type. This is because the *getAccelerometerData.m* script, which receives the data from PX4 Serial Transmit and Receive model, expects the header to be **[5 5]**.

Note: The **Sample time** parameter specified in Header (Constant block inside Send Header Subsystem) and in the PX4 Accelerometer block (inside Send Accel Data Subsystem) must be set to **-1**.

Understand the Protocol

This model uses a simple protocol to exchange data between the hardware and host computer.

The getAccelerometerData.m script in MATLAB sends a request to the Pixhawk Series flight controller. The request data sent from the host computer is in the form of HEADER_HOST+DATA_REQUEST.

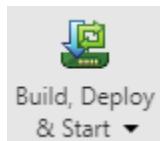
The PX4 Serial Transmit and Receive model running on the Pixhawk hardware receives the above data and parses it to get the DATA_REQUEST by stripping the HEADER_HOST. Upon validation that request is genuine, the Pixhawk hardware sends the accelerometer data in the form of HEADER_PIXHAWK+ACCEL_DATA back to the host computer.

The getAccelerometerData.m executing on MATLAB receives the above data and parses it to get the ACCEL_DATA by stripping the HEADER_PIXHAWK.

Task 3 - Deploy Model to Hardware and Get Accelerometer Values in MATLAB

1. Open the PX4 Serial Transmit and Receive model.

2. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**.



The lower left corner of the model window displays the status while Simulink prepares, downloads, and runs the model on the hardware.

3. Open the `getAccelerometerData.m` in MATLAB. Ensure that the comport defined in Line 14 is the same as the serial port that you entered in the *Build options* pane (of the Configuration Parameters dialog box), as mentioned in Task 1, Step 4 above.

4. Execute the `getAccelerometerData.m` script in MATLAB to view the accelerometer values. For example, the accelerometer values may look like the following:

```
Accelerometer data (x | y | z) in m/s^2: 0.12 | 0.03 | 9.74
```

5. Change the orientation of the Pixhawk hardware board and repeat Step 4 to see the updated accelerometer values.

Other Things to Try

- Enhance the example by using a Header-Terminator protocol to exchange data between Pixhawk hardware and host computer.
- Configure the **PX4 Serial Transmit and Receive** model to exchange data over other serial ports of the Pixhawk Series controller.
- Create a Simulink model that uses **Serial Receive** and **Serial Send** blocks from **Instrument Control Toolbox** to exchange serial data between host computer and Pixhawk Series flight controller.

Getting Started with PWM Blocks for PX4 Autopilots

This example shows you how to use the PX4 PWM Output block to generate signals on the PWM pins of a Pixhawk Series controller, and verify the PWM values.

Introduction

UAV Toolbox Support Package for PX4® Autopilots enables you to generate PWM signals on the PWM outputs of the Pixhawk Series controller which are connected to ESC and thus drive the motors.

In this example, you will learn how to create and run a Simulink® model to generate signals on the PWM outputs.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

To run this example, you will need the following hardware:

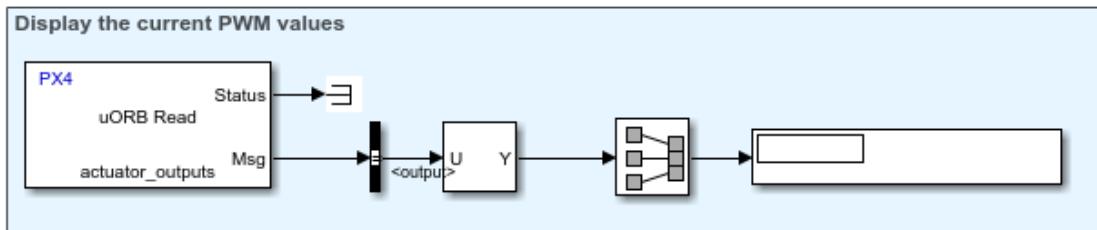
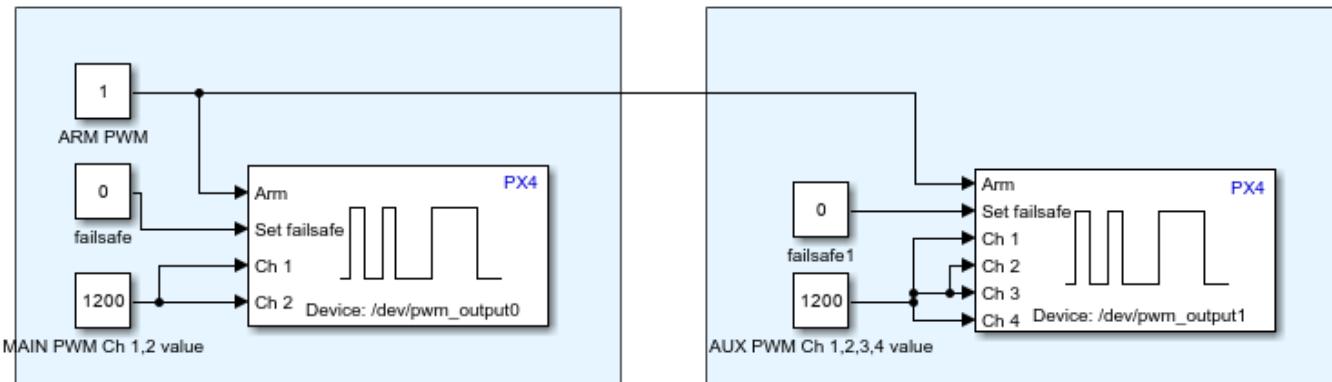
- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card (already used during the initial Hardware Setup)
- Safety switch (if required by the Pixhawk Series flight controller)
- Cathode Ray Oscilloscope (CRO)

Model

You will use Simulink to create PWM signals on the PWM outputs. For verification purposes, you can connect the PWM output pins of the Pixhawk Series hardware to a CRO to see the pulse width.

You will be creating the below px4demo_PWM model for generating PWM signals:

PX4 PWM



Copyright 2018-2022 The MathWorks, Inc.

Task 1 - Configure the Simulink Model for PWM

- From the MATLAB® toolbar, select HOME > New > Simulink Model to open the Simulink Start Page. Click **Blank Model** to launch a new Simulink model.
- In the model window, from the **Simulation** tab select **Library Browser** to open the Simulink Library Browser. Click the **UAV Toolbox Support Package for PX4 Autopilots** tab (you can also type *px4lib* at the MATLAB command prompt).
- Drag a PX4 PWM Output block to the model. Double-click the block to open the Block Parameters dialog box and to change the block properties.
- For generating PWM signals for the Main PWM channels, select **/dev/pwm_output0** as the PWM device. For the AUX channels, select **/dev/pwm_output1**.
- Select the channels for which you want to generate the PWM signals. A Pixhawk Series controller can have a maximum of 8 PWM channels for Main/AUX.

Note: It is not necessary that all 8 PWM channels will be present on the Pixhawk® hardware. For example, Pixracer has only 6 Main PWM channels and Pixhawk 1 has 6 AUX channels. Ensure that the channels that you select on the PX4 PWM Output Block Parameters dialog box exist on the Pixhawk board.

- Click **Apply** and then **OK** to close the dialog box.

7. From the Simulink > Sources tab in the Library Browser, drag and drop three Constant blocks into the model.

8. Connect the output port of the first Constant block to the **Arm** input of the PX4 PWM Output block.

9. Double-click this Constant block (which is the **Arm** input of the PWM block), set the value of the Constant block as **1**, and set the Output data type as **boolean**.

10. Connect the output port of the second Constant block to the **Set failsafe** input of the PWM block.

11. Double-click this Constant block (which is connected to **Set failsafe** input), set the value of the Constant block as **0**, and set the Output data type as **boolean**.

12. Connect the output port of the third Constant block to selected PWM channels input of the PWM block (for example, Ch1, Ch2, and so on).

Note: To select the required PWM channels as inputs, double-click the PX4 PWM Output block. In the Block Parameters dialog box, ensure that you select all the channels that belong to the same channel group. If you need to connect to PWM channels that are in different groups, select all channels in all the desired groups. Only one PX4 PWM Output block per channel category (Main or AUX) is allowed in the entire Simulink model.

Tip: To identify the channel groups in the Pixhawk Series flight controller board, use QGroundControl application on your host computer. For details, see PX4 PWM Output (refer to the sub-topic: Parameters > PWM Channels).

13. Double-click this Constant block (which is connected to the PWM channel inputs), set the value of the Constant block as **1400**, and set the Output data type as **uint16**.

Task 2 - Read the PWM block Using uORB Blocks

In this task, you will configure the model created in Task 1 to read the written PWM values, using the uORB topic **actuator_outputs**.

Note: The PWM values are published to the uORB topic **actuator_outputs** for the Main channels only. Also, the Pixhawk Series flight controller must have a **px4io** processor to read the **actuator_outputs** topic values. For example, Pixracer does not have the **px4io** processor, and does not read the **actuator_outputs** values.

1. From the UAV Toolbox Support Package for PX4 Autopilots tab in Library Browser, drag a PX4 uORB Read block to the model. Double-click the block.

2. Click **Select** next to the **Topic to subscribe to** field, and select **actuator_outputs.msg** from the pop-up window. Click **OK** to close the dialog box.

3. From the Simulink > Signal Routing tab in the Library Browser, drag a Bus Selector block to the model.

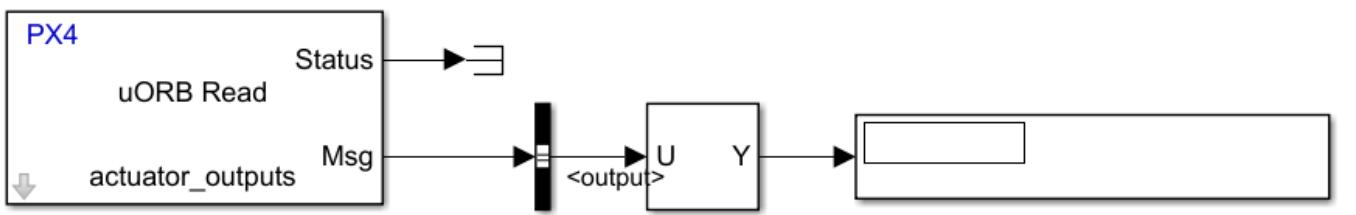
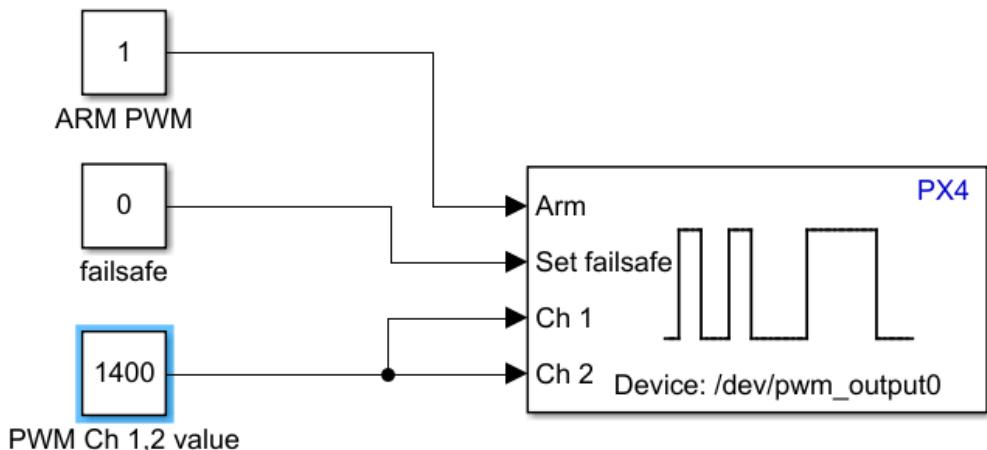
4. Connect the **Msg** output of the PX4 uORB Read block to the input port of the Bus Selector block.

5. Double-click the Bus Selector block. Add the **output** signal to the block output. Then, remove **signal1** and **signal2** from the list of output signals.

Note: If the **output** signal is not listed as an element of the input bus, close the dialog box. To ensure that the bus information is propagated, on the **Modeling** tab, click **Update Model**. Then, reopen the dialog box.

6. From the Simulink > Sinks tab in the Library Browser, drag a Display block to the model. Connect the output of the Bus Selector block to the Display block.

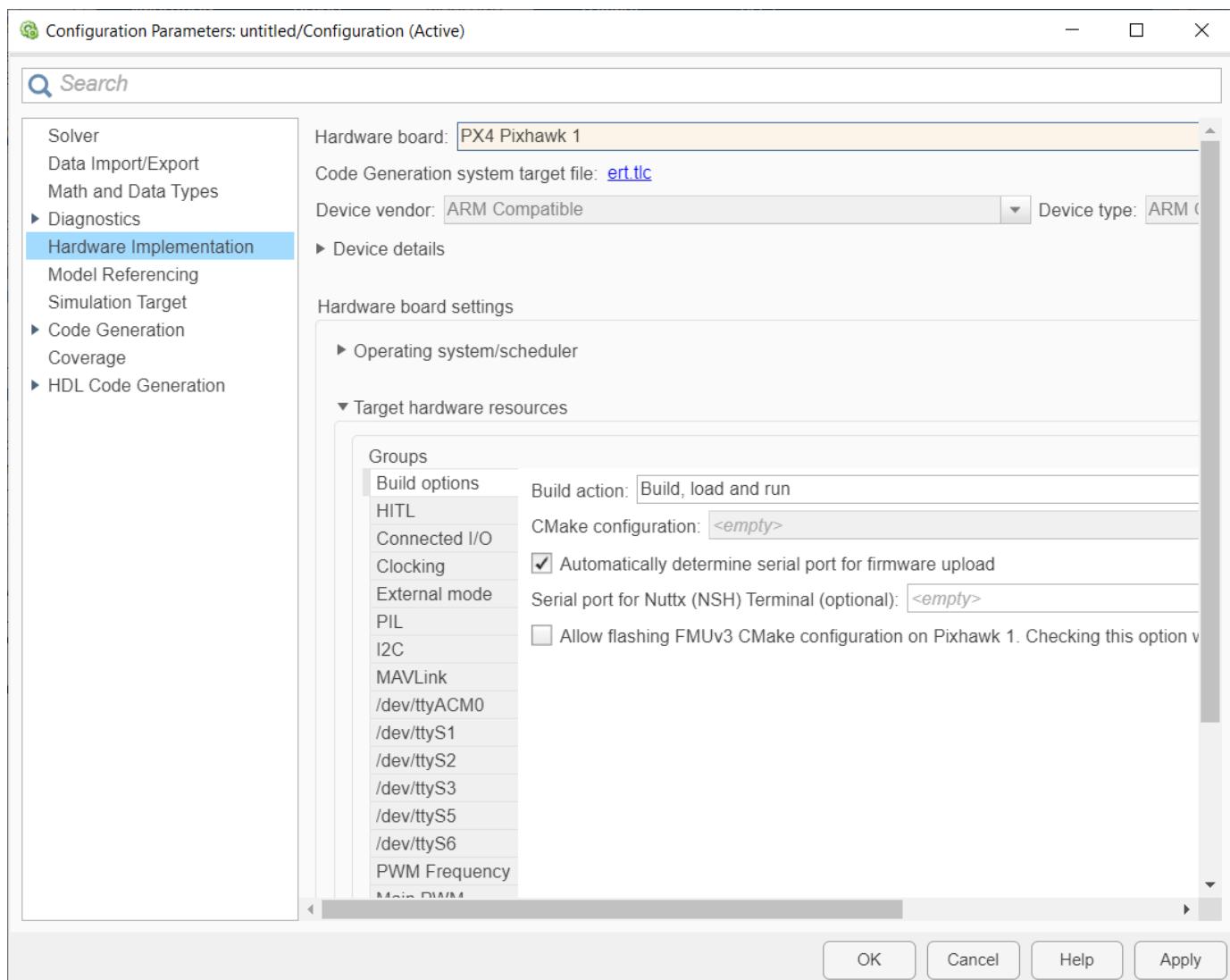
Your completed Simulink model should look like this:



Task 3 - Configure the PWM Frequency and PWM Limits

In this task, you will configure and run your model.

1. Connect the Pixhawk Series controller to the host computer using micro-USB cable.
2. Open Configuration Parameters dialog box, and set the Target Hardware as the Pixhawk Series controller you have selected during Hardware Setup screens.
3. In the **Target Hardware Resources** section, enter the serial port of the host computer to which the Pixhawk Series controller is connected, in the **Serial port for firmware upload** field. Click **Apply**.



4. In the **PWM Frequency** pane, set the frequency for the PWM signals as 400 Hz for the Main PWM Channels.

Note: Limit for setting PWM signal frequency is 50-400 Hz.

5. You can also set the Disarmed and Failsafe values for the Main and AUX PWM channels, in the **Main PWM** or **AUX PWM** panes.

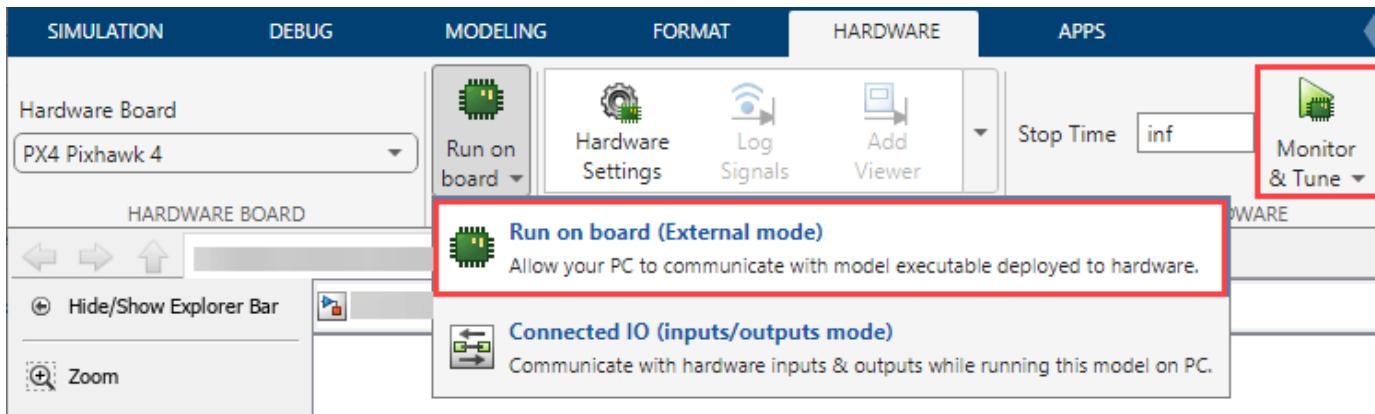
6. Click **Apply** and then **OK** to close the Configuration Parameters dialog box.

Task 4 - Connect the Pixhawk Series Controller and Generate PWM

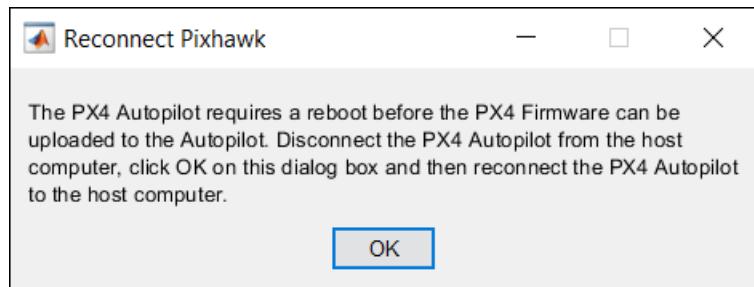
1. Connect the PWM channels of the Pixhawk Series controller for which you want to generate PWM signals to a CRO (these are the channels you selected in Task 1).
2. If your Pixhawk Series controller has a safety switch, connect the safety switch to the board.
3. In the **Simulation** tab, specify the stop time for parameter tuning simulation. The default value for the **Stop Time** parameter is 10.0 seconds. To run the model for an indefinite period, enter **inf**.

4. Run the model using one of the following options.

- **Monitor & Tune:** To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

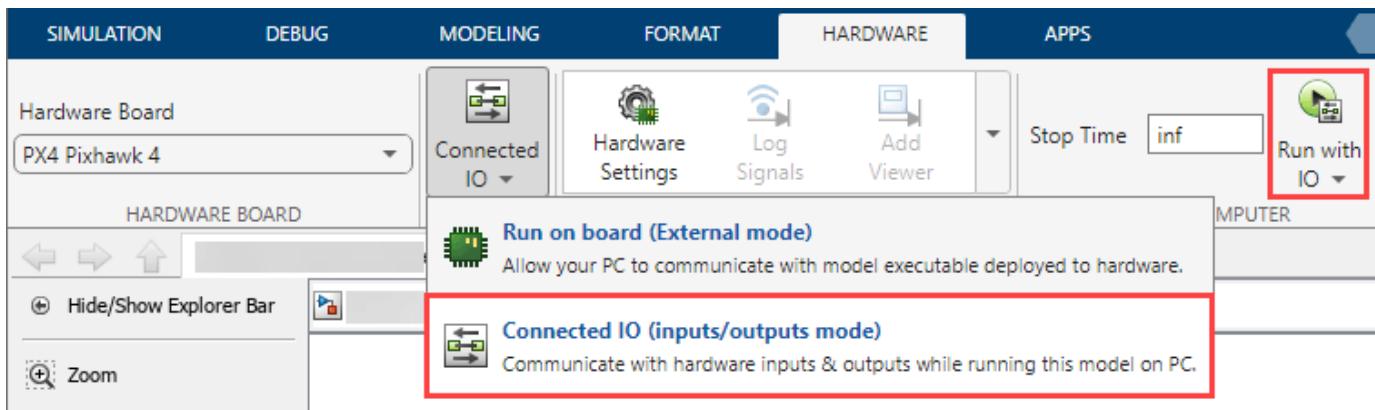


Wait for the code generation to be completed. Whenever the dialog box appears instructing you to reconnect the flight controller to the serial port, click **OK** and then reconnect the PX4 Autopilot on the host computer.



The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

- **Connected IO:** To run this model in the Connected I/O mode, on the **Hardware** tab, in the **Mode** section, select **Connected IO** and then click **Run with IO**.

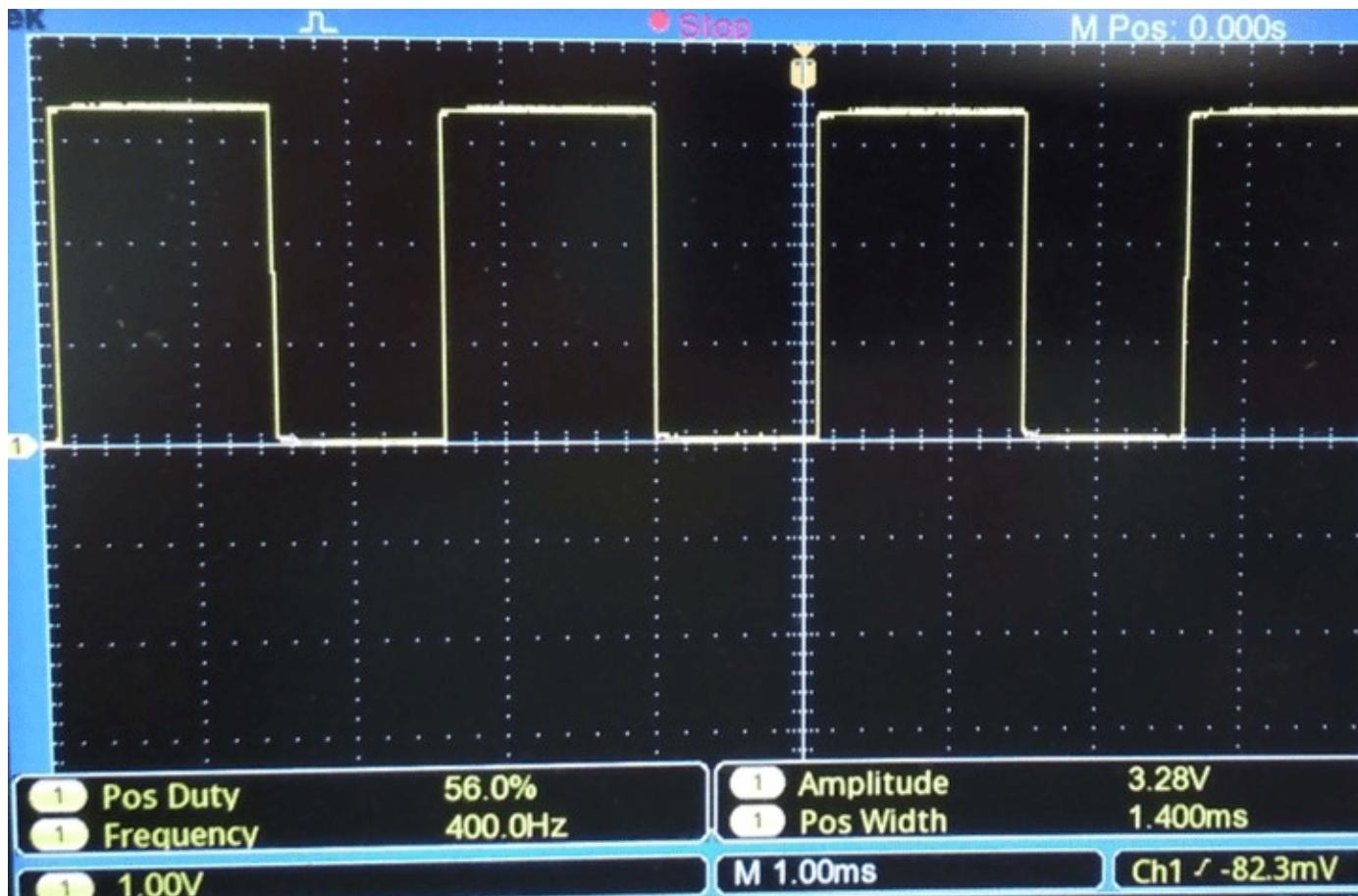


If the Connected IO firmware is not deployed on the hardware, then the dialog box instructing you to reconnect the flight controller to the serial port appears otherwise the dialog box is not displayed. If the dialog box appears, click **OK** and then reconnect the PX4 Autopilot on the host computer.

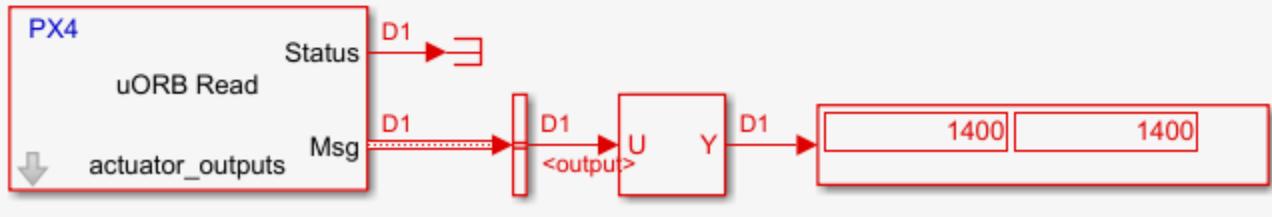
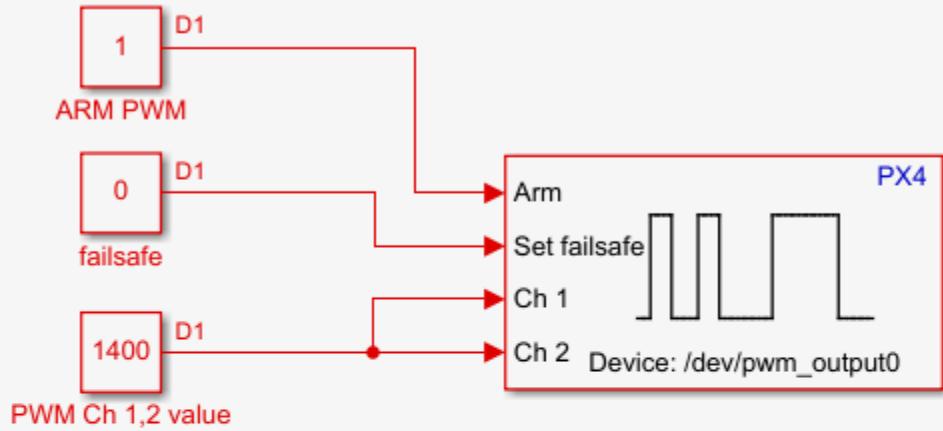
5. Observe that the PWM waveforms are not yet appearing in the CRO.
6. Press the safety switch and keep it pressed for more than 3 seconds. Release it after 3 seconds.

Note: The PWM values are published to the uORB topic `actuator_outputs` for the Main channels only. Also, the Pixhawk Series flight controller must have a `px4io` processor to read the `actuator_outputs` topic values (for example, Pixracer does not have the `px4io` processor, and does not read the `actuator_outputs` values).

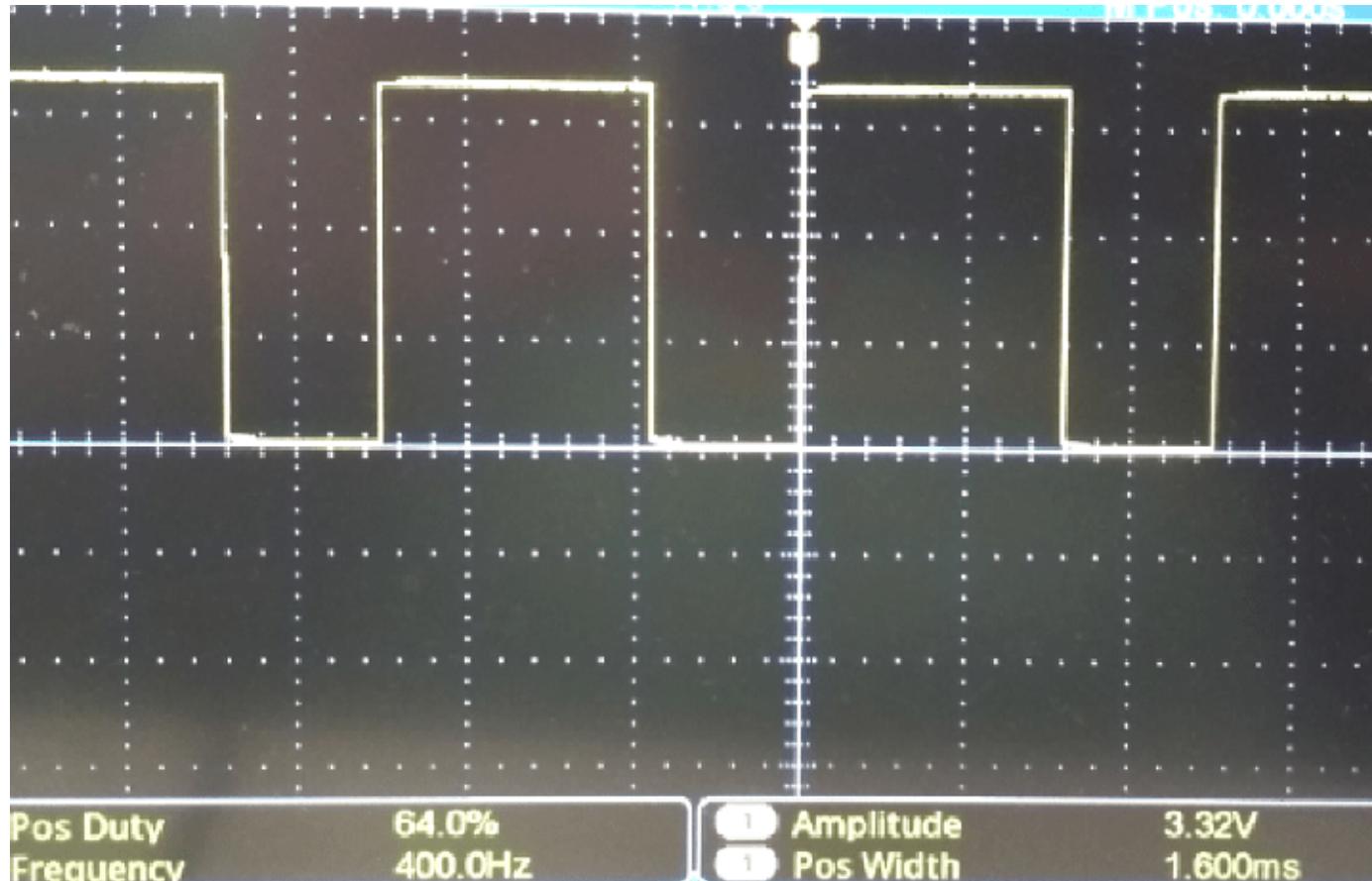
Now you should be able to see the PWM waveform of frequency 400 Hz with an ON time value of 1400us (as entered in the Constant block), on the CRO.



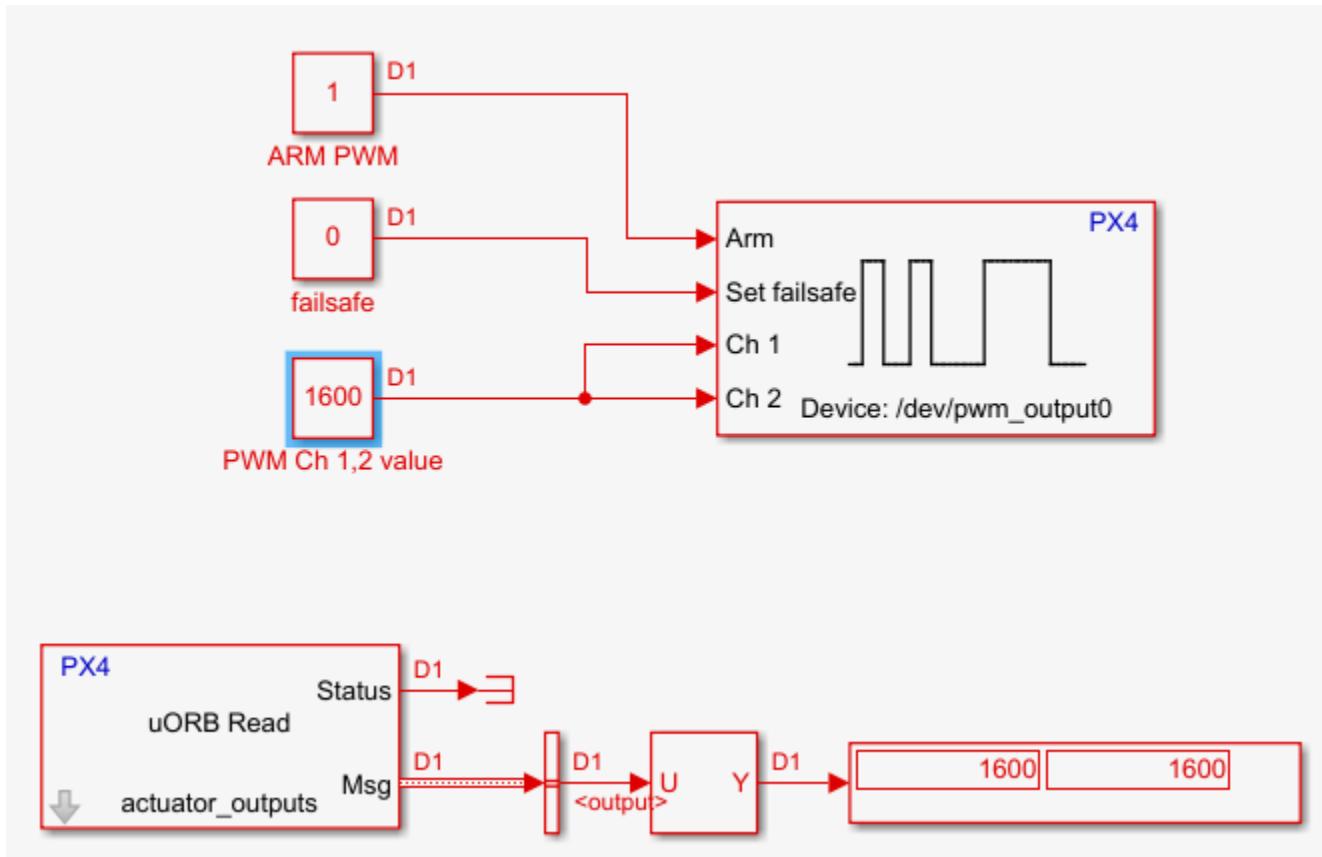
The ON-time value written to the PWM block is published to the `actuators_output` topic, and also appears in the Display block in the model.



7. Double click on the Constant block connected to the Channel inputs of the PWM blocks. Change the value to 1600 and click OK. On the CRO, you should see a waveform with an ON time value of 1600us.



The ON-time value written to the PWM block is published to the actuators_output topic, and also appears in the Display block in the model.



8. Disarm the PWM signals by changing the value of Constant block connect to **Arm** input of PWM block to 0. The pulse width of the PWM waveforms will change to the disarmed values set in the Model Configuration Parameters. You can arm the PWM again by setting the value as 1.

9. You can set the failsafe mode by setting the value of Constant block connect to **Set failsafe** input as 1. The pulse width of the PWM waveforms will change to the failsafe values set in the Model Configuration Parameters. You can remove the failsafe mode by setting the value as 0. If the *Force terminate failsafe mode* option is selected in PWM Block Parameters dialog box, the failsafe mode becomes permanent once **Set failsafe** is set.

Read PX4 System Parameters Using PX4 Autopilots Support Package

This example shows you how to read the PX4® system parameters using UAV Toolbox Support Package for PX4 Autopilots.

Introduction

In this example, the pre-configured model (*px4demo_Read_Parameter*), is used to demonstrate how to read the Gyroscope calibration parameters like offset and scaling, and use them along with raw Gyroscope values from a PX4 flight controller.

The PX4 flight controller uses many parameters to store and access during various operations. Much of these include sensor/actuator calibration data and are stored in flash memory which is accessible by MTD via NuttX.

You can read the default parameters or follow the guide to create your own parameters.

In this example, you will learn how to create and deploy a Simulink® model to read PX4 system parameters.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

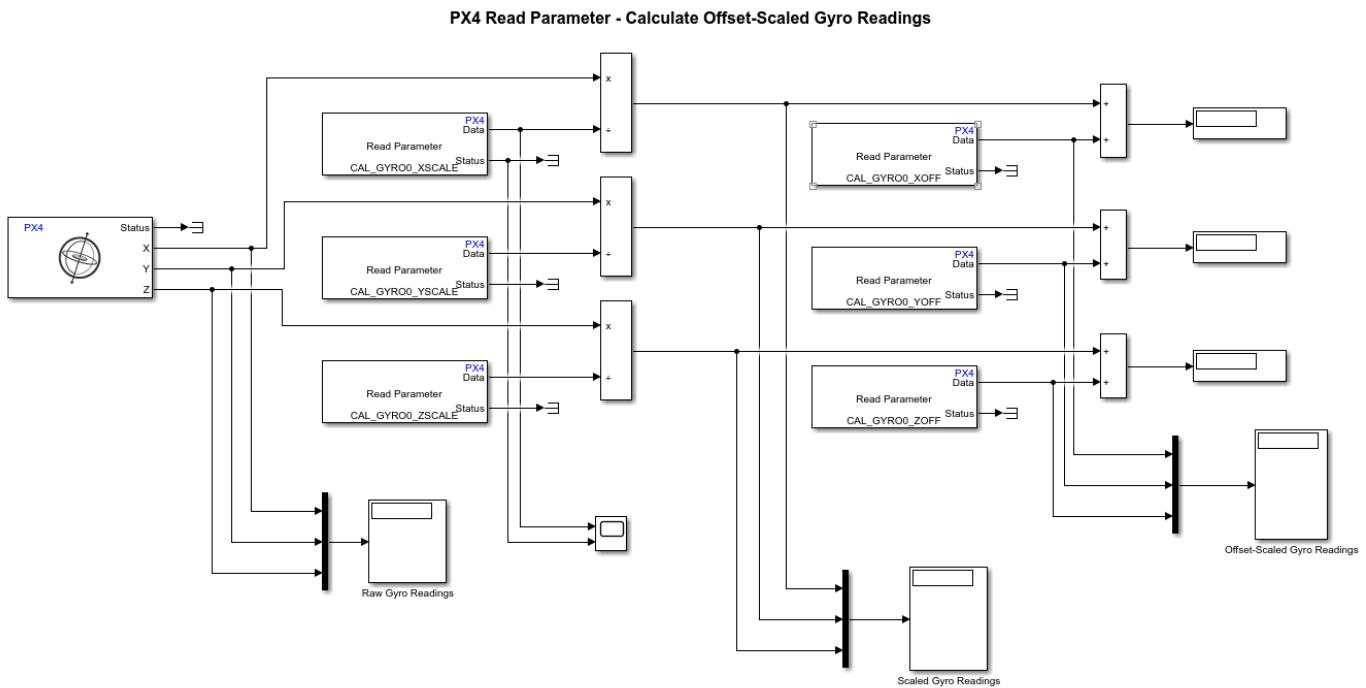
To run this example, you will need the following hardware.

- Supported PX4 Autopilot
- Micro USB type-B cable
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)

Task 1 - Configure the Model for Pixhawk Hardware

1. Connect your Pixhawk® board to the host computer using the USB cable.
2. Open the *px4demo_Read_Parameter* model.

This model is configured to use the PX4 Pixhawk Series boards, and it contains six PX4 Parameter Read blocks that can read six different PX4 system parameters.



3. To configure the model, go to the **Modeling** tab and click **Model Settings**.
4. In the Configurations Parameters dialog box, select **Hardware Implementation**.
5. Set the Target Hardware as the PX4 Autopilot hardware you have selected during hardware setup screens.
6. From the Groups list under Target hardware resources, select **Build options**. In the *Serial port for firmware upload* field, enter the serial port to which the hardware PX4 flight controller is connected to host computer.
7. From the Groups list under Target hardware resources, select External mode. In the hardware board serial port, select **/dev/ttyACM0**.
8. Click **Apply** and then **OK** to close the dialog box.

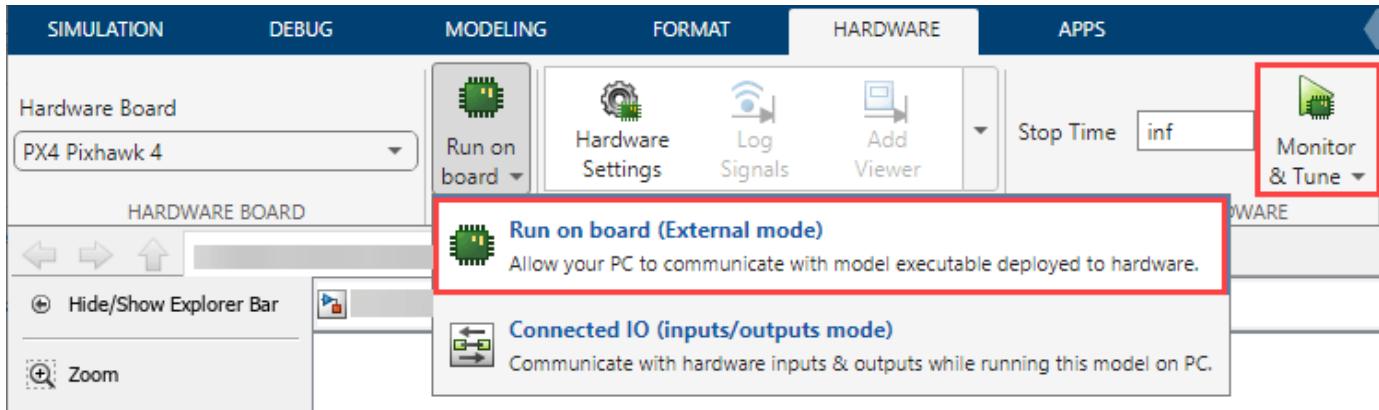
Task 2 - Configure and Run the Model

In this task, you will configure and run the Simulink model. While running the model, you can select either Monitor and Tune option or the Connected IO option. For more information, see “Monitor and Tune the Model Running on PX4 Autopilots” and “Communicate with Hardware Using Connected I/O”.

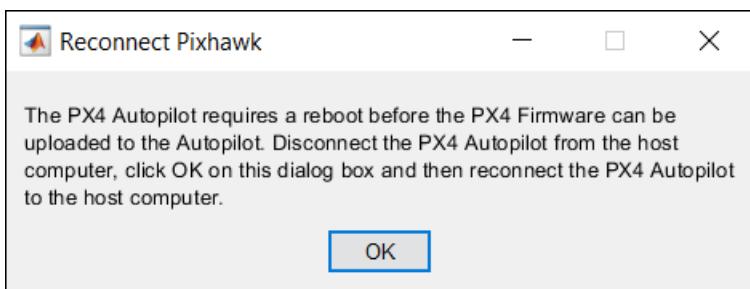
Perform these steps to configure and run the model.

1. In the **Simulation** tab, specify the stop time for parameter tuning simulation. The default value for the **Stop time** parameter is 10.0 seconds. To run the model for an indefinite period, enter **inf**.
2. Run the model using one of the following options.

- **Monitor & Tune:** To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

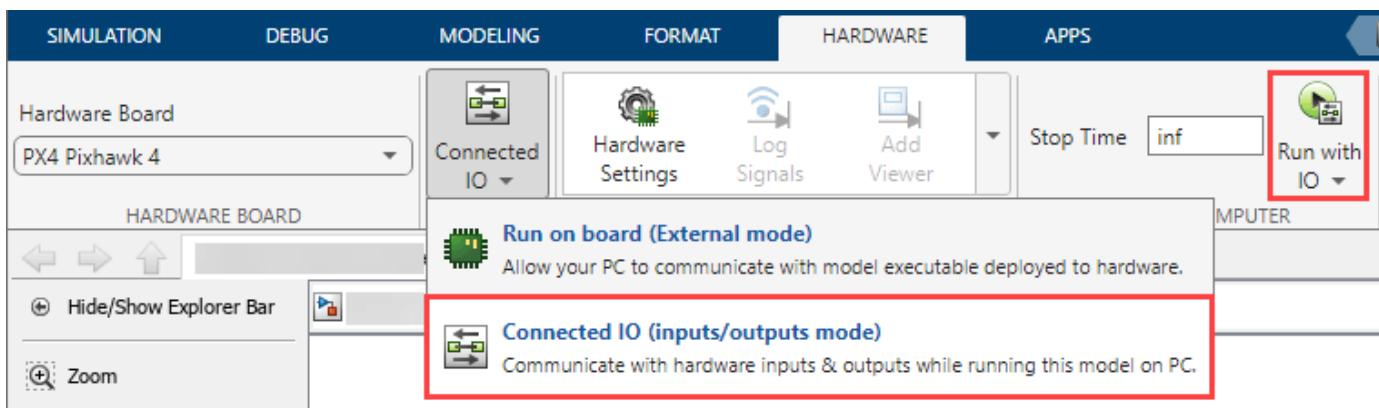


Wait for the code generation to be completed. Whenever the dialog box appears instructing you to reconnect the flight controller to the serial port, click **OK** and then reconnect the PX4 Autopilot on the host computer.



The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

- **Connected IO:** To run this model in the Connected I/O mode, on the **Hardware** tab, in the **Mode** section, select **Connected IO** and then click **Run with IO**.



If the Connected IO firmware is not deployed on the hardware, then the dialog box instructing you to reconnect the flight controller to the serial port appears otherwise the dialog box is not displayed. If the dialog box appears, click **OK** and then reconnect the PX4 Autopilot on the host computer.

3. At each time step, the Read parameter block reads the parameter mentioned in the PX4 Parameter Read block and provides the same as output.
4. Observe the outputs in the **Offset-Scaled Gyro Readings** Display block.
5. To stop running the model on your hardware, click **Stop** in the **Hardware** tab of the Simulink toolbar.

Code Verification and Validation with Processor-in-the-Loop (PIL) Simulation

This example shows you how to use UAV Toolbox Support Package for PX4 Autopilots for code verification and validation with Processor-in-the-Loop (PIL).

In this example, you will learn how to configure a Simulink model to run Processor-in-the-Loop (PIL) simulations. In a PIL simulation, the generated code runs on PX4 flight controllers. The results of the PIL simulation are transferred to Simulink to verify the numerical equivalence of the simulation and the code generation results. The PIL verification process is a crucial part of the development cycle to ensure that the behavior of the deployment code matches the design.

For more details about PIL, refer to this link.

This example introduces the Simulink code generation and verification workflow by showing you how to configure a Simulink model to run PIL simulations on the PX4 flight controllers. This example is pre-configured to run on Pixhawk 6x board. You can configure this model for other supported PX4 flight controllers by selecting the "Hardware board" on the Hardware Implementation pane of the Configuration Parameters dialog box.

Required Hardware

To run this example you will need the following hardware:

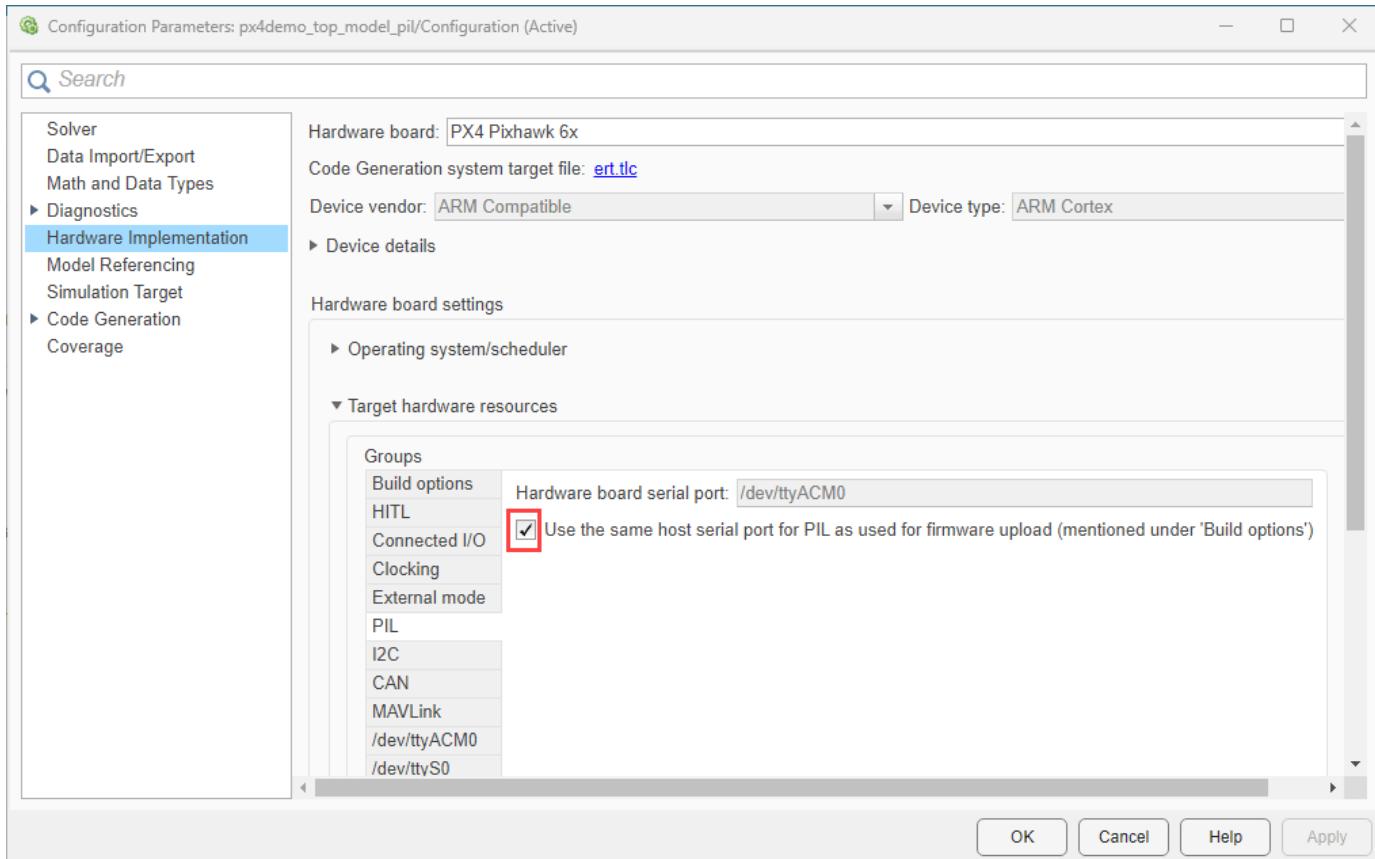
- Supported PX4 flight controller board
- USB type A to Mini-B cable

Choosing a Communication Interface for PIL Simulation

The PX4 flight controller supports interface that does not require any additional cables or hardware besides a USB type A to Mini-B cable that is used to connect the flight controller to the host computer that runs UAV Toolbox Support Package for PX4 Autopilots.

1. Open the px4demo_top_model_pil model.
2. Open the **Modeling** tab and click **Model Settings** to open Configuration Parameters dialog box.
3. Go to **Hardware Implementation > Target Hardware Resources > PIL**.

PIL can be run only on the USB (/dev/ttyACM0) port of the hardware. By default PIL uses the same host serial port used for firmware upload. To manually specify the host serial port, clear the checkbox and specify the port value.



Tip: You can use the Device Manager to identify the COM port of the host computer to which the PX4 flight controller is connected.

Verifying Top Model Code with PIL

This example shows how to verify the generated code for a model by running a PIL simulation. With this approach:

- You can verify code generated for a top model
- You must configure the model to load test vectors or stimulus inputs from the MATLAB workspace
- You can easily switch the entire model between normal and PIL simulation mode

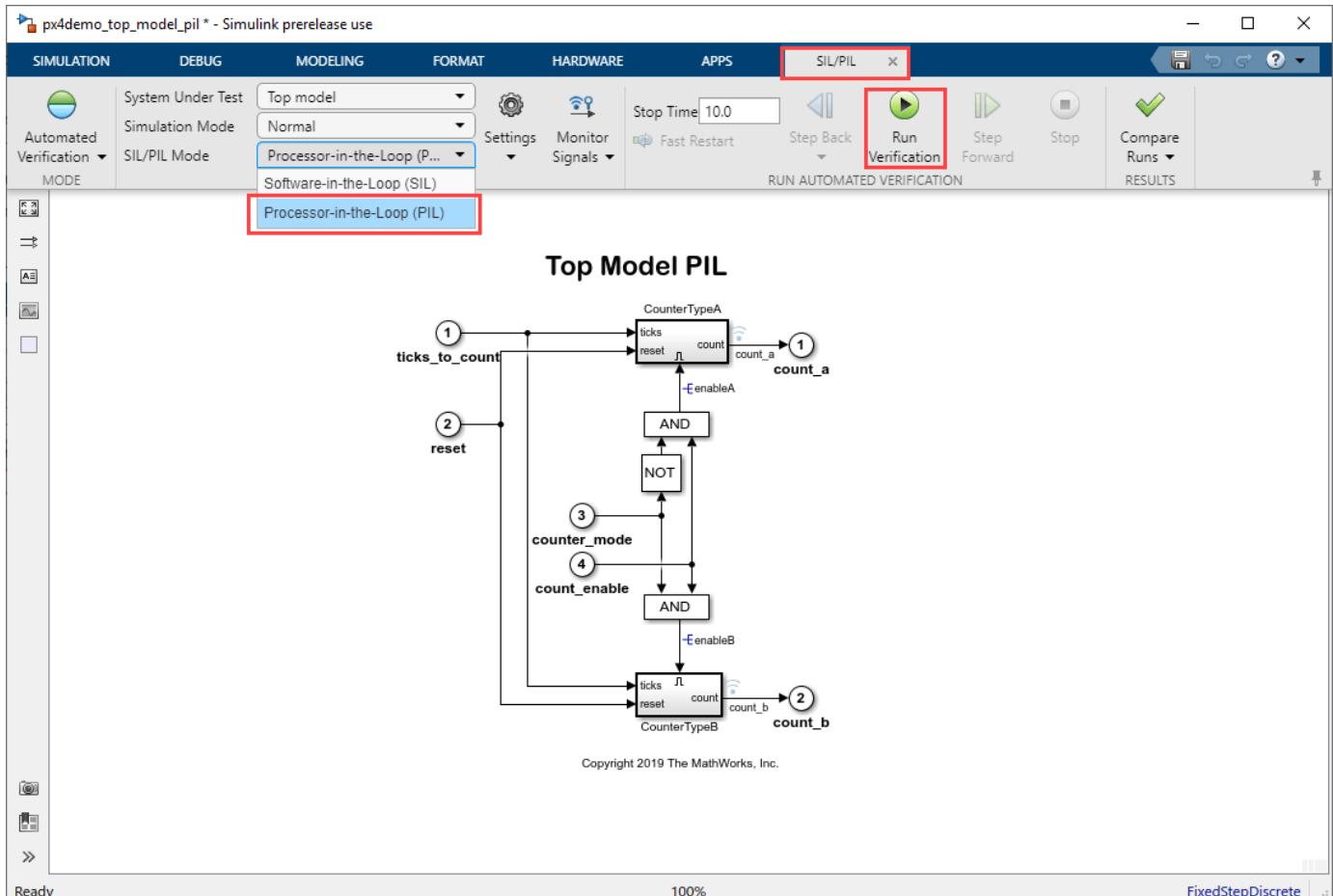
1. Open the px4demo_top_model_pil model.

This model is configured for the **PX4 Pixhawk 6x** target. You can run the model for other PX4 Autopilot targets, by changing the Hardware board to the supported PX4 flight controller in the Configuration Parameters > Hardware Implementation pane.

2. Choose PIL communication interfaces by following the steps in **Choosing a Communication Interface for PIL Simulation** section above.

3. In the Simulink model window, go to **Apps** tab and search **SIL/PIL Manager**.

4. Select **Processor-in-the-Loop(PIL)** from the **SIL/PIL Mode** drop-down and click **Run Verification**.



5. When the PIL simulation is completed, a **logsOut** variable is created in the base workspace. The **logsOut** data contains PIL simulation results. You can access the logged data for signals **count_a** and **count_b** by using the following commands:

a. `count_a = get(logsOut,'count_a');`

`count_a.Values.Data`

b. `count_b = get(logsOut,'count_b');`

`count_b.Values.Data`

Verifying Referenced Model Code with PIL

This example shows how to verify the generated code for a referenced model by running a PIL simulation. With this approach:

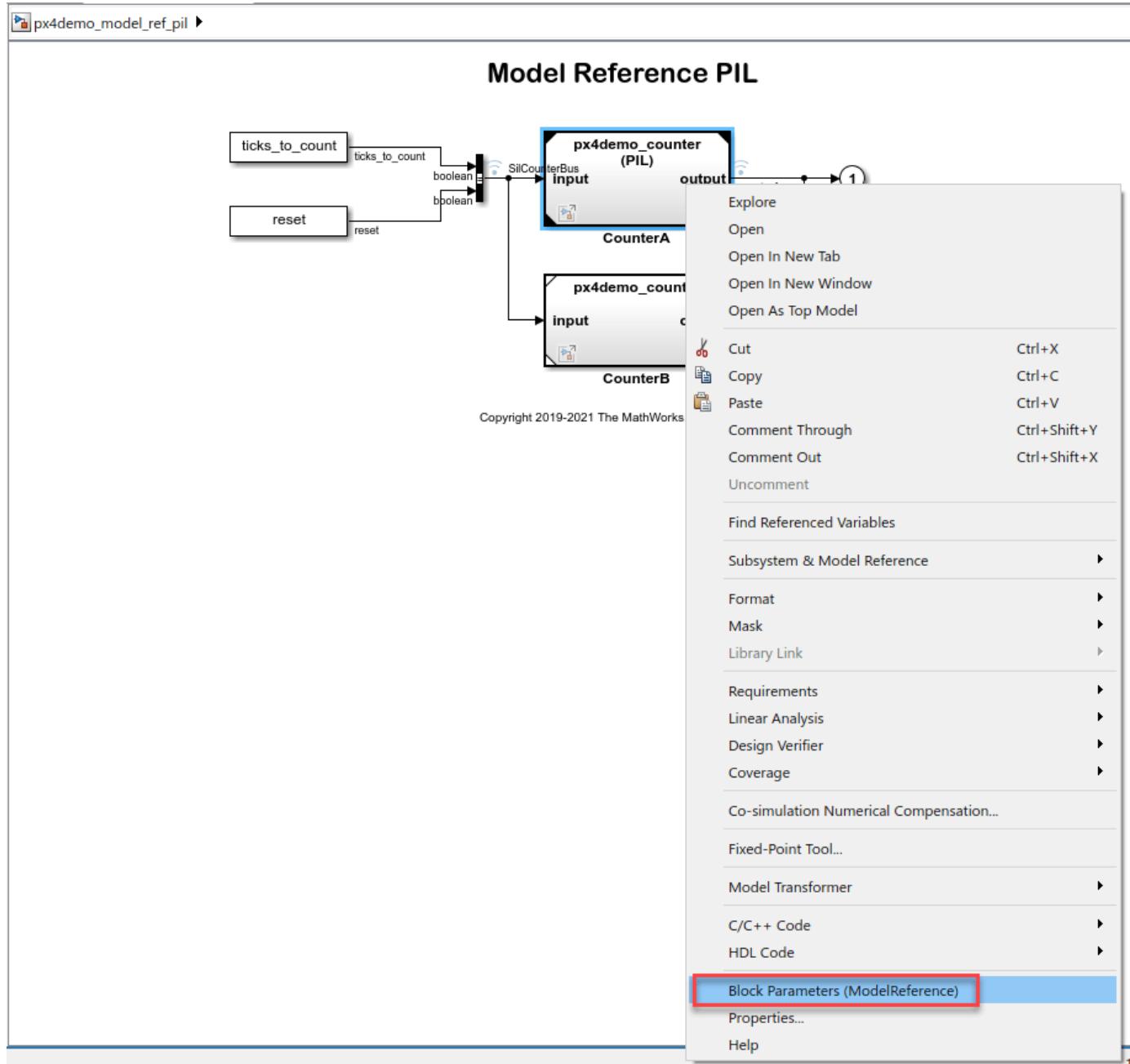
- You can verify code generated for referenced models
- You must provide a test harness model to provide a test vector or stimulus inputs
- You can easily switch a Model block between normal and PIL simulation mode

1. Open the px4demo_model_ref_pil model.

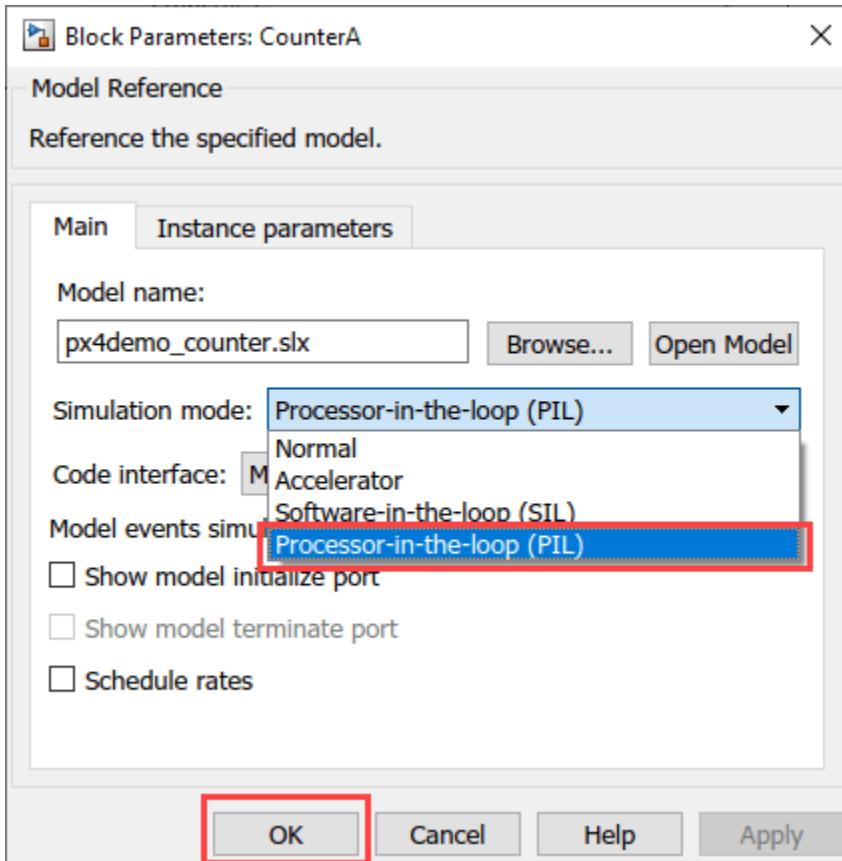
This model is configured for **PX4 Pixhawk 6x** target. You can run the model for other PX4 targets, by changing the Hardware board to supported PX4 flight controllers in the Configuration Parameters > Hardware Implementation pane.

The model contains two Model blocks (CounterA and CounterB) that both point at the same referenced model. Note that the Hardware board change has to be made in the reference model also (right-click CounterA or CounterB block, and select **Open as Top Model**), by following the steps above. You will configure one of the Model blocks to run in PIL simulation mode and the other in normal mode.

2. Choose a PIL communication interface by following the steps in **Choosing a Communication Interface for PIL Simulation** section above.
3. Configure and run **CounterA** Model block in PIL simulation mode by following steps below:
 - a. Right click on block **CounterA** and select **Block Parameters (ModelReference)**



- b. In the **CounterA** block parameter, select **Simulation mode** as **Processor-in-the-Loop(PIL)**, and click **OK**.



c. Go to **Simulation** tab and click **Run**.

4. When the model starts running, **Scope1** displays the PIL simulation output running on the PX4 Autopilot while **Scope2** shows the normal mode simulation output.

Verifying the Generated Code for a Subsystem Using a PIL Block

This example shows how to use a PIL block for subsystem code verification. With this approach:

- You can verify the code generated for a subsystem
- You must provide a test harness model to supply a test vector or stimulus inputs
- You must swap your original subsystem with a generated PIL block; you should be careful to avoid saving your model in this state as you would lose your original subsystem.

1. Open the px4demo_pil_block model.

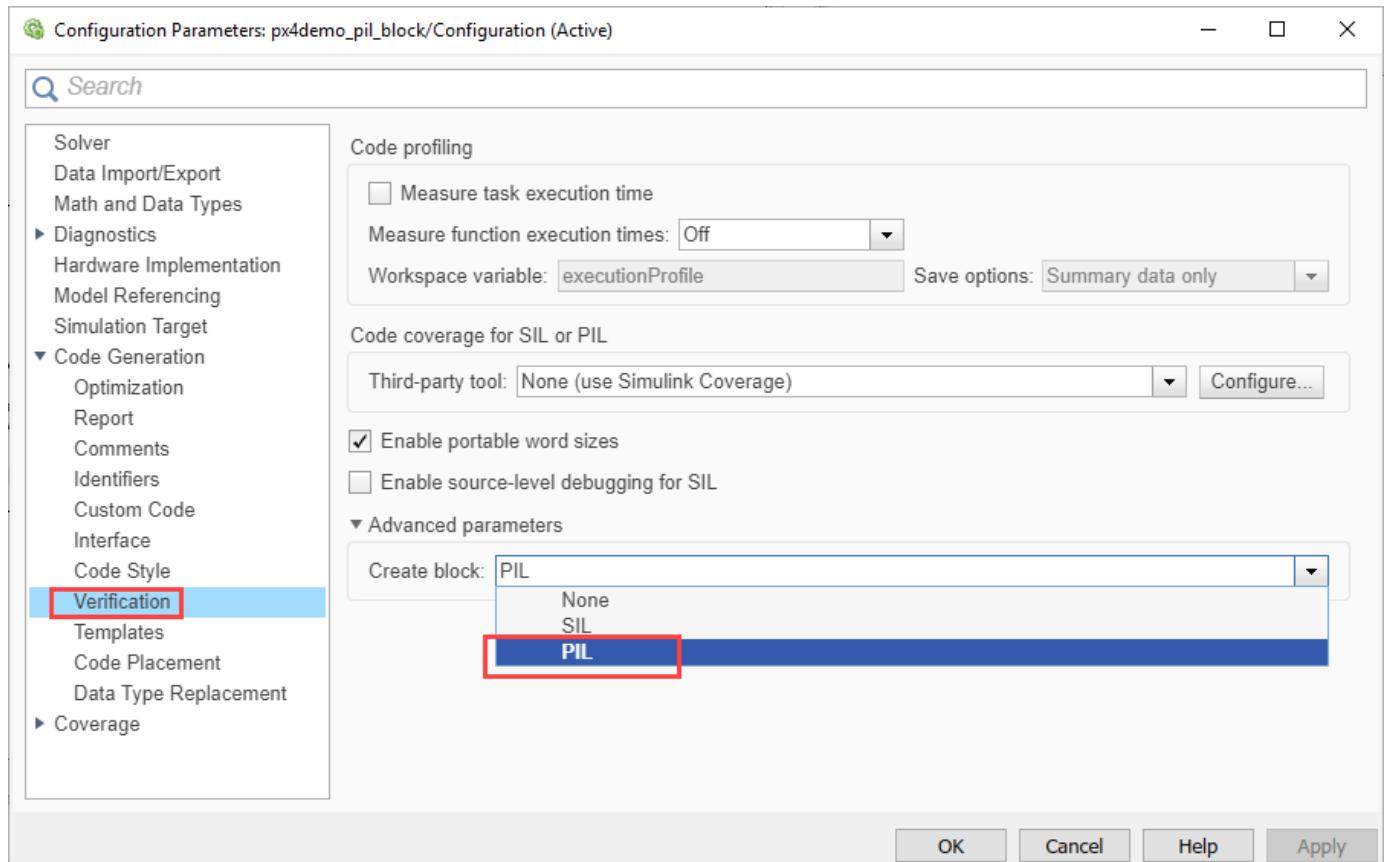
This model is configured for the **PX4 Pixhawk 6x** target. You can run the model for other PX4 targets, by changing the Hardware board to supported PX4 flight controller in the Configuration Parameters > Hardware Implementation pane.

The objective here is to create a PIL block out of the **Controller** subsystem that you will run on the PX4 flight controller.

2. Choose a PIL communication interface by following the steps in **Choosing a Communication Interface for PIL Simulation** section above.

3. Enable PIL by following these steps:

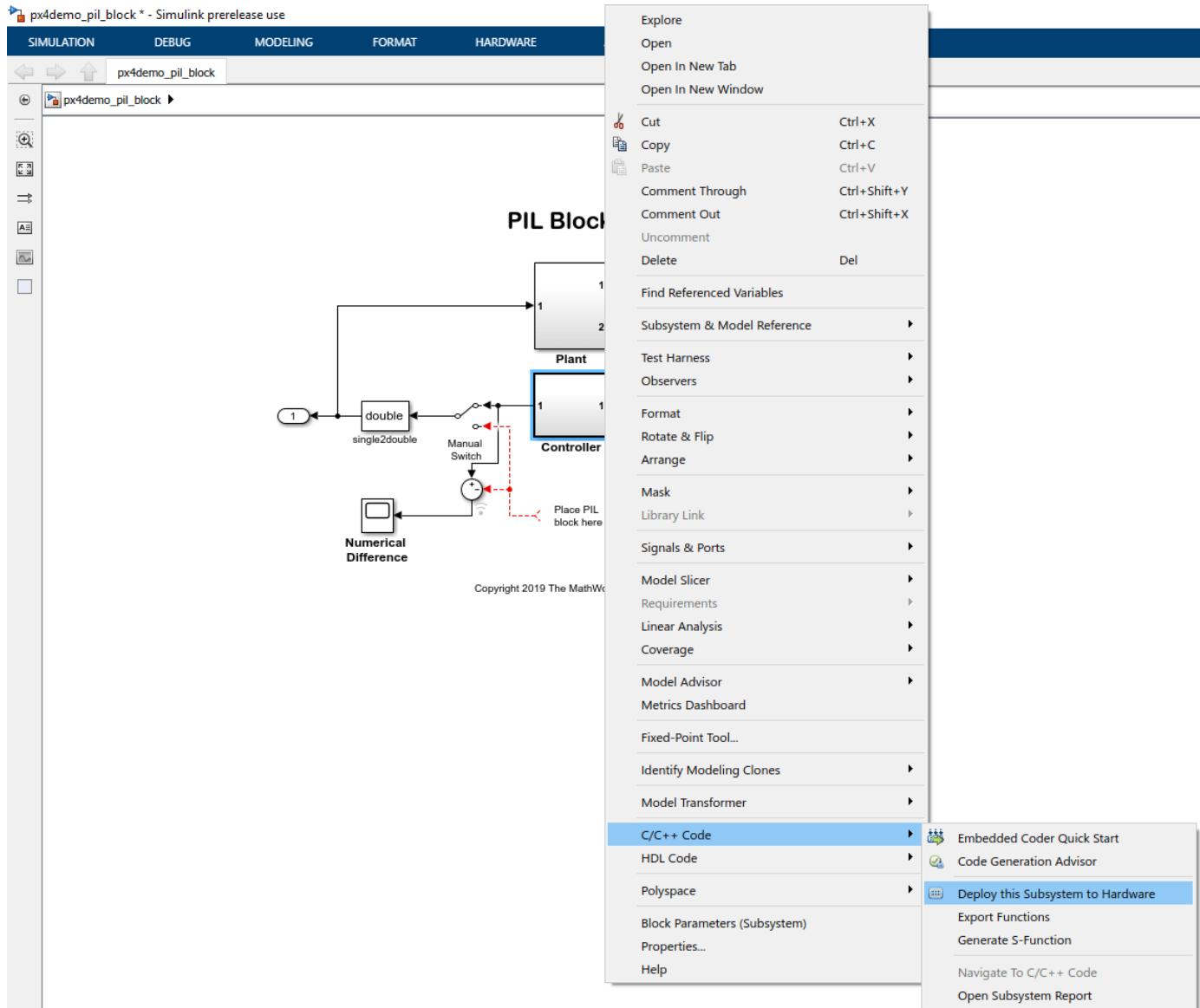
- Go to the **Modeling** tab and click **Model Settings** to open Configuration Parameters dialog box.
- Go to **Code Generation > Verification > Advanced parameters** and select **PIL**.



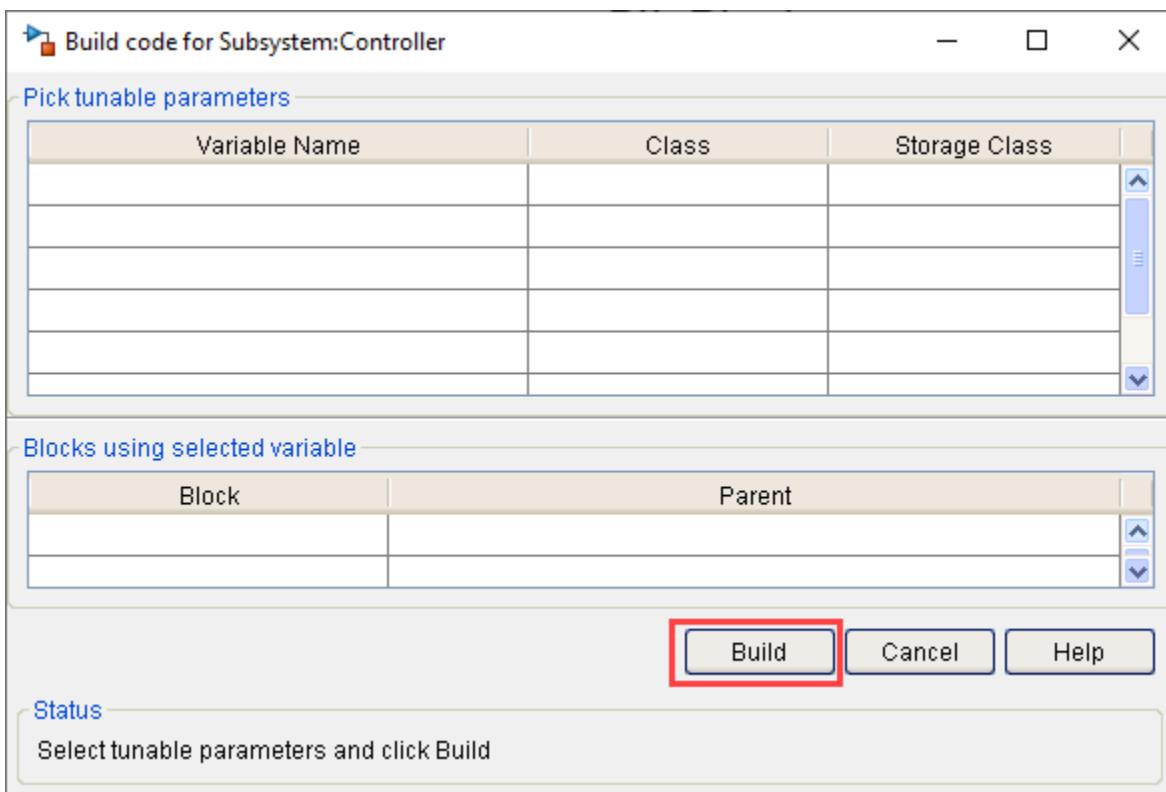
4. Create a PIL block for the **Controller** subsystem by following these steps:

- Right click on the **Controller** subsystem and select **Deploy this Subsystem to Hardware**.

1 Blocks

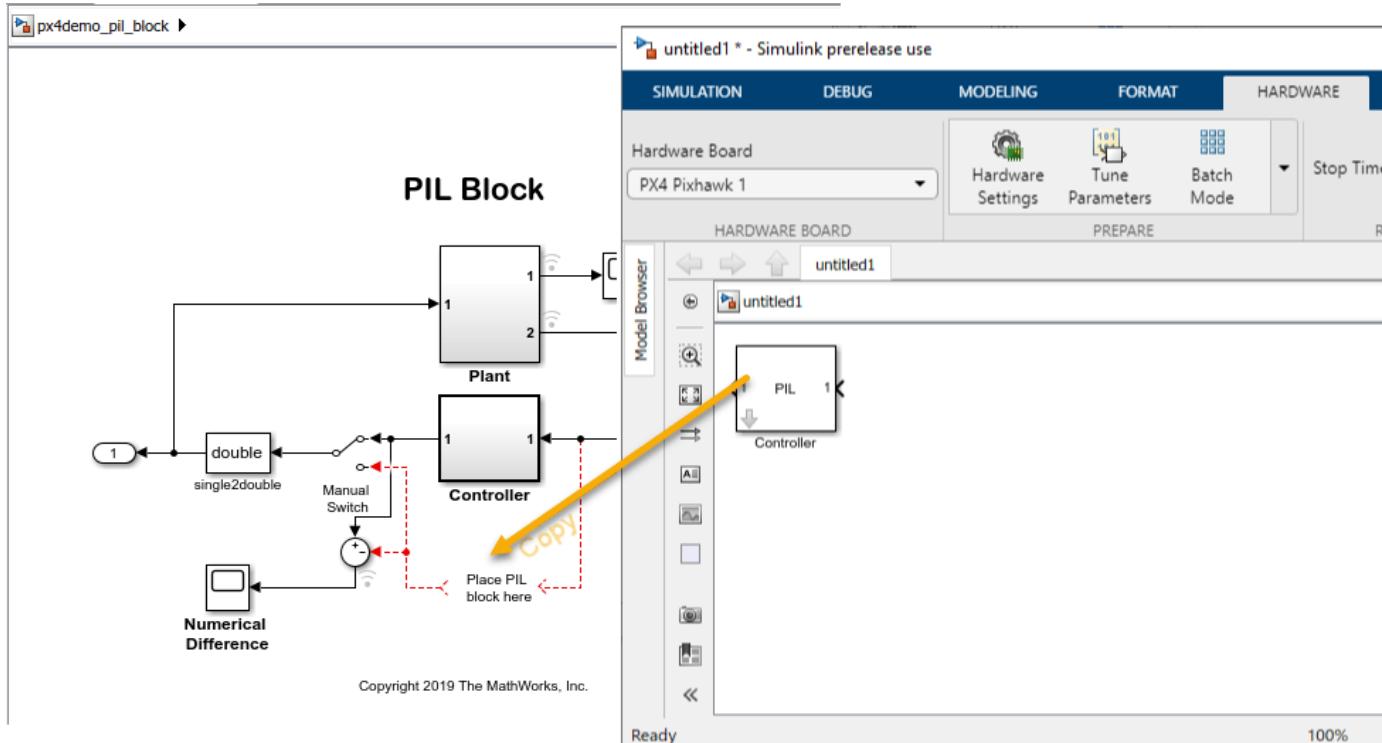


- b. In the **Build code for SubsystemController** dialog box, click **Build**.



5. Run PIL simulation by following below steps:

- Copy the PIL Subsystem block to your model.
- Go to the **Simulation** tab, and click **Run**.



6. The generated executable is copied to the board.

7. You can switch between the original and PIL block subsystems by double clicking on the **Manual Switch** block. Double click on the **Numerical Differences** block to see the difference between the simulated **Controller** subsystem and the PIL block running on the Pixhawk Series controller hardware board.

Perform Code Profiling with PIL

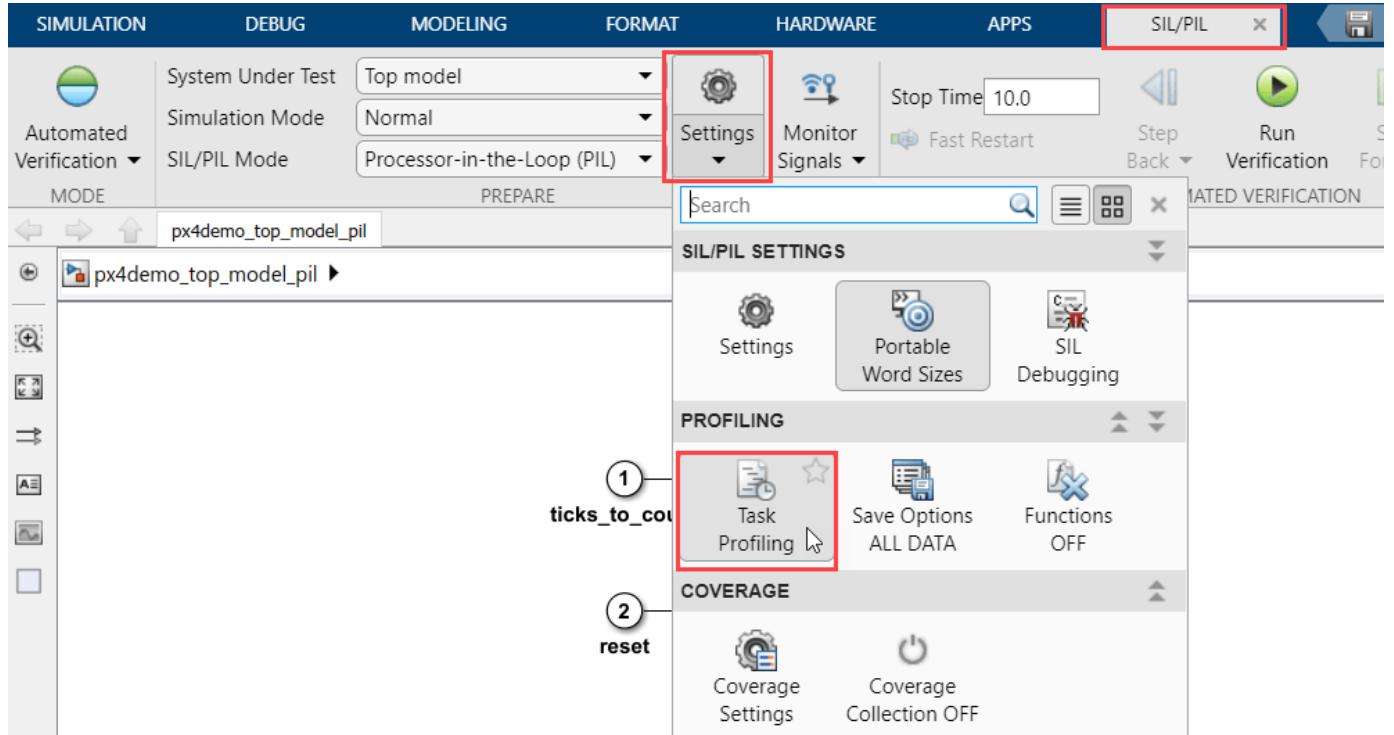
You can perform code profiling while verifying code using PIL. Code profiling helps you to check whether the generated code meets real-time performance requirements. You can identify the tasks that require the most time and then investigate whether a trade-off between functionality and speed is possible.

You can enable code profiling along with PIL simulation by using SIL/PIL Manager App (which enables basic code profiling) or by using the Configuration Parameters dialog box (which provides advanced options for code profiling).

Note: The execution time profiling depends on the clocking frequency that is configured for the selected hardware board. In the Configuration Parameters dialog box, go to **Hardware Implementation > Target hardware resources > Clocking** to view the clocking frequency.

Enable Code Profiling from SIL/PIL Manager App

1. After you open the Simulink model, go to **Apps** tab and select **SIL/PIL Manager**.
2. In the SIL/PIL tab, select **Processor-in-the-Loop(PIL)** from the SIL/PIL Mode drop-down list.
3. Go to **Settings** and select **Task Profiling**. Selecting this option enables the execution time profiling of the model.



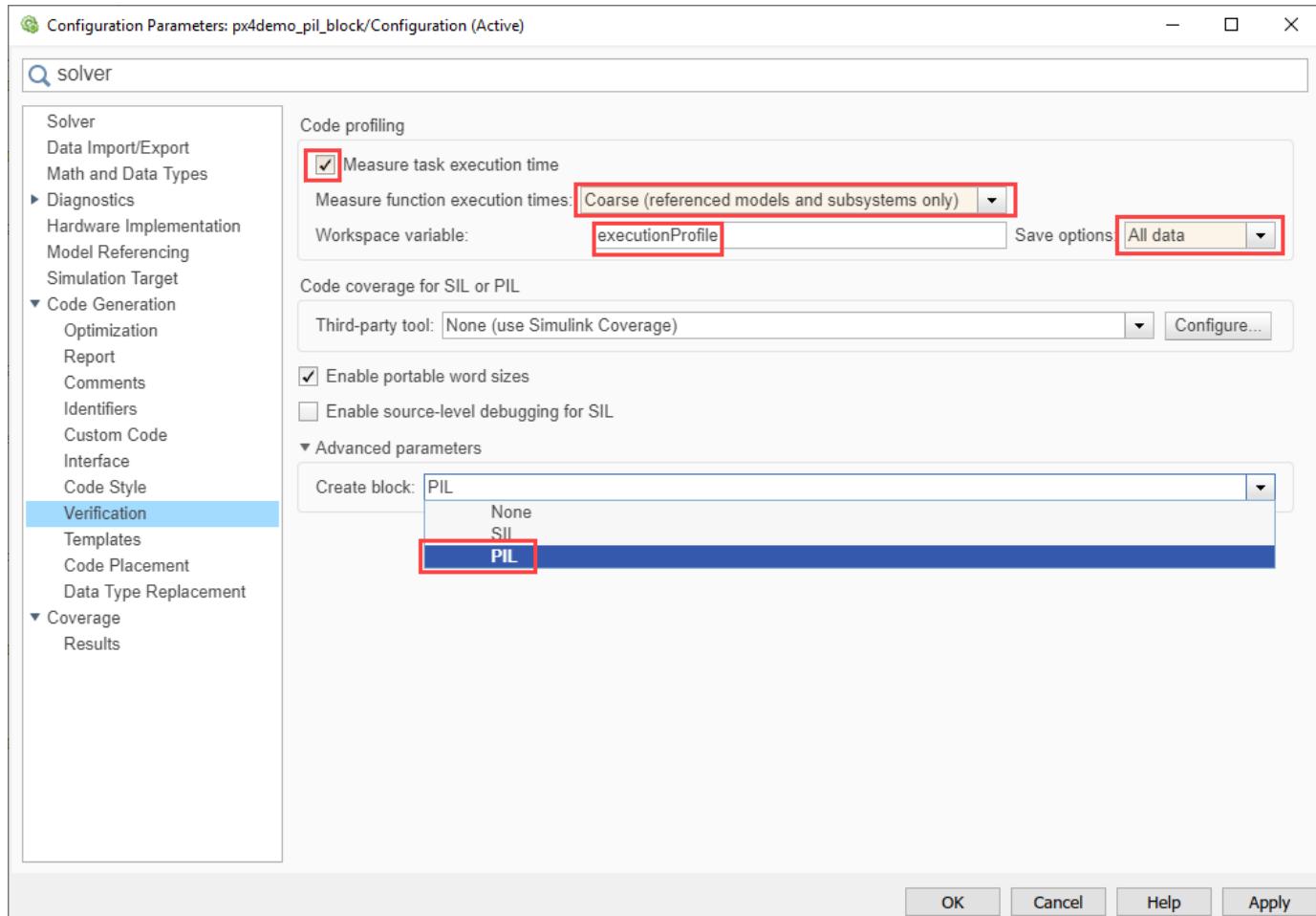
Enable Code Profiling from Configuration Parameters Dialog Box

1. After you open the Simulink model, go to the **Modeling** tab and click **Model Settings** to open Configuration Parameters dialog box.

2 Go to **Code Generation > Verification > Advanced parameters** and select **PIL**.

3 To profile the execution time for each rate in the model, select **Measure task execution time**.

The next parameter **Measure function execution times** is set to *Off*, by default. You can set this parameter to *Coarse (referenced models and subsystems only)* or to *Detailed (all function call sites)*. Additionally, set the **Save options** to *All data*. Verify that the name of the Workspace variable is **executionProfile**. Click **Apply** and then **OK**.



Complete the PIL simulation by following the required steps (as mentioned in the previous sections).

After the PIL simulation is completed, the `executionProfile` variable appears in the MATLAB workspace. Obtain the profiling report and analyze different turnaround and execution times:

```
report(executionProfile)
```

Attitude Control for X-Configuration Quadcopter Using External Input

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to design an attitude controller for an X-configuration quadcopter that uses input from a joystick or radio control transmitter. In this example, you also verify the controller design using PX4 Host Target and SIH in Host Target simulator.

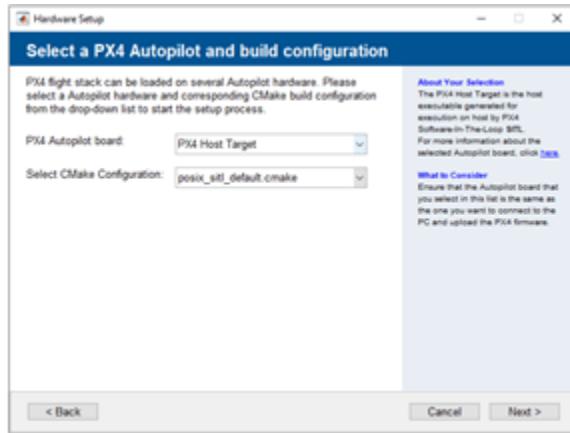
Introduction

UAV Toolbox Support Package for PX4 Autopilots enables you use Simulink® to design flight controller algorithm to stabilize the vehicle based on the current vehicle attitude and track the desired attitude using Simulink.

In this example, you will learn how to use the PX4 Host Target with SIH simulator and jMAVSIM visualizer to design as well as verify the attitude controller for quadrotor vehicle. You will also learn how to control the vehicle using joystick or radio control transmitter. The jMAVSIM simulator which is part of the Software In The Loop (SITL) simulation as defined in PX4 website, is installed as part of the support package installation.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial setup and configuration tasks of the support package using Hardware Setup screens. In the Hardware Setup screen **Select a PX4 Autopilot and build configuration**, select PX4 Host Target as the Hardware board from the drop-down list.



- Install the ground control station software - QGroundControl.

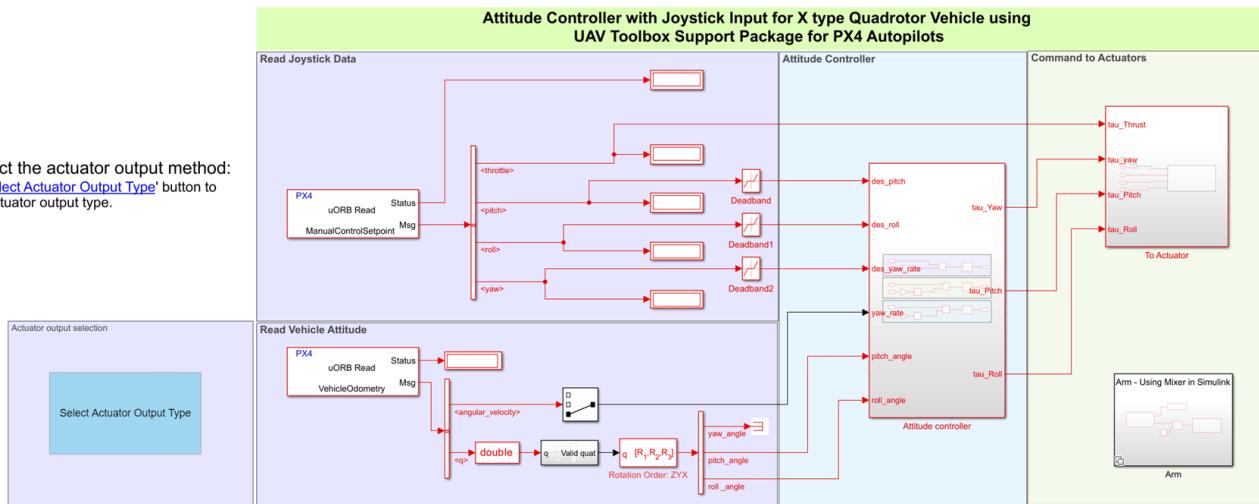
Required Hardware

To run this example, you will need the following hardware:

- Either a Joystick (for example, Logitech F310 Gamepad) or a Radio control transmitter (for example, FlySky FS-i6)

Model

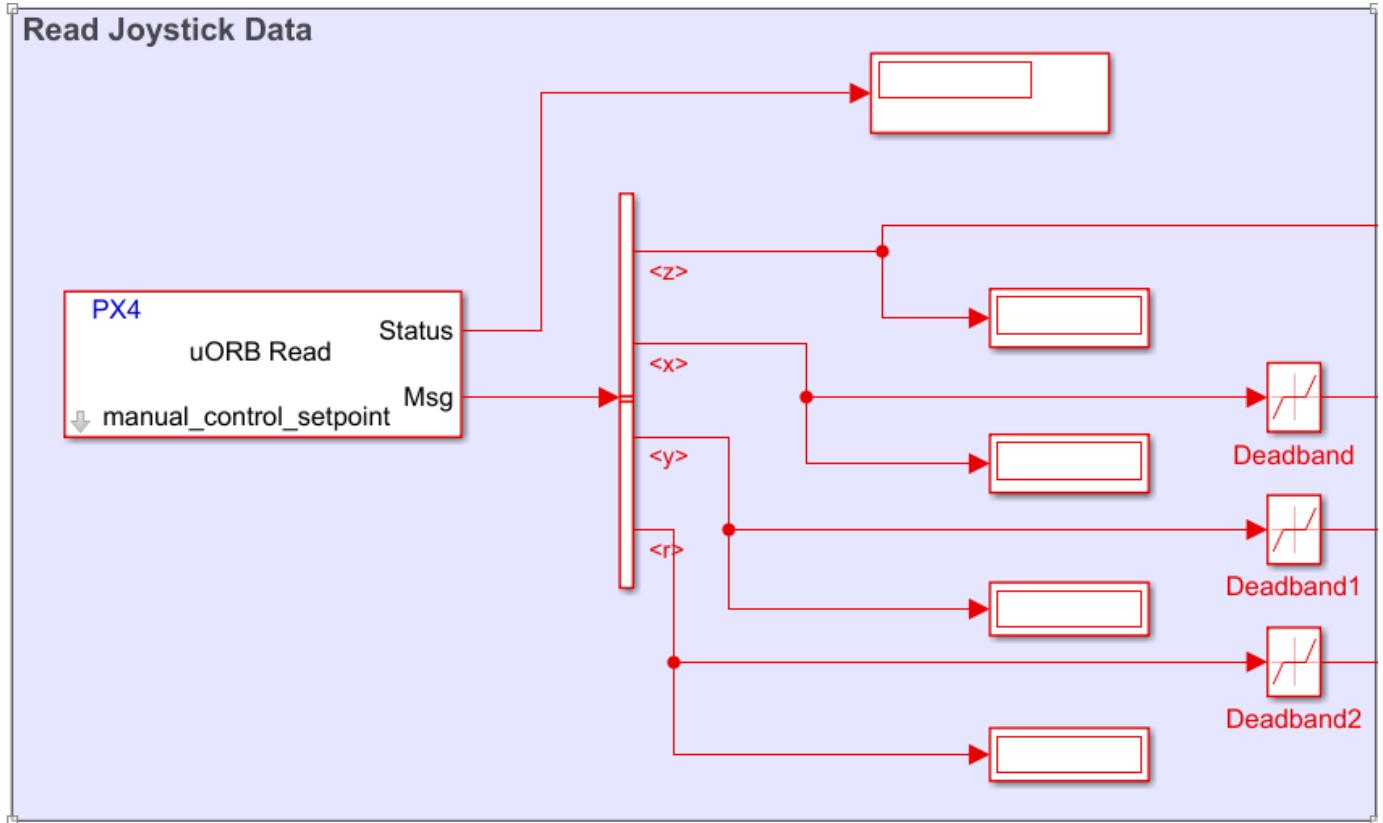
Open the px4demo_AttitudeControllerJoystick model.



The model implements a Proportional-Integral-Derivative (PID) controller to control the attitude of an X type quadrotor aerial vehicle. At each time step, the algorithm adjusts rotational speeds of different rotors to track the desired attitude, based on input from the joystick or radio control transmitter.

Task 1 - Configure the Model to Read Input from Joystick or Radio Control Transmitter

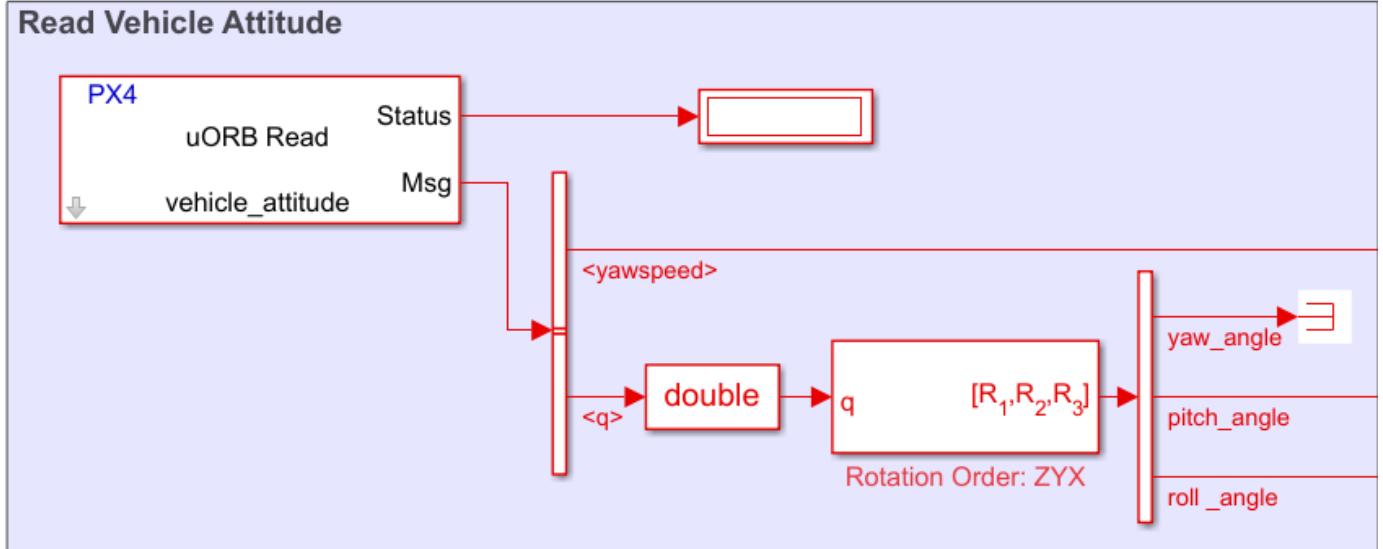
In this task, you configure the model to obtain the stick position of the joystick or radio control transmitter. This is implemented by using the uORB Read block, which is configured to read the message `manual_control_setpoint`.



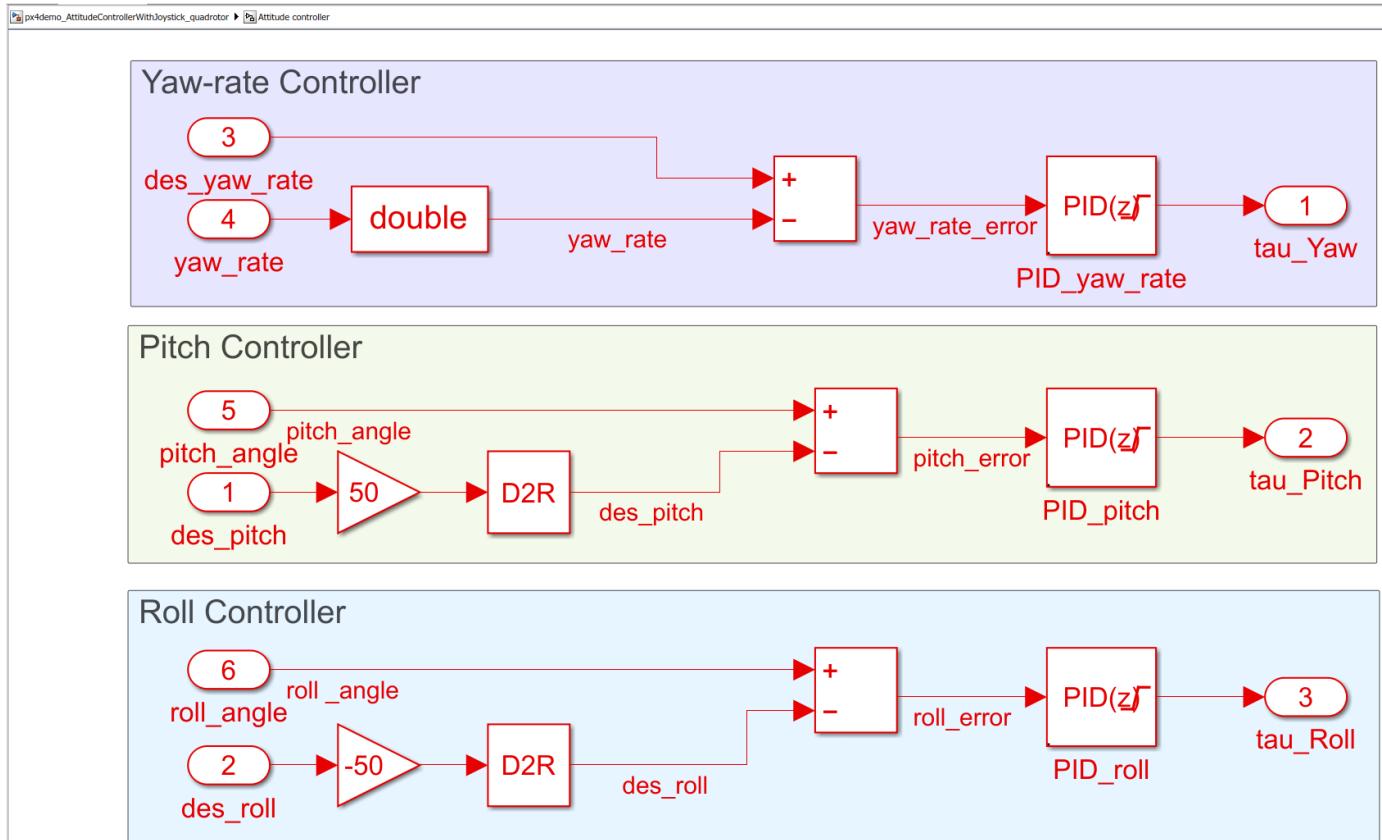
Note: The joystick data in uORB message `manual_control_setpoint` is only available after QGroundControl is running and connected to PX4 Host Target.

Task 2 - Read Vehicle Attitude and Design Attitude Controller Using PID 1Dof

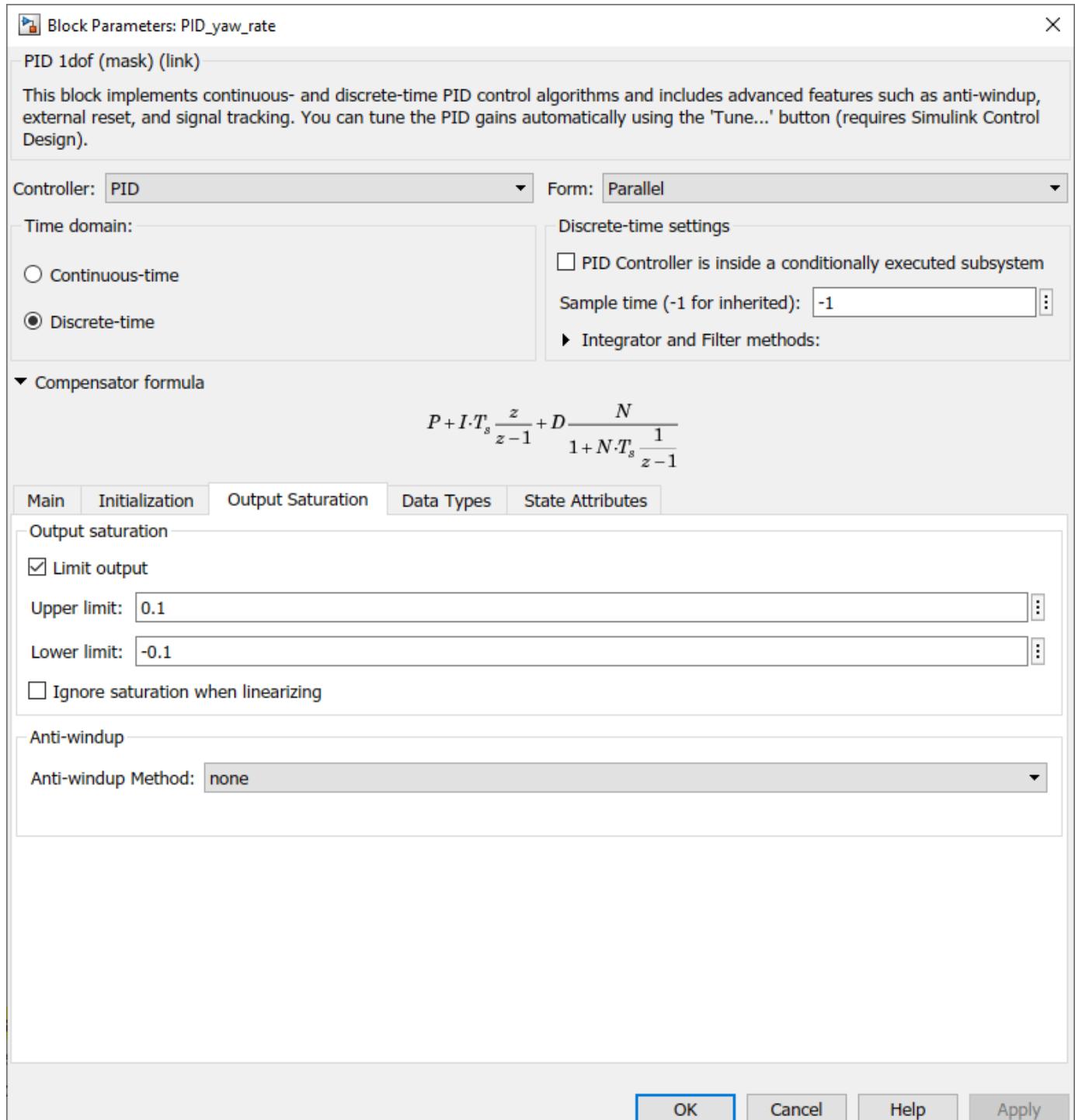
In this task, you configure the model to read the vehicle attitude and obtain the required values (pitch, roll, and yaw) and design an attitude controller using the PID 1Dof block in Simulink.



- Use a uORB Read block, which is configured to read the message `vehicle_attitude`, to access the vehicle's current attitude.
- Use the Quaternions to Rotation Angles block to convert attitude quaternion to various Euler rotation angles (pitch, roll, and yaw) according to selected rotation order.
- In the *Attitude Controller* subsystem, use three PID 1Dof blocks to generate the required angular rates from the difference between the desired value (as read from the joystick) and the current attitude value.

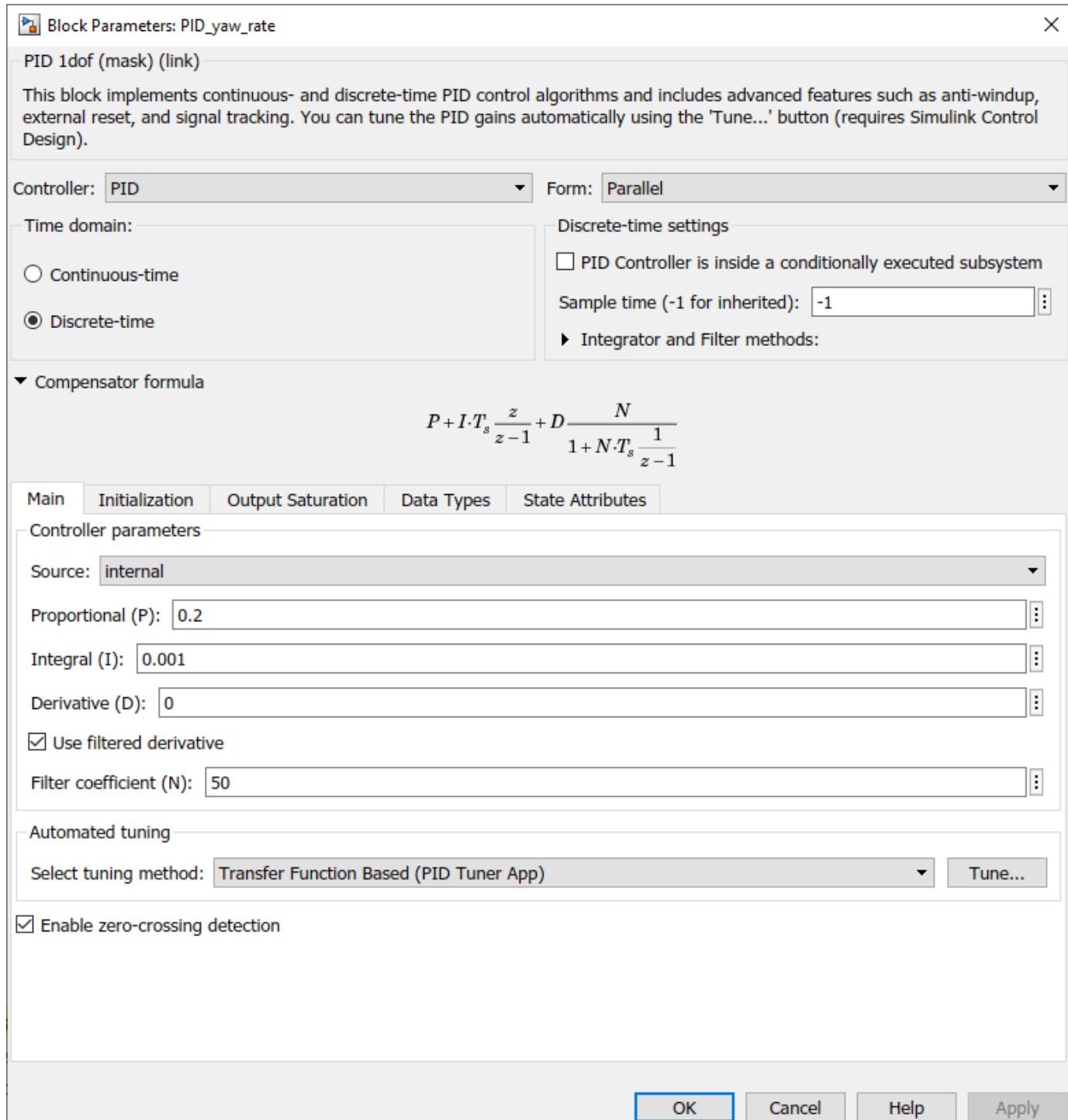


The output of each PID 1Dof block is restricted between predefined maximum and minimum values to limit the rate at which rotation angles change.



- Each PID 1Dof block uses filtered derivatives to eliminate the amplification of any undesired noise signal due to signal differentiation.

This figure shows the settings in the Main tab of the PID 1Dof block mask.



Task 3 - Combine Angular Rate Demands to Generate Rotational Speed of Individual Rotor

In this task, you configure the model to generate the individual rotational speed of each of the four rotors from current angular rate demands and also based on the total thrust requirement from the throttle stick.

The input from the throttle stick of the joystick or radio control transmitter is the base value of the rotational speed of all the rotors. According to the angular rate demands, a small fraction is added to or subtracted from the base value.

For this example, the motor configuration is as shown in the following figure.

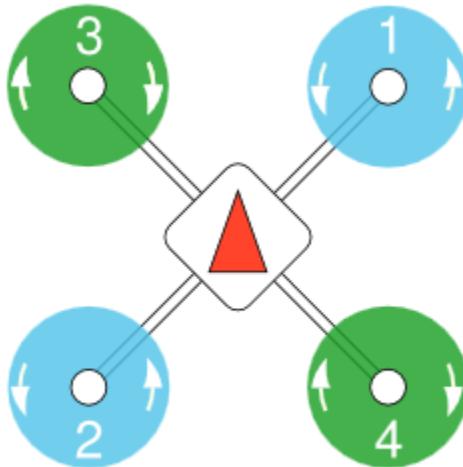
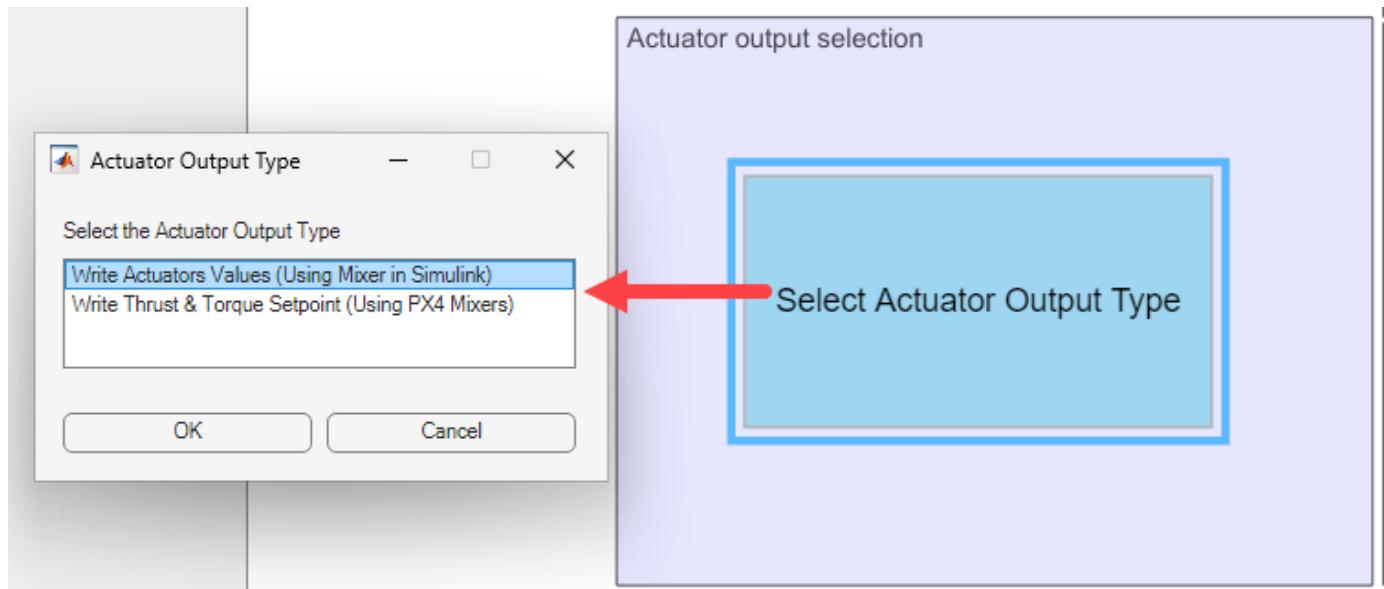


Image courtesy https://dev.px4.io/v1.9.0/en/airframes/airframe_reference.html

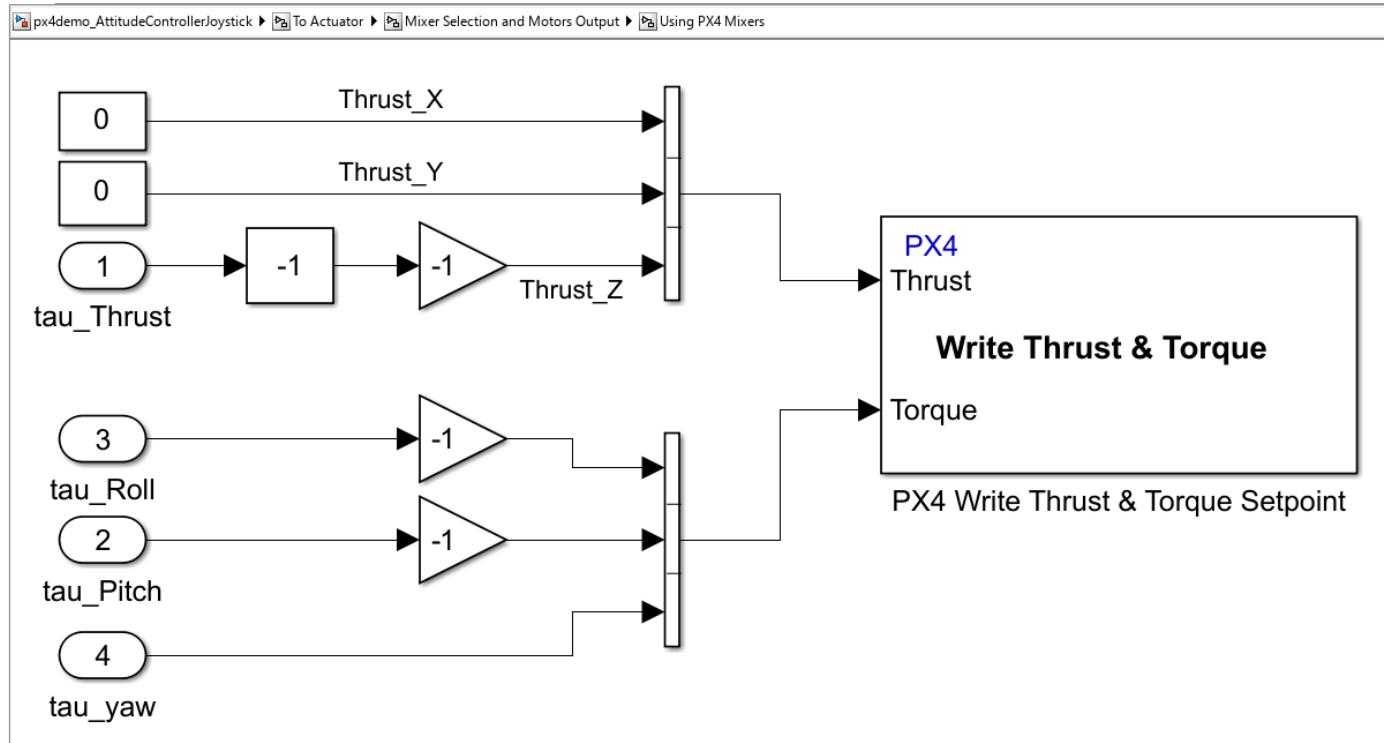
Select Actuator Output Type

In the Simulink model, click **Select Actuator Output Type** and then select the required option to use PX4 mixers or Simulink mixers from the Actuator Output Type dialog box.



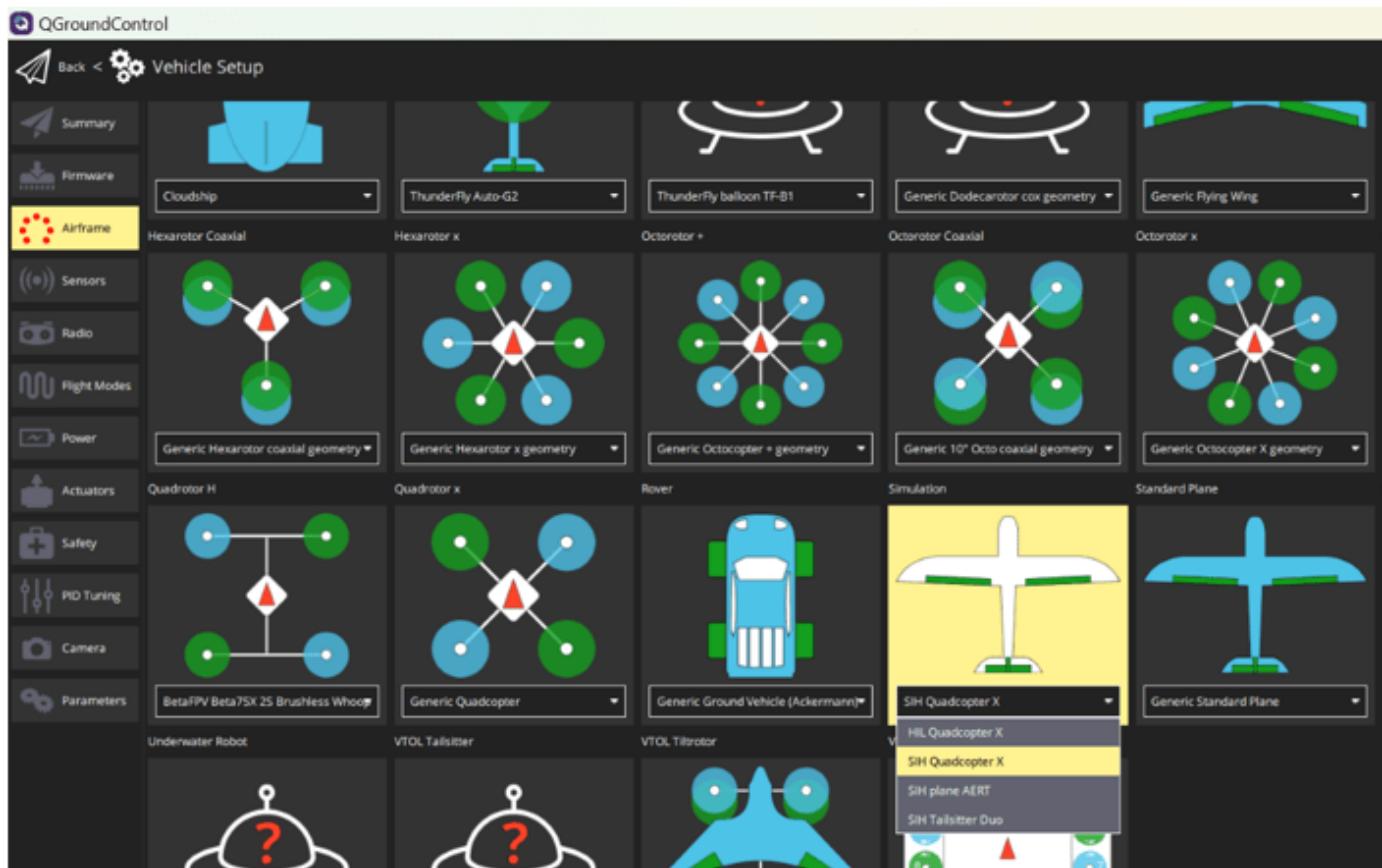
- **Using PX4 Mixers**

If you select Write Thrust & Torque Setpoint (Using PX4 Mixers) option, then the *Using PX4 Mixers* subsystem contains a PX4 Write Thrust & Torque Setpoint block and the airframe configurations will be defined in QGC and used by PX4 for mixing.

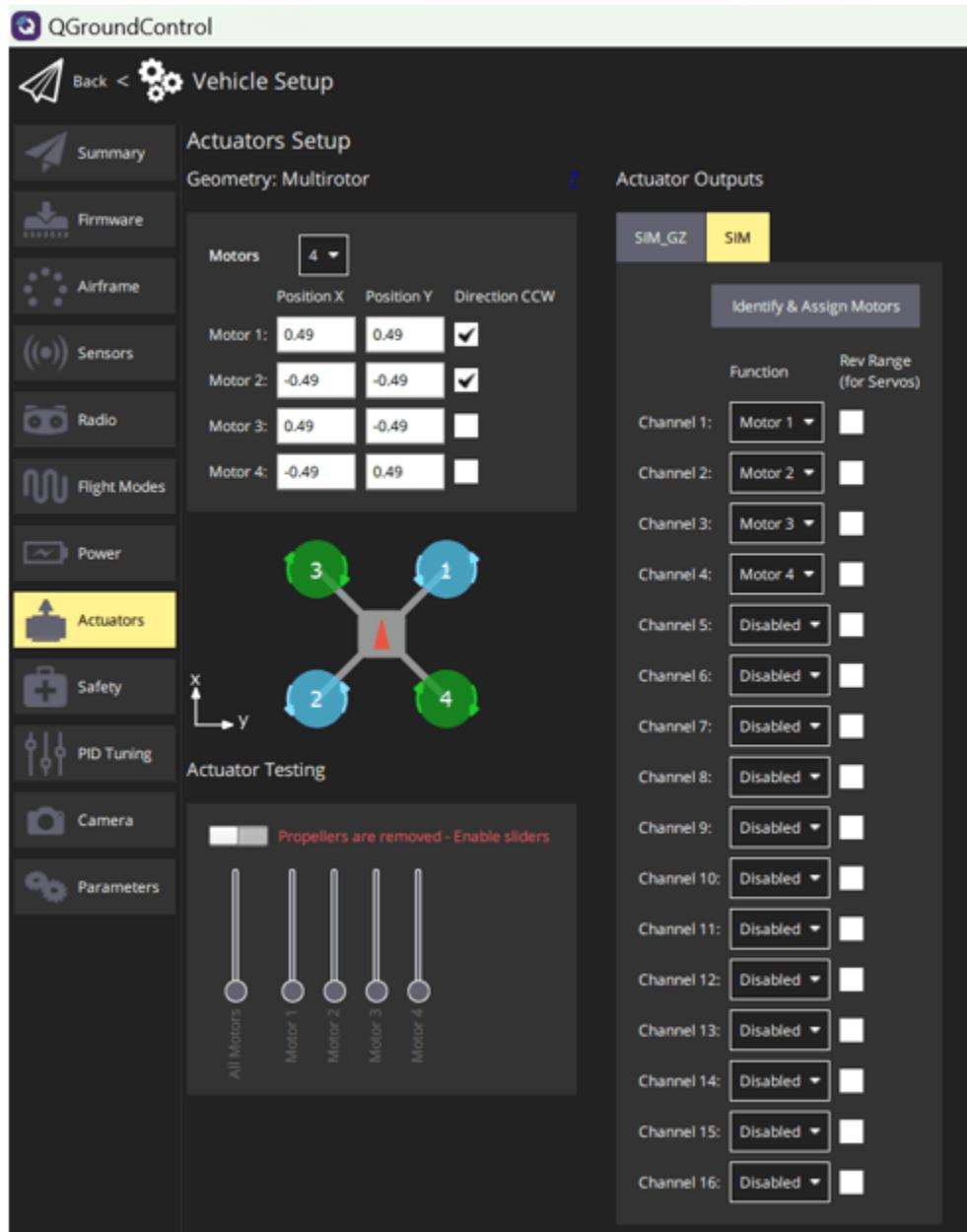


To use PX4 mixers as the actuator output type, you must configure these settings in QGC.

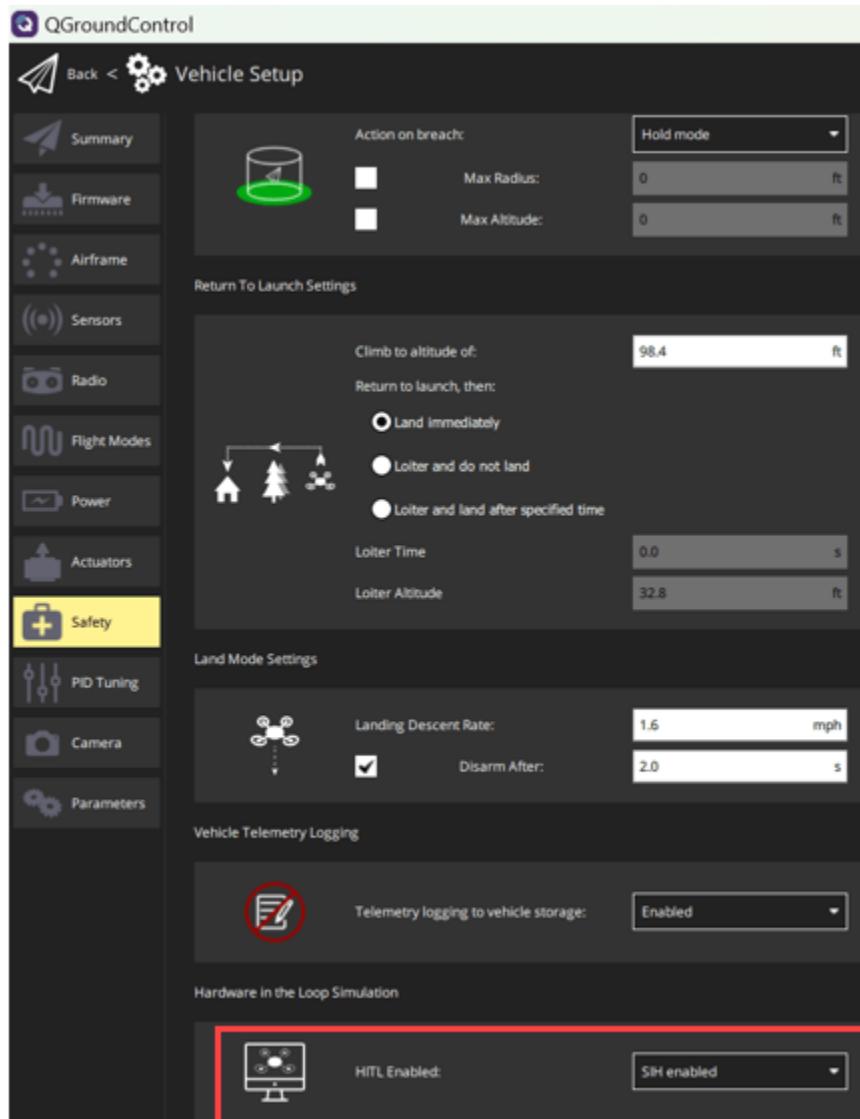
1. In the QGC, select Airframe.



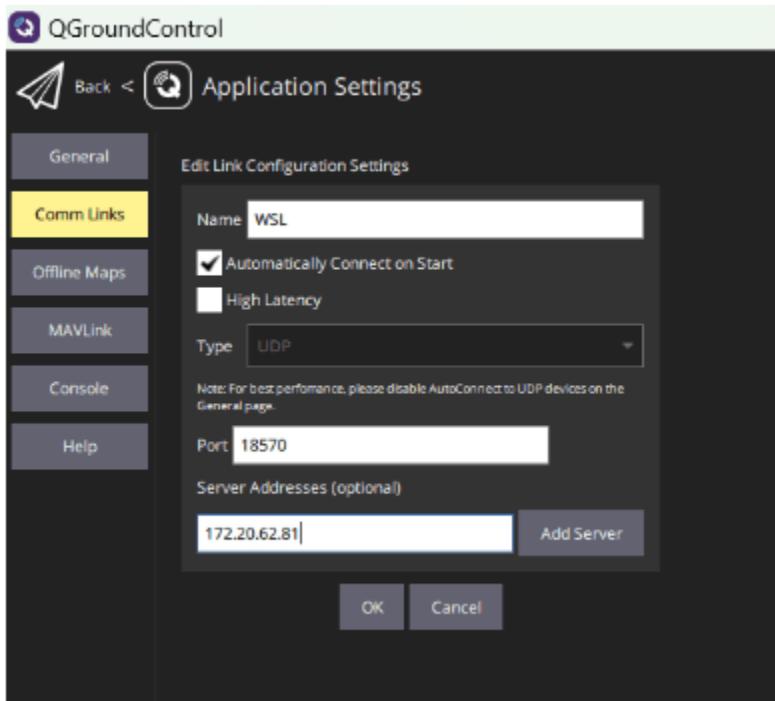
- Navigate to **Setup > Airframes**.
 - Select **SIH Quadcopter X**. Click **Apply and Restart** on top-right of the **Airframe Setup** page.
 - Restart QGC to apply the changes.
2. Click **Actuators** and select the required actuators.



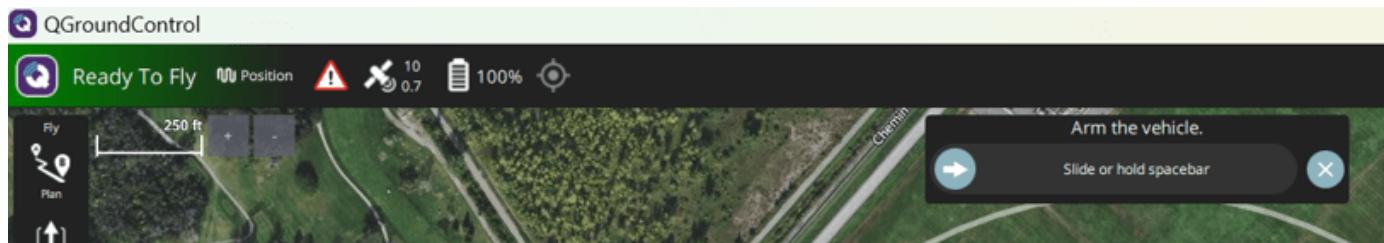
3. Navigate to **Setup > Safety** section.



4. Click **Comm Links** and configure the communication links. For more information see “Configuring QGC for Vehicle Visualization Without a Display”.



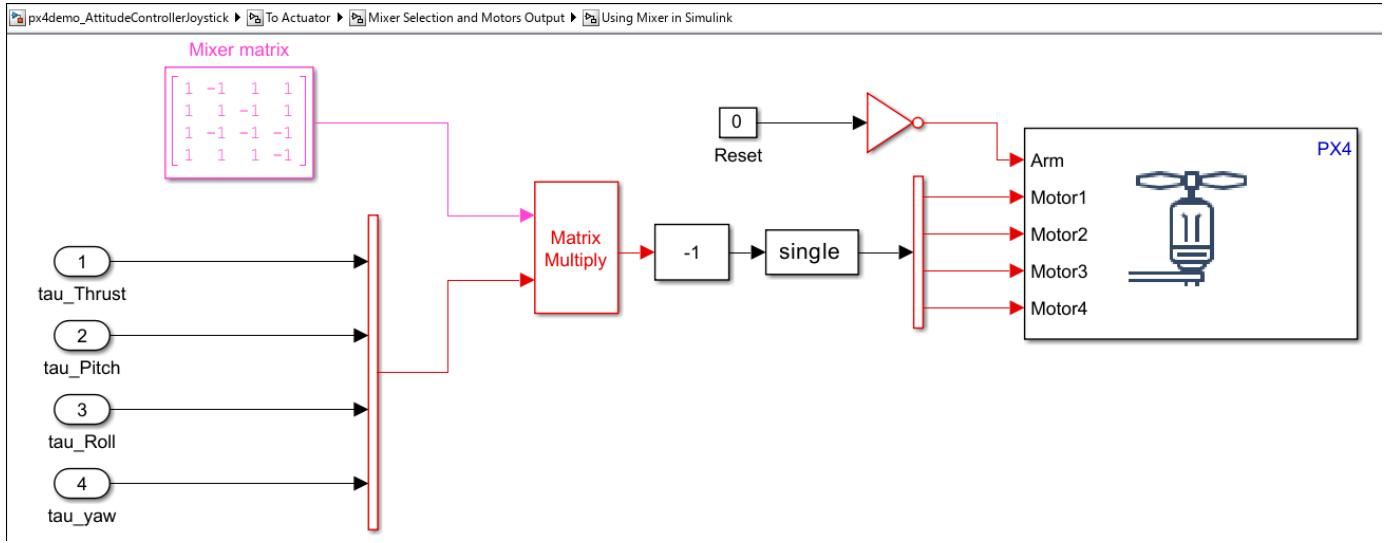
5. Arm the vehicle by moving the slider.



Select **SIH enabled** from the **HITL Enabled** list.

- **Using Mixer in Simulink**

The *Using Mixer in Simulink* subsystem in the model contains a Mixer matrix block, which is defined in Simulink. The structure of the mixer matrix depends on the arrangement of motors with respect to the frame of reference.



The PX4 Actuator Write block takes the individual rotational speed values and sends it to a particular actuator. Along with rotational speeds, the block also has input field for Arm.

Task 4 - Run the Model on PX4 Host Target and Start jMAVSim Simulator

In this task, you select the Hardware Board as PX4 Host Target and start the jMAVSim simulator.

1. In the **Modeling** tab, click **Model Settings**.
2. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane.
 - Set the **Hardware board** to *PX4 Host Target* (the same option that you selected during Hardware Setup screens).
 - In the **Target Hardware Resources** section, set the **Build options** to **Build, load and run**.
3. Once Simulink model is ready, there are two options to build the Simulink model and launch the simulator:
 - On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. Once the model is deployed in the target hardware (PX4 Host Target in this example), the jMAVSim simulator is launched and the model executes independently in target hardware without having any dependencies and without communication with Simulink.
 - On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune**. The jMAVSim simulator is launched and you can monitor and log the data while the model executes in the target hardware.

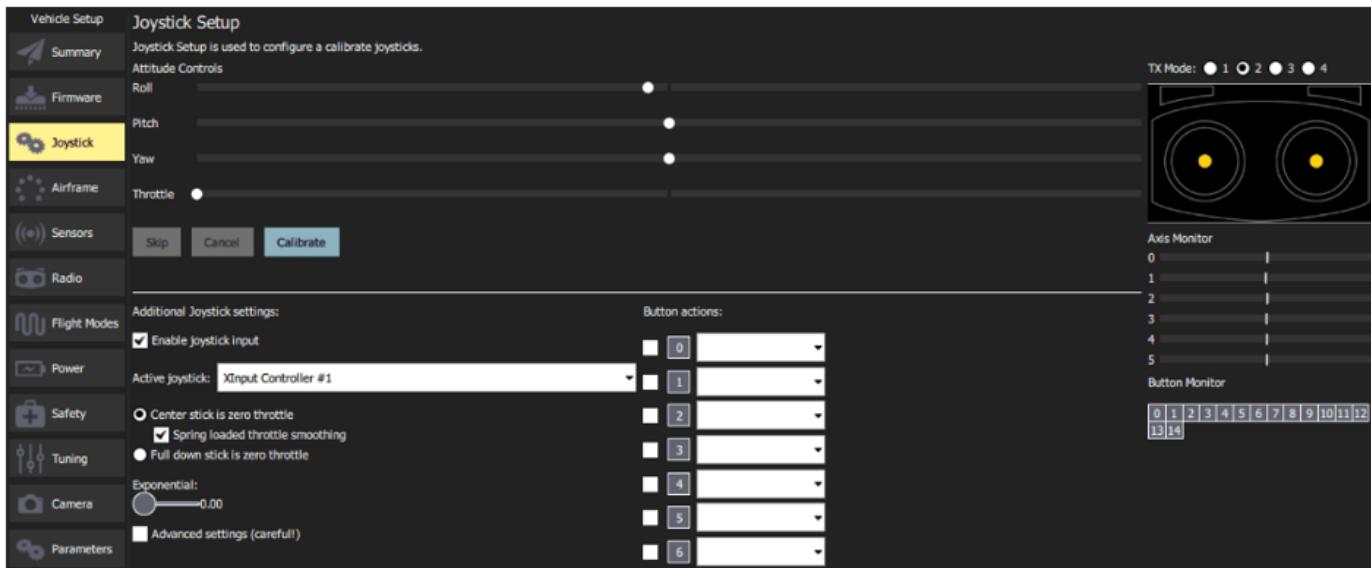
During Monitor and Tune operation, various parameters and variables can be monitored using display or scope blocks.

Task 5 - Connecting Joystick or Radio Control Transmitter to Host Computer and Configuring QGroundControl

While performing simulation using the PX4 Host Target and the jMAVSim simulator, you can connect external input devices to QGroundControl, which transmits commands (joystick movement and button presses) to PX4 Autopilot using MAVLink.

Connecting Joystick

- While the model is running on PX4 Host Target, connect a joystick to the host computer and configure the joystick using QGroundControl. In this example, we use Logitech F310 Gamepad. The below figure shows the configuration settings in QGroundControl, required for the joystick.

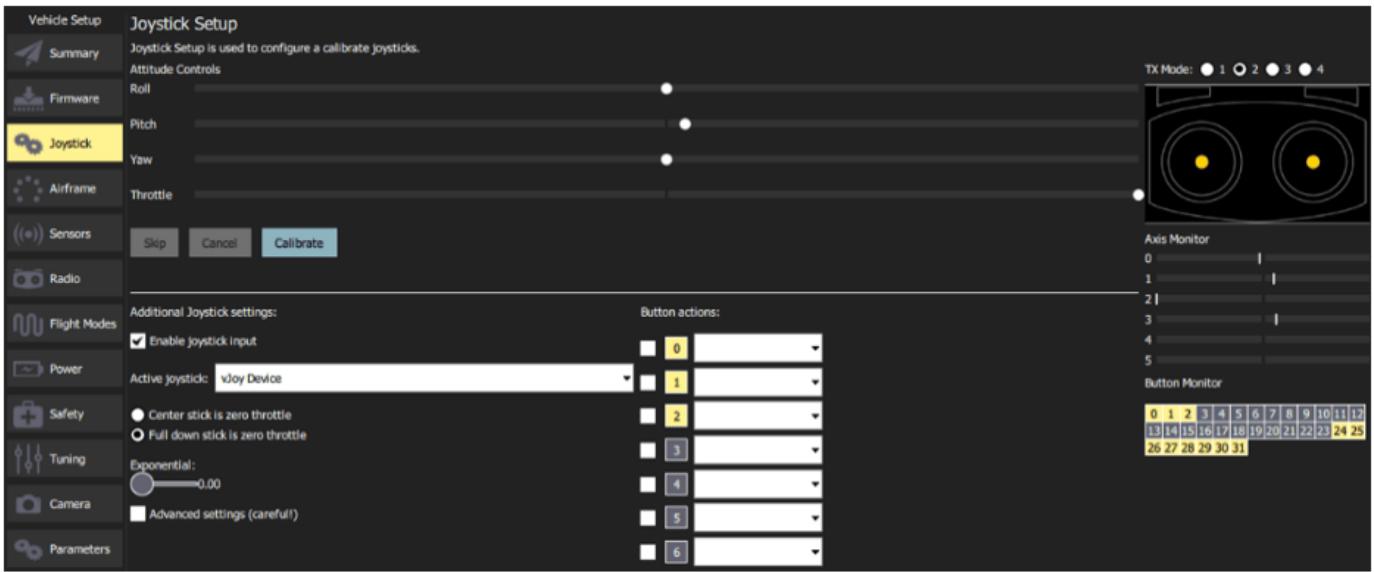


Connecting Radio Control Transmitter

- While the model is running on the PX4 Host Target, connect a radio control transmitter to the host computer using different interfaces provided by various manufacturers. In this example, we use FlySky FS-i6 radio control transmitter.

The FlySky FS-i6 can be connected to the host computer's microphone input port using a trainer cable. An external software utility such as SmartPropoPlus can be used to generate a virtual joystick device, which mimics the behavior of joystick while taking stick inputs from radio control transmitter.

Create a virtual joystick, connect the radio control transmitter to the host computer, and while the model is running on PX4 Host Target, configure the joystick using QGroundControl. The below figure shows the configuration settings in QGroundControl, required for the radio control transmitter.



For further understanding of how the joystick is connected with QGroundControl and how QGroundControl communicates joystick values to PX4 Host Target, refer to this page.

Task 6 - Analyze Response to Inputs from Joystick or Radio Control Transmitter

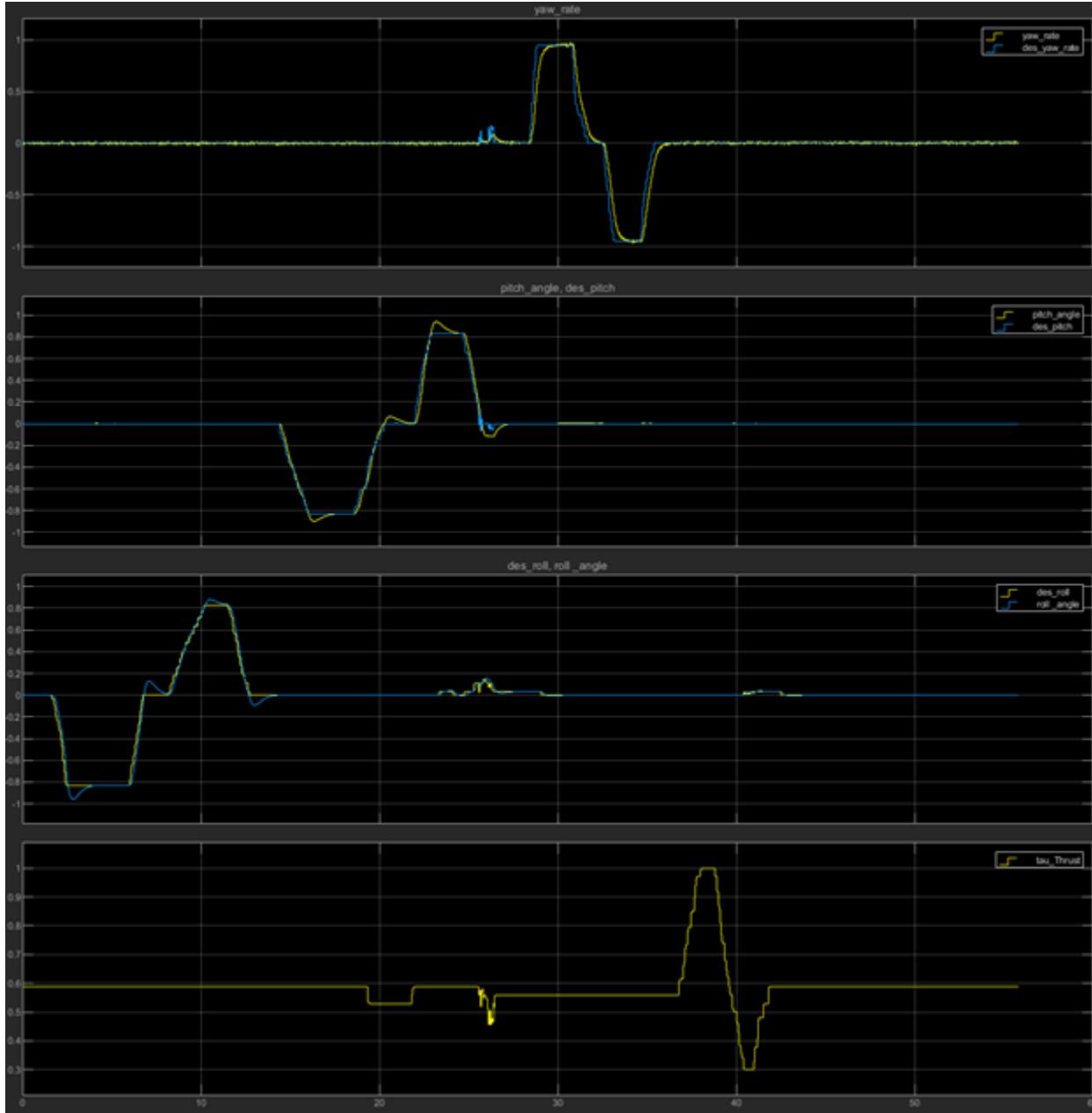
Once the configuration is done correctly, the data based on position of joystick is available in the uORB message `manual_control_setpoint`.

The response of the simulated vehicle to the various inputs are as follows:

- Throttle stick input corresponds to rate at which the quadrotor vehicle climbs or descends. The default quadrotor vehicle of jMAVSim simulator hovers at around 50% of throttle input (that is, 1500us actuator output).
- If the throttle stick input is more than the input required for hovering, the vehicle gains the altitude and the climb rate is proportional to the positive difference between actual throttle applied and throttle required for hover.
- If the throttle stick input is less than input required for hovering, the vehicle loses the altitude and the descend rate is proportional to the negative difference between actual throttle applied and throttle required for hover as well as the height from which vehicle starts the descend.
- Pitch and roll stick inputs corresponds to pitch and roll angle of the vehicle respectively. Center stick results in a level flight with zero pitch / roll. Depending on the direction of stick, the pitch / roll angle increases or decreases linearly up to the maximum limit, which is set at 50 degrees for this example. The maximum limit of pitch / roll angle is usually inferred from vehicle characteristics, and for a more agile vehicle, it can be higher.
- Yaw stick input corresponds to the rate at which the yaw angle increases or decreases. When the yaw stick is kept centered, the yaw angle remains constant. Depending on the direction of stick, the angle increases or decreases.

If you use the **Monitor and Tune** option to run the model on PX4 Host Target (as explained in Task 4), you can use real-time data logging. This is helpful while tuning PID controllers for a desired response. The below figure shows Simulink viewer which is configured to monitor current values of

vehicle attitude and applied input. The figure indicates that the PX4 Autopilot is able to follow applied input in all three rotational axes.



To log various variables into the MATLAB® workspace, the ToWorkspace block of Simulink can be used. For configuration of logging parameters during the Monitor and Tune operation, refer to Set External Mode Properties for Logging to Workspace.

For more information about tuning PID controllers for attitude control, see the example “Position Tracking for X-Configuration Quadcopter” on page 1-149.

Position Tracking for X-Configuration Quadcopter

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to design a position controller for an X-configuration quadcopter. In this example, you also verify the controller design using the PX4 Host Target and jMAVSim simulator.

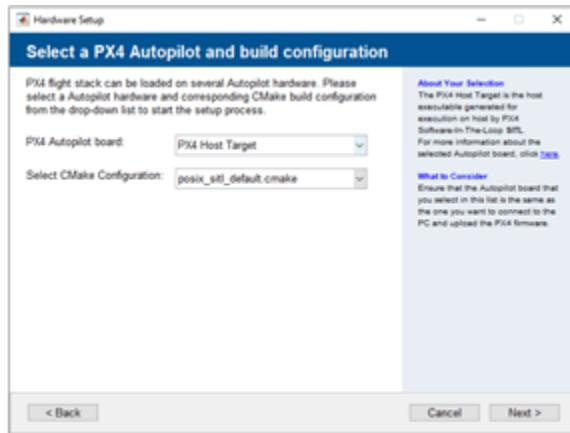
Introduction

UAV Toolbox Support Package for PX4 Autopilots enables you to use Simulink® to design flight controller algorithm to stabilize the vehicle based on the current vehicle attitude, position and velocity, and also track the desired attitude using Simulink.

In this example, you will learn how to use the PX4 Host Target and jMAVSim simulator to design as well as verify position controller for X-configuration quadrotor vehicle and control the vehicle position using various sliders, available in Simulink model. The jMAVSim simulator, which is part of the Software In The Loop (SITL) simulation as defined in PX4 website and is installed as part of the support package installation.

Prerequisites

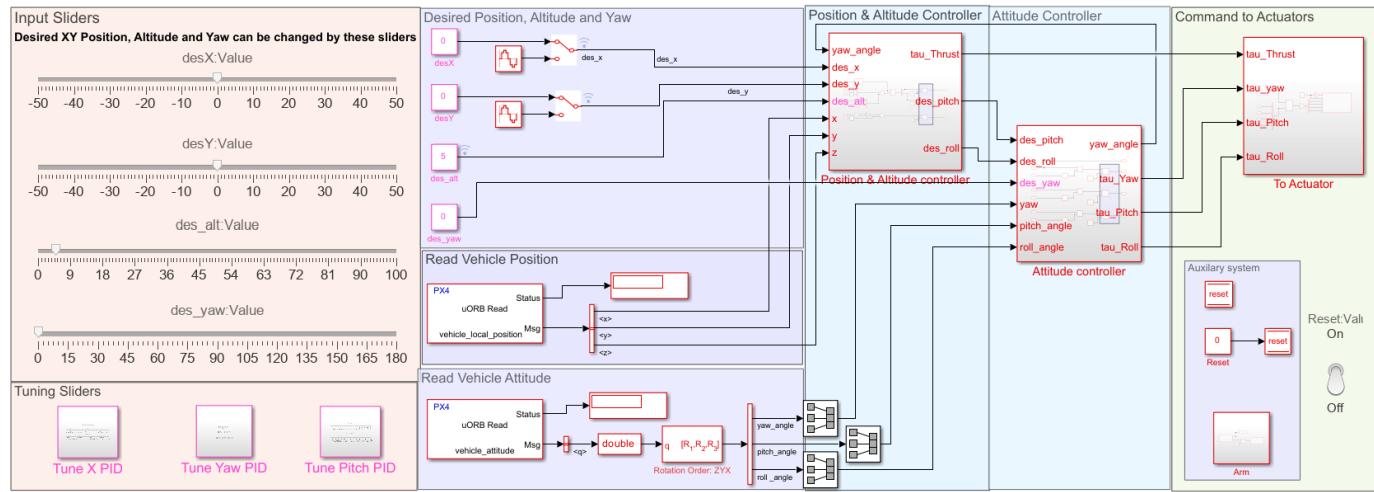
- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens. In the Hardware Setup screen **Select a PX4 Autopilot and build configuration**, select **PX4 Host Target** as the Hardware board from the drop-down list.



Model

Open the px4demo_PositionController_quadrotor model.

Altitude and Position Tracking Controller for X type Quadrotor Vehicle using UAV Toolbox Support Package for PX4 Autopilots



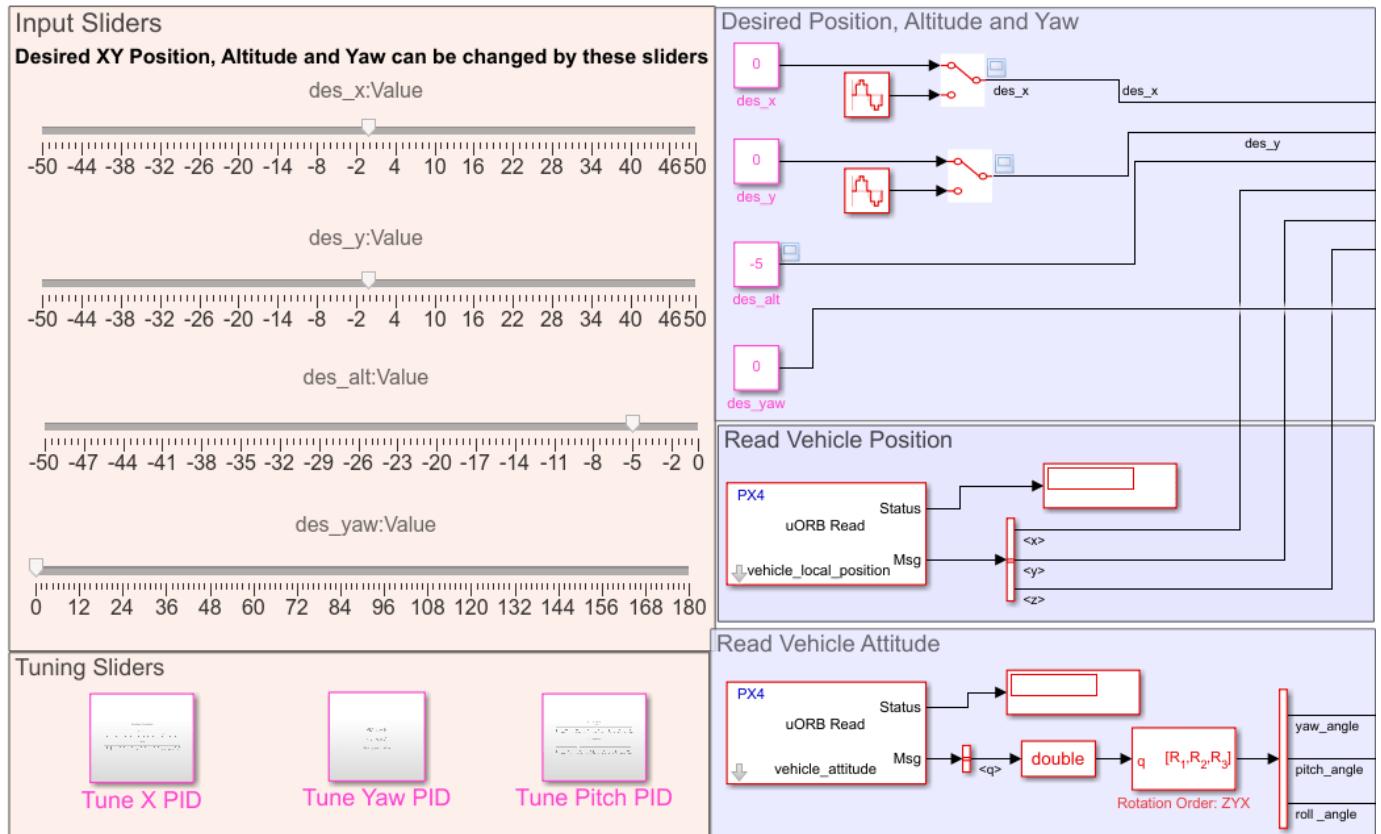
Copyright 2019-2022 The MathWorks, Inc.

The model implements a Proportional-Integral-Derivative (PID) controller to control the position and attitude of an X-configuration quadrotor aerial vehicle. At each time step, the algorithm adjusts rotational speeds of different rotors to track the desired attitude, based on the position error.

Task 1 - Read Desired Position and Current Position

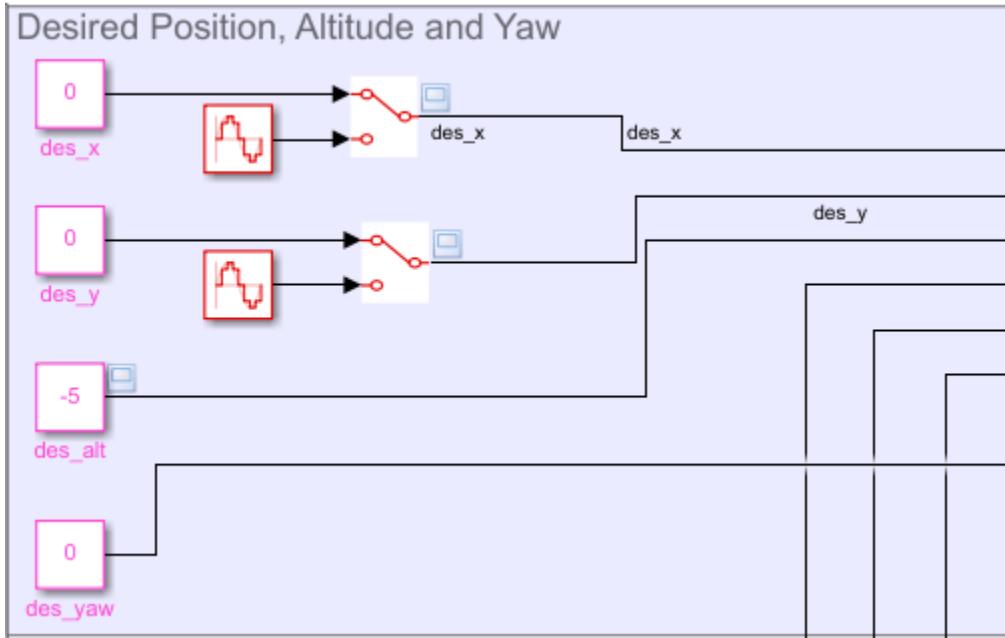
In this example, consider the influence of wind gust on the quadcopter while it is flying. The quadcopter must try to maintain the desired position and altitude. According to the error in position, the pitch and roll commands are generated and the output to the actuator motors are modified.

The *Input Sliders* in the example model can be used to provide the desired coordinates of the flight of the quadcopter.

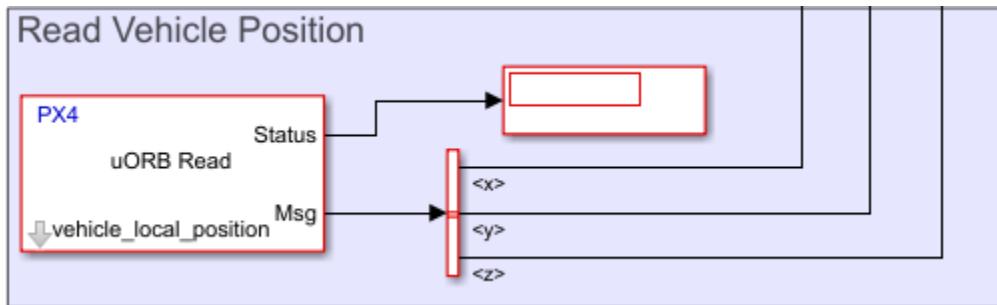


- The variable des_alt represents an altitude at which the vehicle hovers. The altitude value can be set using the respective slider or directly changing the value of constant des_alt.
- The desired position for X and Y can be set by assigning desired values to constants des_x and des_y or using respective sliders.

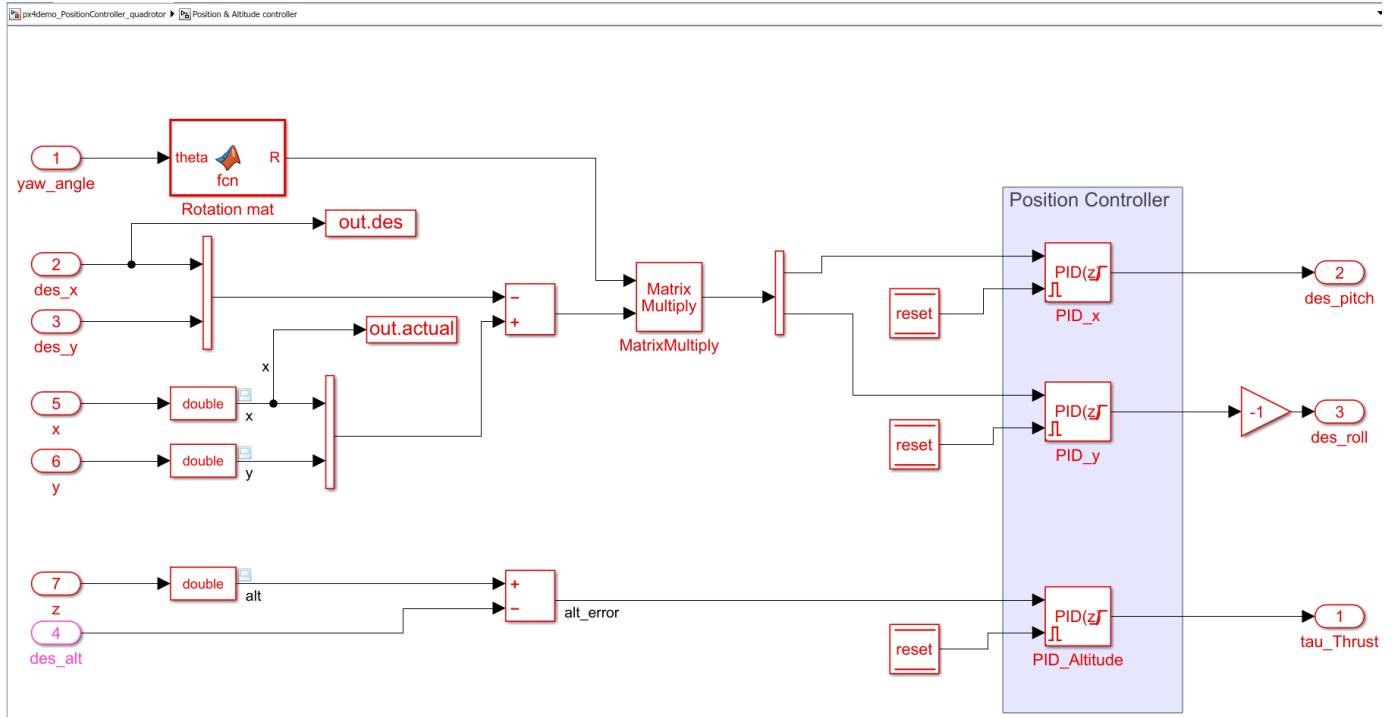
In the *Desired Position, Altitude and Yaw* area, you can also provide dynamic inputs (instead of only static inputs) to test the tracking capabilities of the position controller. In this example, the sine wave is used as the dynamic input and can be selected using respective manual switches.



The vehicle's position in the NED reference frame and its rates can be accessed using uORB Read block, which is configured to read the message 'vehicle_local_position'.



Task 2 - Design Position Controller

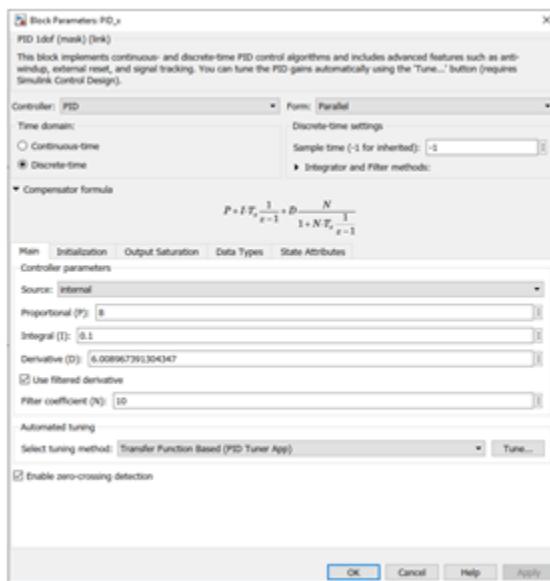


In the *Position & Altitude Control* subsystem, three separate PID blocks utilize the difference between the desired value and the current value (that is read using the uORB read block) to generate the requirement for rotation angles pitch and roll. An output of each PID block is restricted between the predefined maximum and minimum values to limit roll and pitch angles. The limit is set at 40 degrees for this example. The maximum limit of pitch and roll angle is usually inferred from vehicle characteristics, and for a more agile vehicle, it can be higher.



Each PID block utilizes filtered derivatives to eliminate the amplification of any undesired noise signal due to signal differentiation.

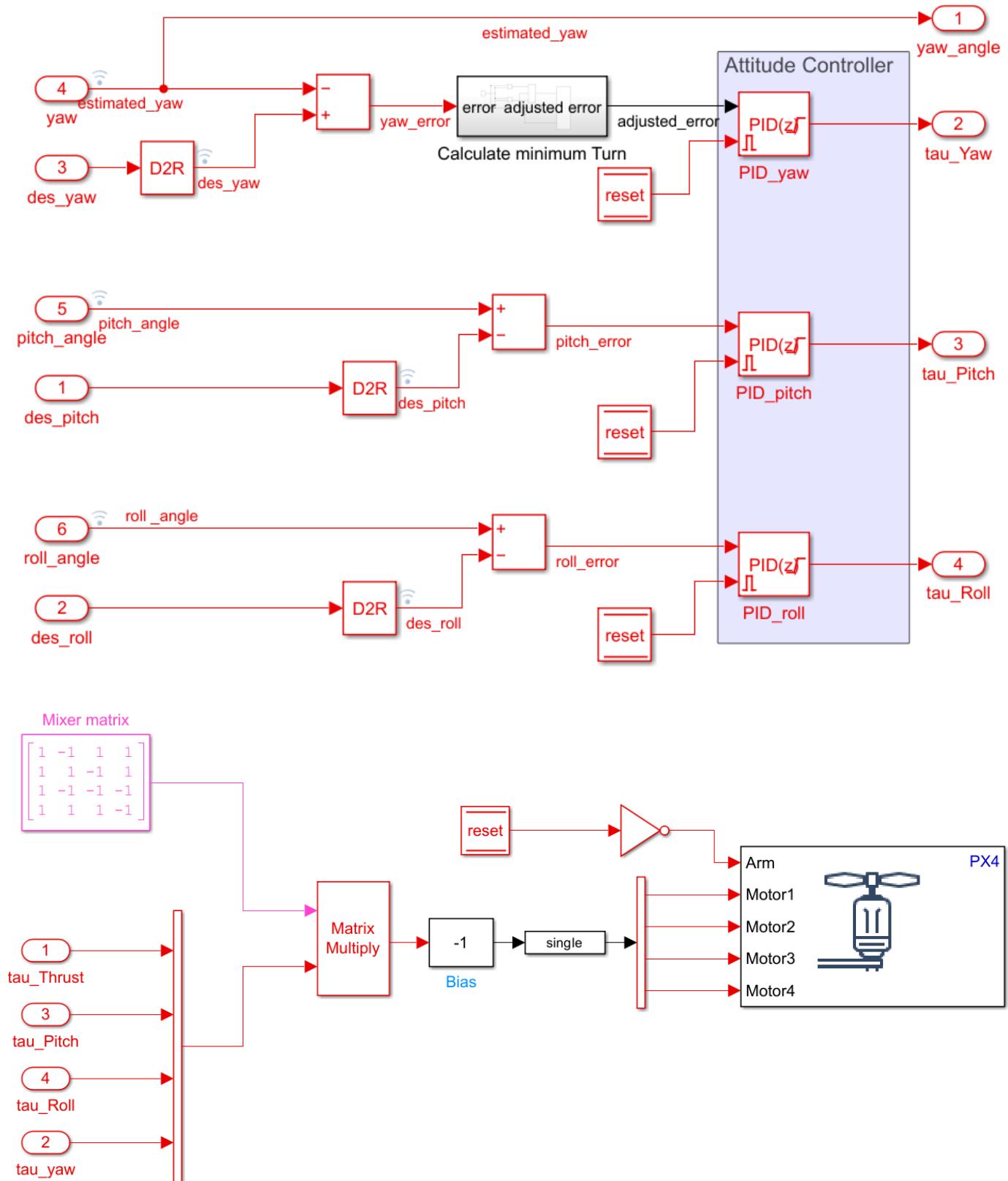
The NED reference frame is an inertial reference frame and the direction of the respective axis remains fixed with respect to a starting position on the ground. On the contrary the rotation angles, roll and pitch are measured with respect to a moving reference frame that is attached to vehicle's body. This is because actuators are rigidly mounted with vehicle body. Any non-zero yaw angle changes vehicle heading and orientation of the vehicle in NED reference frame. This results in a phenomenon where, depending on the yaw angle of the vehicle, the same pitch or roll angle causes different translation motion in a plane. Hence, you must account for the non-zero yaw angle while generating pitch and roll commands from the position error. This has been achieved by calculating a 2-D rotation matrix and multiplying position error with the rotation matrix.



Task 3 - Design Attitude Controller and Mix Angular Rate Demands

The Attitude Controller acts on the outputs from the Position Controller and modifies actuator output to achieve the desired pitch and roll. The design of the attitude controller and mixing algorithm is similar to the example "Attitude Control for X-Configuration Quadcopter Using External Input" on page 1-133.

Any non-zero yaw angle changes vehicle heading. So you must account for the non-zero yaw angle while generating pitch and roll commands from the position error.



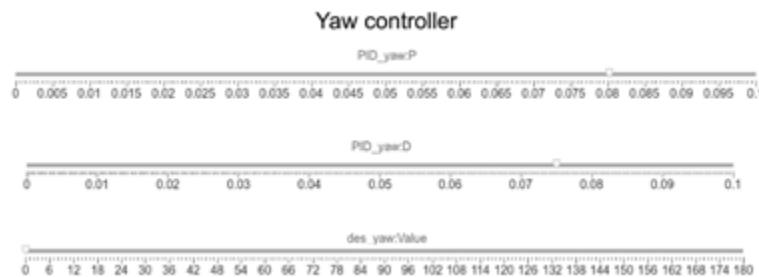
Task 4 - Tune PID Using Monitor and Tune

For instructions to build the model and perform Monitor and Tune operation with data monitoring, refer to the example “Attitude Control for X-Configuration Quadcopter Using External Input” on page 1-133.

When you start Monitor and Tune, the jMAVSIM simulator is also launched.

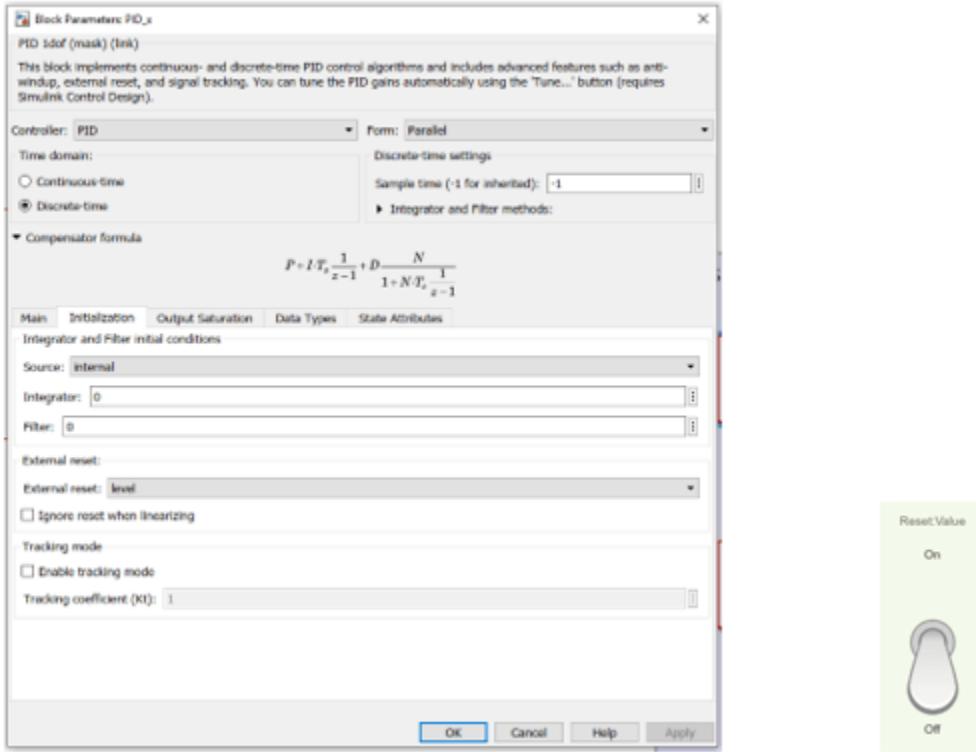
In this example, you can use the Monitor and Tune functionality to change the PID values. Each PID gain can be connected to individual sliders and the gain values can be altered using sliders while simulating in external mode. The *Tuning Sliders* area in the model helps you to achieve this. The effect of any changes in gains can be observed in real-time using various Scope blocks.

The following figure shows one such tuning subsystem. Here, the desired value of the yaw angle is connected to a slider, which can be used to modify the desired yaw angle while simulating the model in external mode. The desired and actual value of yaw angle can be connected to the scope, and the response of the vehicle to any change in desired value can be observed. According to the vehicle response, gains can be adjusted from the respective sliders, and the process of giving an excitation and observing the response can be repeated.



While tuning various PID controllers for the desired response, you may encounter frequent crashes or diverging oscillations in any degree of freedom. The vehicle state can be restored to its initial state using jMAVSIM simulator. However, the accumulated integral part of various PID controllers will again destabilize the vehicle even if the vehicle state has been restored to its initial state. This can be avoided by using an external reset for the PID block.

In this example, all PID blocks are configured for external reset and the reset variable is connected to a toggle switch. In the event where a reset of vehicle state is required, the toggle switch can be set to high. This resets all the accumulated integral part and send minimum output to actuators. Once the state is restored using jMAVSIM, the toggle switch can be set to low, and this results in a normal controller operation.



To fine-tune the system response and evaluate the controller design, various performance metrics such as Integrated Squared Error (ISE), Integrated Absolute Error (IAE), Integrated Time Squared Error (ITSE), and Maximum Peak overshoot (Mp) can be utilized.

A considerable value of each performance metric reveals a particular flaw in the controller, and hence allows you to take further steps to rectify the flaw and achieve the desired response. For example:

- A system with small sustained oscillations around the setpoint has a large IAE.
- A system with a slow response to the applied input has a large ISE.
- A system with a small steady-state error has a large ITSE.

The applied excitation and response of the system can be logged for a certain time to calculate such performance metrics. The *ToWorkspace* block of Simulink can be used to log various variables into the MATLAB® workspace. For configuration of logging parameters during the external mode of simulation, refer to Set External Mode Properties for Logging to Workspace.

Once variables are logged, an error can be calculated and normalized using applied input. The normalized error history along with time, can be utilized further to calculate ISE, IAE, ITSE and Mp.

The following figure shows an example of such a process for control system design.

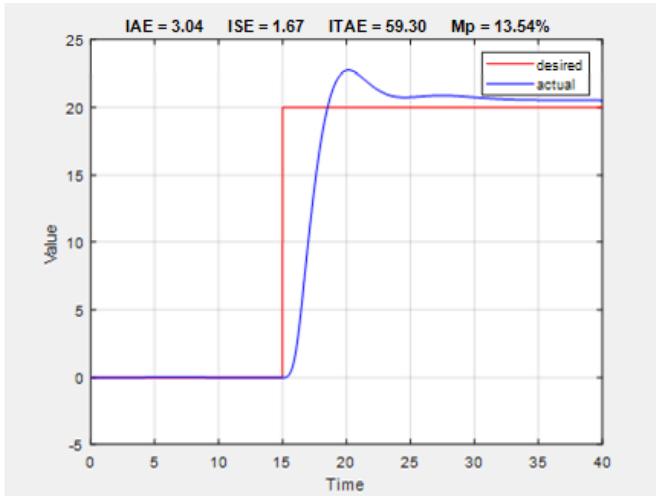


Figure A $K_p = 4; K_i = 0.1; K_d = 6$

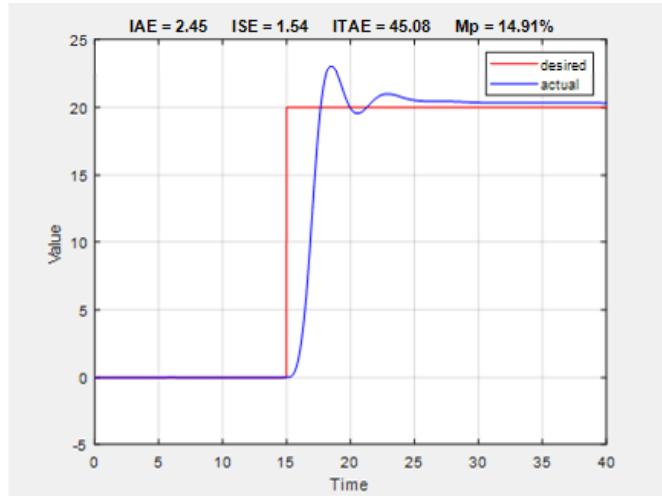


Figure B $K_p = 8; K_i = 0.1; K_d = 6$

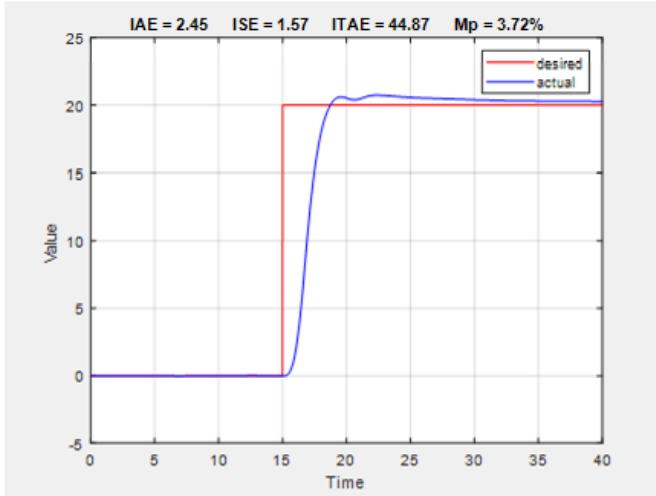


Figure C $K_p = 8; K_i = 0.1; K_d = 10$

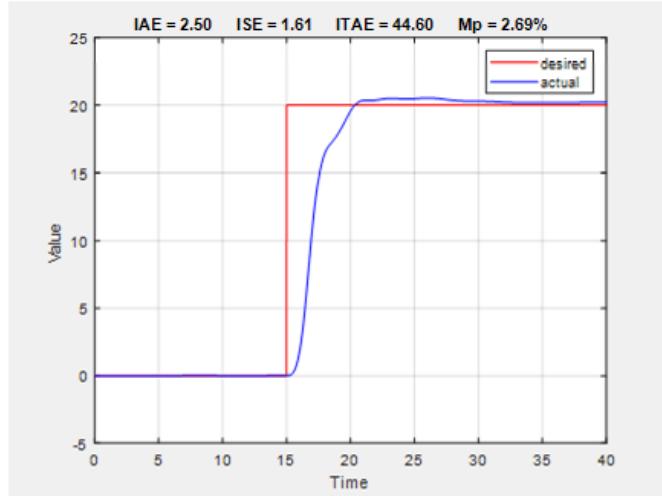
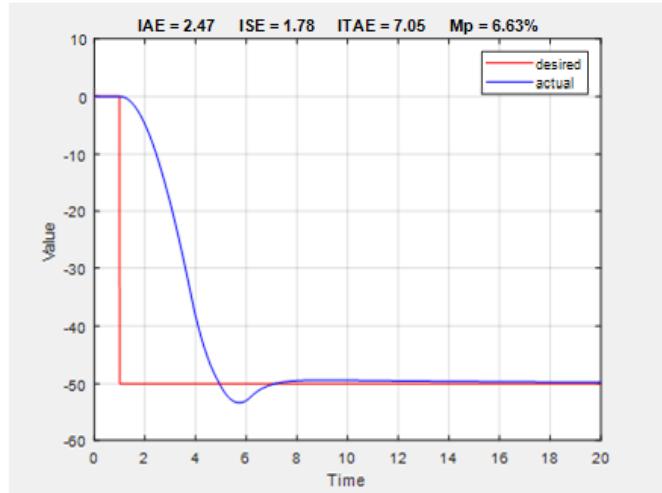
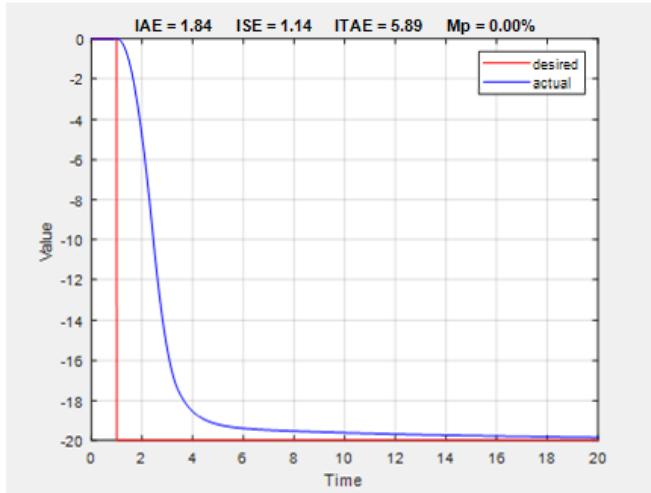


Figure D $K_p = 8; K_i = 0.1; K_d = 12$

In this example, starting from an origin, a step input of 20m is applied in the x-direction at 15s, and the vehicle response for the total 40s is recorded.

1. Figure A indicates that the controller is not able to correct the overshoot, and hence the response has a large steady-state error.
2. Figure B indicates the improved response due to increment in the proportional gain. Except for the overshoot, all the performance metrics are improved.
3. Figure C indicates improvement in transient response due to increment in the derivative gain. It can be observed from Figure C that, all the performance metrics except the peak overshoot, remains the same as the earlier case. This observation indicates direct correlation between overshoot and derivative gain.
4. Any excessive derivative gain can result in a sluggish response. This phenomenon can be seen in Figure C and Figure D, where increasing the derivative gain further results in a slow system response with comparatively high ISE and IAE.

Limitations of Design with Position Controller



- Figure E and Figure F show a vehicle response to a step input for altitude 20m and 50m, respectively.

For a step input of 20m, there is no overshoot in the vehicle response. However, for a step input of 50m, there is a considerable overshoot. Hence, even though this controller can be tuned to give the desired response for a certain type and magnitude of input signals, the same response cannot be achieved for a vast range of operations.

- If the initial desired XY position is non-zero, then depending on the desired position, the controller may not be able to handle position and altitude error simultaneously and the vehicle may crash.
- For details about the limitation and improvement of the present controller, see “Position Tracking for X-Configuration Quadcopter Using Rate Controller” on page 1-160.

See Also

Related Examples

- “Attitude Control for X-Configuration Quadcopter Using External Input” on page 1-133

Position Tracking for X-Configuration Quadcopter Using Rate Controller

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to design a position controller using rate control for an X-configuration quadcopter. In this example, you also verify the controller design using the PX4 Host Target and jMAVSIM simulator.

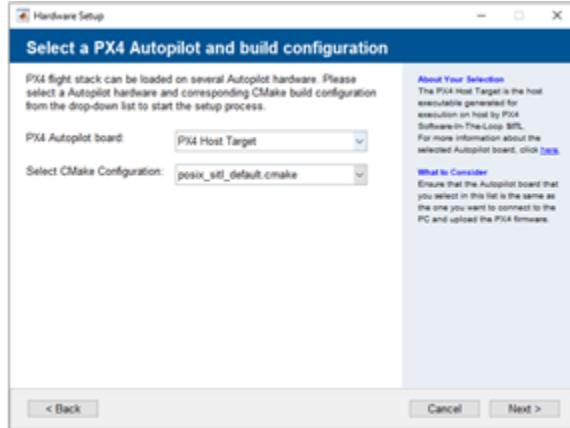
Introduction

UAV Toolbox Support Package for PX4 Autopilots enables you use Simulink® to design flight controller algorithm to stabilize the vehicle based on the current vehicle attitude, position and velocity, and also track the desired attitude using Simulink.

In this example, you will learn how to use the PX4 Host Target and jMAVSIM simulator to design and verify position and velocity controller for X-configuration quadrotor vehicle and control the vehicle position using various sliders, available in Simulink model. The jMAVSIM simulator, which is part of the Software In The Loop (SITL) simulation as defined in PX4 website and is installed as part of the support package installation.

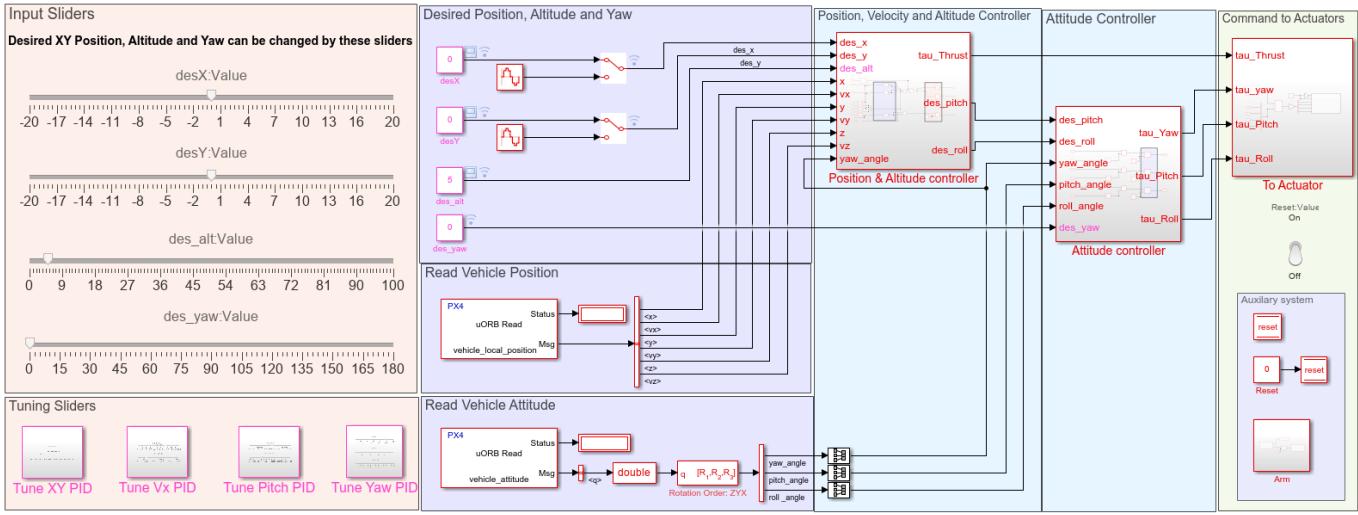
Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens. In the Hardware Setup screen **Select a PX4 Autopilot and build configuration**, select **PX4 Host Target** as the Hardware board from the drop-down list.



Model

Open the px4demo_PositionAndRateController model.

Altitude and Position Tracking via Rate Controllers for X type Quadrotor Vehicle using UAV Toolbox Support Package for PX4 Autopilots


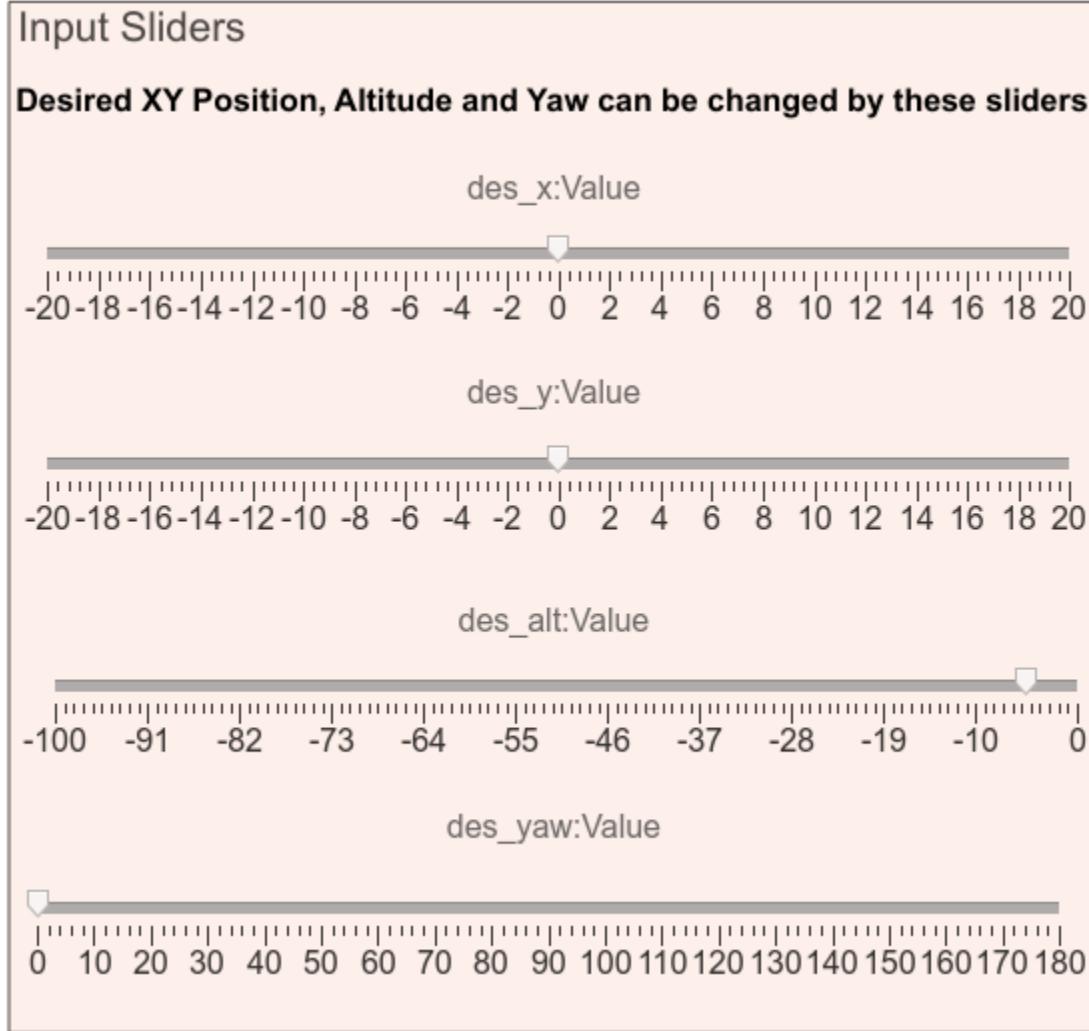
Copyright 2019-2022 The MathWorks, Inc.

The model implements a Proportional-Integral-Derivative (PID) controller to control the position, velocity and attitude of an X-configuration quadrotor aerial vehicle. At each time step, the algorithm adjusts rotational speeds of different rotors to track the desired attitude, depicted by the velocity error.

Task 1 - Read Desired Position and Current Position

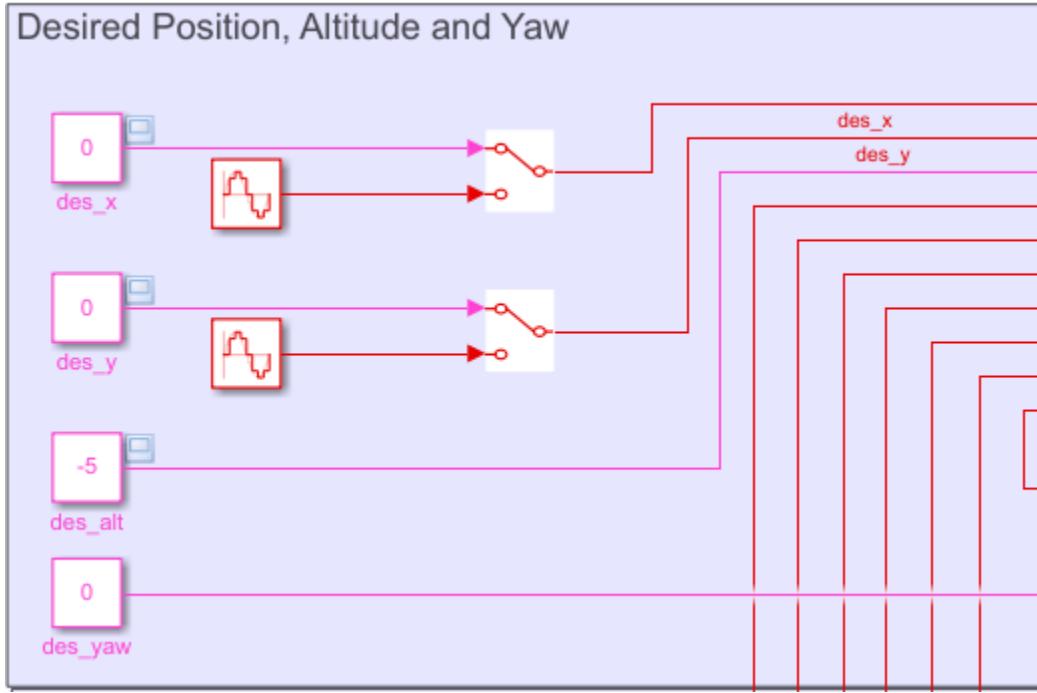
In this example, we consider the influence of wind gust on the quadcopter while it is flying. The quadcopter must try to maintain the desired position and altitude. According to the error in position, the pitch and roll commands are generated and the output to the actuator motors are modified.

The *Input Sliders* in the model can be used to provide the desired coordinates of the flight of the quadcopter.

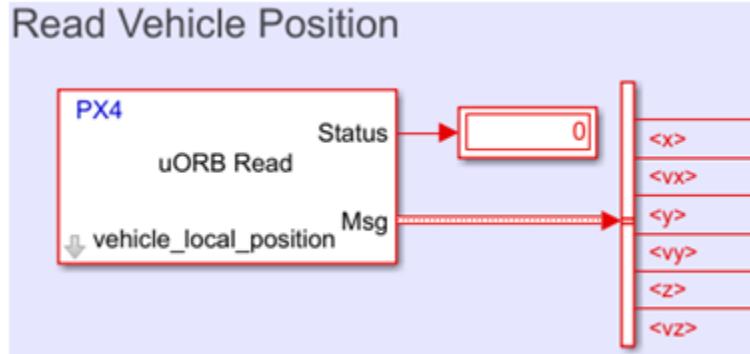


- The variable des_alt represents an altitude at which the vehicle hovers. The altitude value can be set using the respective slider or directly changing the value of constant des_alt.
- The desired position for X and Y can be set by assigning desired values to constants des_x and des_y or using respective sliders.

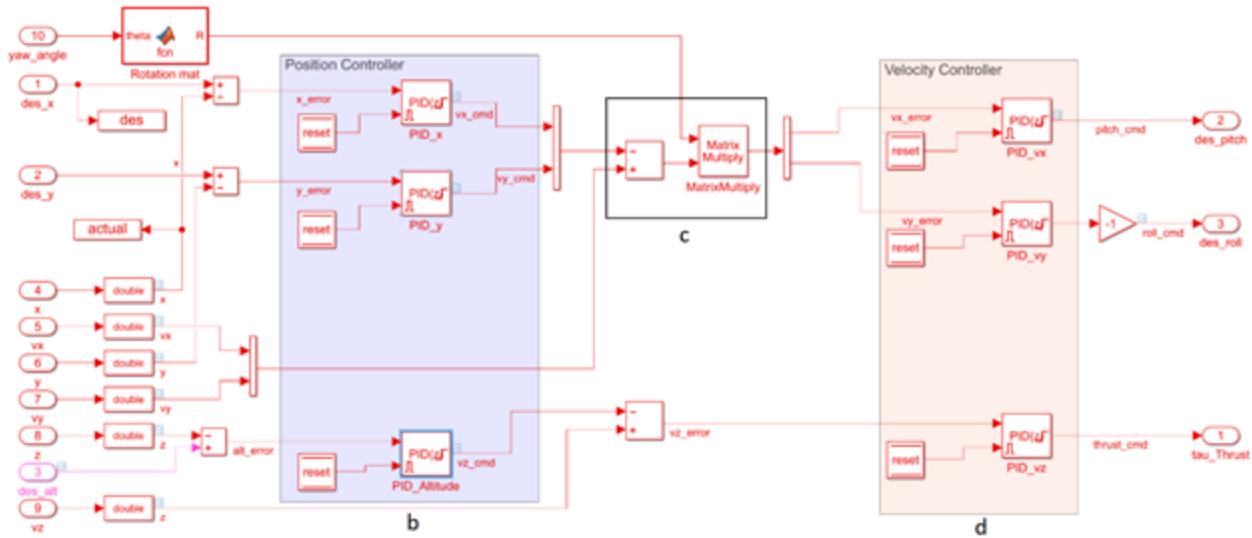
In the *Desired Position, Altitude and Yaw* area, you can also provide dynamic inputs (instead of only static inputs) to test the tracking capabilities of the position controller. In this example, the sine wave is used as the dynamic input and can be selected using respective manual switches.



The vehicle's position in the NED reference frame and its rates can be accessed using uORB Read block, which is configured to read the message 'vehicle_local_position'.



Task 2 - Design Position and Velocity Controller



In the *Position & Altitude Control* subsystem, three separate PID blocks utilize the difference between the desired and current value to generate the requirement for x and y velocities. The output of each PID block is restricted between predefined maximum and minimum values to limit velocities. For this example, the limit is set at 20 m/s in x & y direction, and 10 m/s for altitude. The maximum limits of velocities are usually inferred from vehicle characteristics.

- For a particular vehicle, maximum achievable translation velocities depend on limits of pitch and roll angles. The pitch and roll angles are restricted up to 50 degrees for the simulated vehicle. The limits of achievable translation velocities can be set by manually giving maximum pitch / roll stick input in the example 'px4demo_AttitudeControllerWithJoystick_quadrotor' and observing x & y velocities.
- The maximum rate of climb of the vehicle is primarily dependent on the performance of the propulsion system and the mass of the vehicle. For the simulated vehicle, the maximum rate of climb can be set by manually giving maximum throttle stick input in the example 'px4demo_AttitudeControllerWithJoystick_quadrotor' and observing z velocity.
- Since the velocities are measured in the NED reference frame, the current velocity error is computed first before applying the correction for the non-zero yaw angle via the rotation matrix.
- The corrected velocity error is fed to separate PIDs to generate the requirement for rotation angles, pitch and roll. An output of each PID block is restricted between predefined maximum and minimum values to limit roll and pitch angles.

The rest of the design for attitude controller and mixer is similar to the example 'Position Tracking for X-Configuration Quadcopter'.

Task 3 - Tune PID Using Monitor and Tune

For instructions to build the model and perform Monitor and Tune operation with data monitoring, refer to the example 'px4demo_AttitudeControllerWithJoystick_quadrotor'.

When you start Monitor and Tune, the jMaVSim simulator is also launched.

For more details about tuning the controller using PID, refer to the example 'Position Tracking for X-Configuration Quadcopter'.

Advantages of the Present Controller over the Position-Only Control

- As observed in the example 'Position Tracking for X-Configuration Quadcopter', for a vast range of input signals, uniform system response cannot be achieved by the position control only. For a step input of 20m and 50m respectively, the system has a different response.
- This behavior comes from the fact that for a quadrotor vehicle, we cannot control the position directly. The manipulated or control variable in a quadrotor is the rotational speed of four rotors. The difference in rotational speeds results in a difference in thrust and this asymmetrical thrust generates a moment around various axes. The moment induces angular rates, which changes vehicle attitude (that is, roll, pitch and yaw angles). Non-zero pitch and roll angles generate translation velocities, due to which finally the quadrotor position changes.
- In only position control, we are trying to control the vehicle position while changing rotational speeds of four rotors. There is a huge time delay between the moment at which changes are made in rotational speed and the moment at which the effect of the change in rotational speed observed in the vehicle position. This delay in propagation of vehicle dynamics usually results in an overshoot.

With the addition of a velocity controller in between the position controller and attitude controller, this delay reduces and the controller performance can be improved. Figure C and Figure D show the response of the vehicle to step input of 20m and 50m respectively with rate control. Here, the vehicle has the same response to both inputs without considerable overshoot.

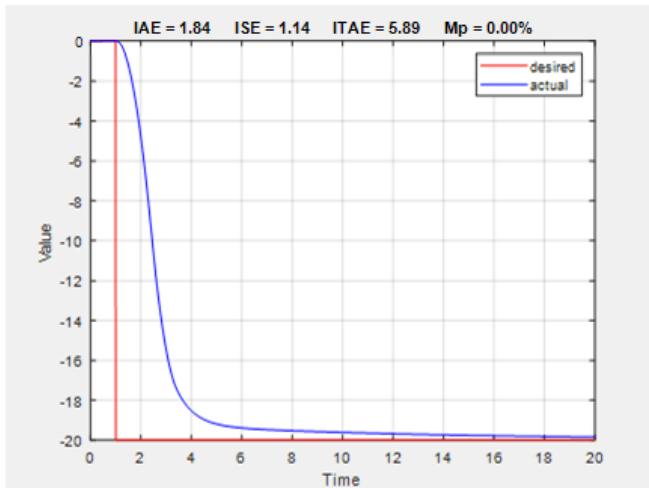


Figure A Step of 20m in Altitude - without rate controller

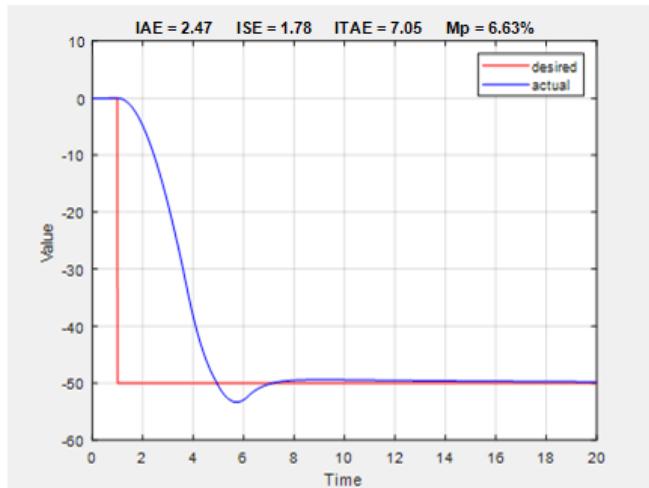


Figure B Step of 50m in Altitude - without rate controller

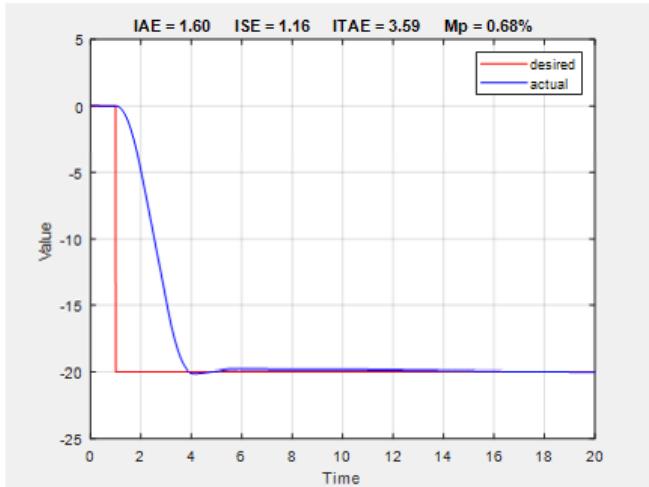


Figure C Step of 20m in Altitude - with a rate controller

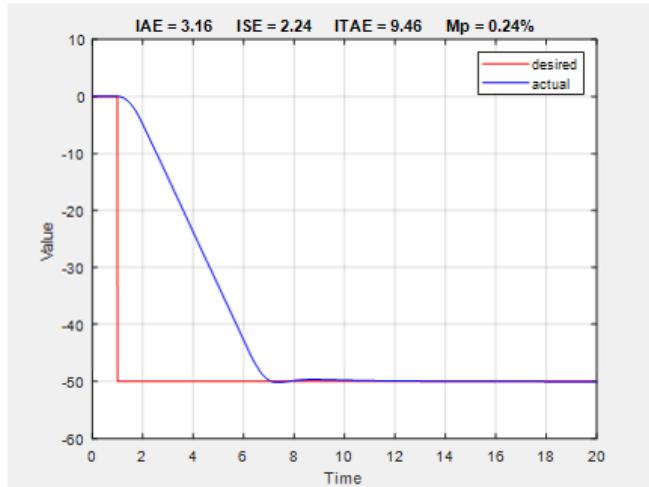
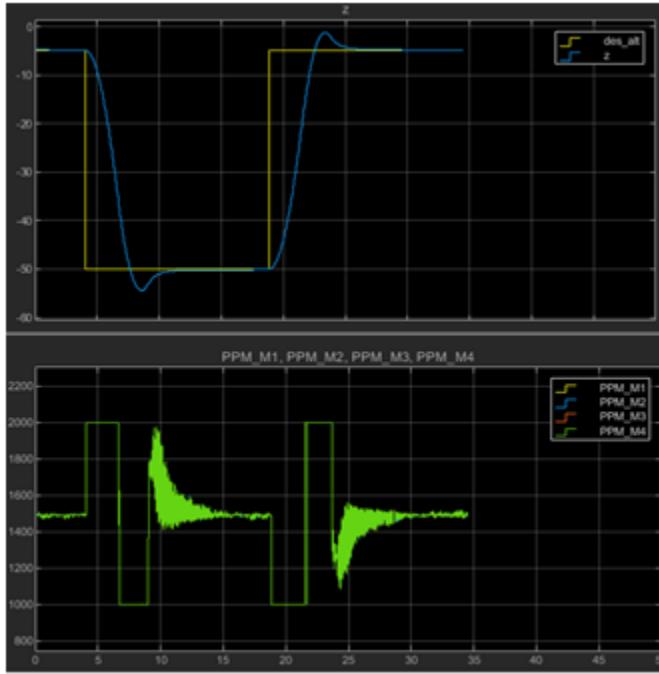


Figure D Step of 50m in Altitude - with a rate controller

- Improvement in the controller performance can be easily observed in the actuator output for the vehicle. The below figure shows actuator outputs for a controller with only the position control.

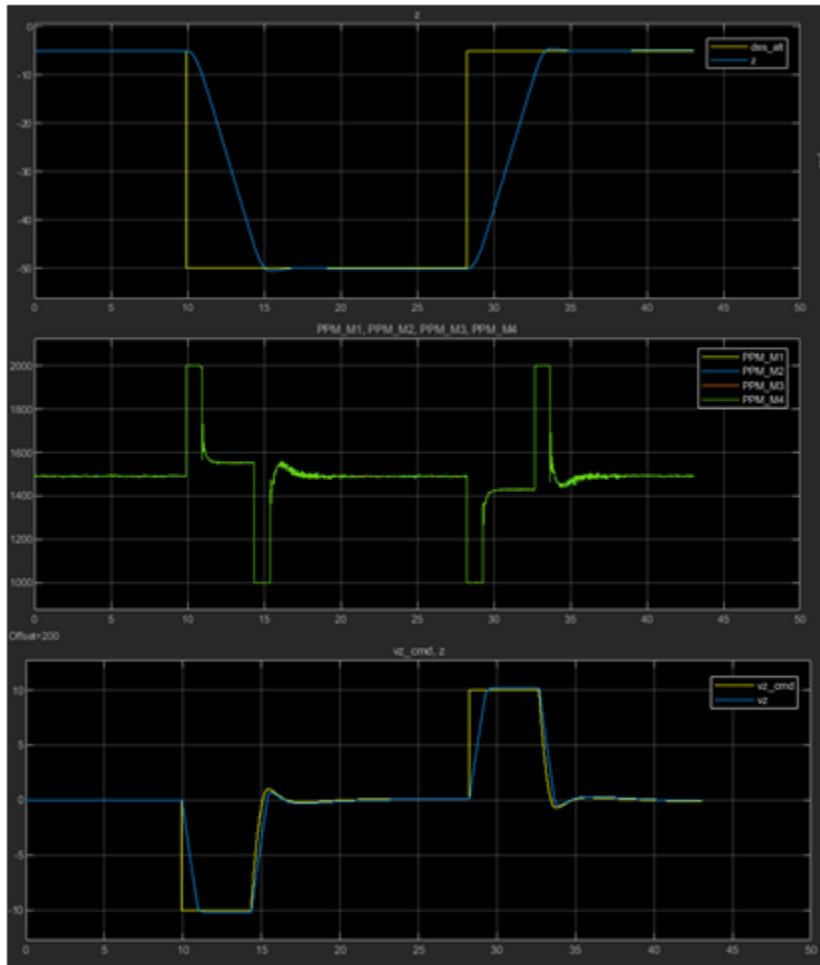


Here actuator outputs are governed by an error in the position. As soon as the step input is applied, actuator outputs saturate to their maximum value due to large error in position. The vehicle starts gaining the altitude with saturated actuator outputs for most of the time. During the climb phase, the vehicle accumulates the vertical velocity, but since it is not controlled directly, actuator outputs remain saturated due to large position error.

Because of this, the vehicle reaches close to the desired altitude with large vertical velocity and the position error quickly reduces. The derivative action kicks in at this instance, and it results in a sudden change in actuator outputs to its minimum value. This behavior closely resembles with bang-bang or on-off controller and results in a significant overshoot.

- In the case of position control with intermediate rate control, instead of position error, actuator outputs are directly governed by velocity error. A large position error generates a significant velocity demand, which is achieved by saturated actuator outputs. Once the desired velocity is achieved, the actuator outputs reduce to maintain the velocity.
- Finally, the vehicle reaches close to the desired altitude with controlled vertical velocity and further, the reduced position error results in a reduction of velocity demand. The controller adjusts actuator outputs to achieve the desired velocity.

This image shows the actuator outputs with both position and rate control.



See Also

Related Examples

- “Position Tracking for X-Configuration Quadcopter” on page 1-149

MAT-file Logging on SD Card for PX4 Autopilots

This example shows you how to efficiently log signals in the MAT-file format from a Simulink® model running on a Pixhawk® hardware, and retrieve the log files from the SD card for further analysis.

Introduction

The UAV Toolbox Support Package for PX4® Autopilots supports logging of signals from your Simulink model on Pixhawk hardware in the MAT-file format. Signal logging enables you to monitor the behavior of the signal and perform analysis of historical data. To use the MAT-file logging feature with the Pixhawk hardware, you must have a Embedded Coder® license.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.
- Refer to the topic “Log Signals on an SD Card”

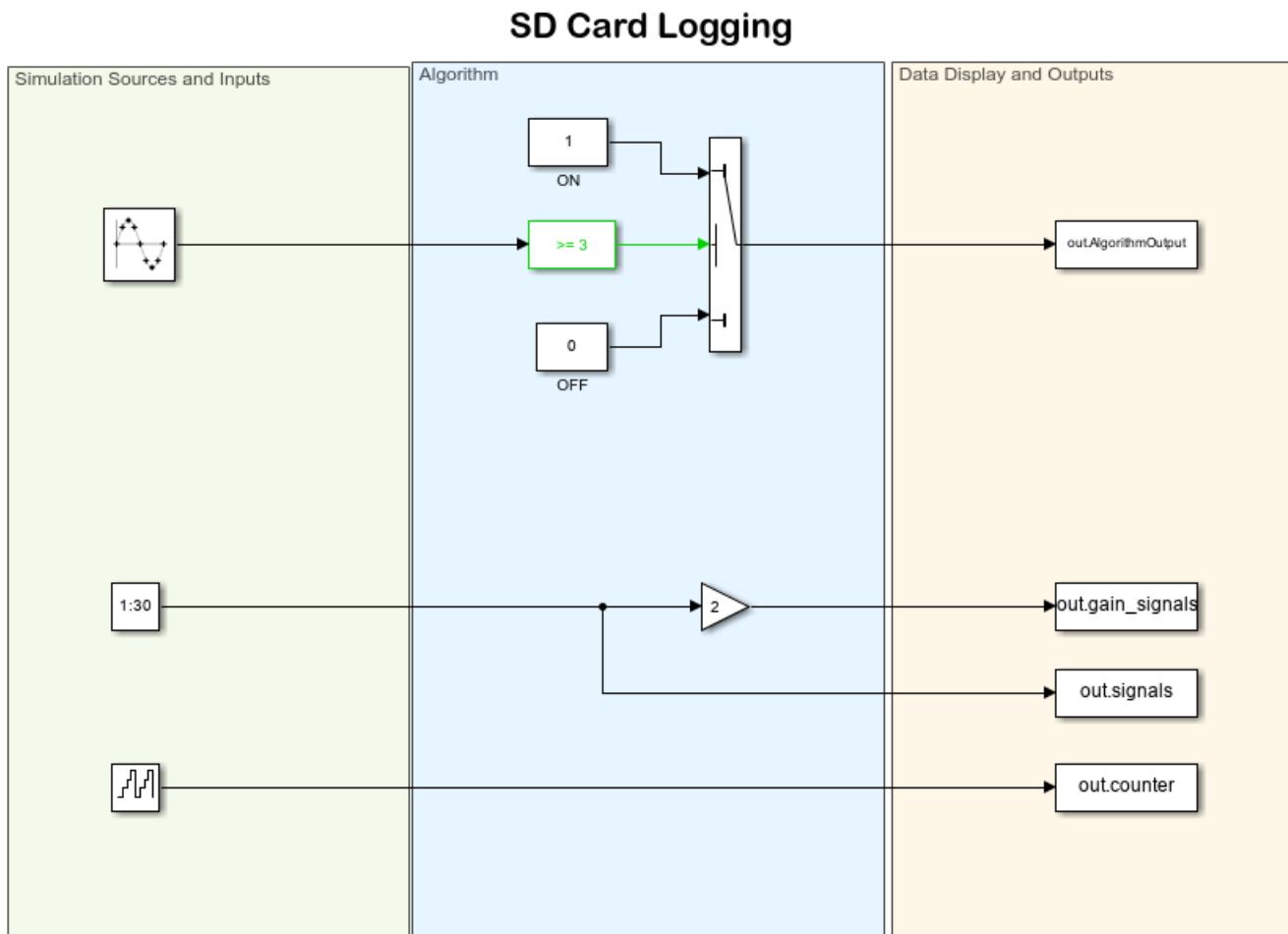
Required Hardware

To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)

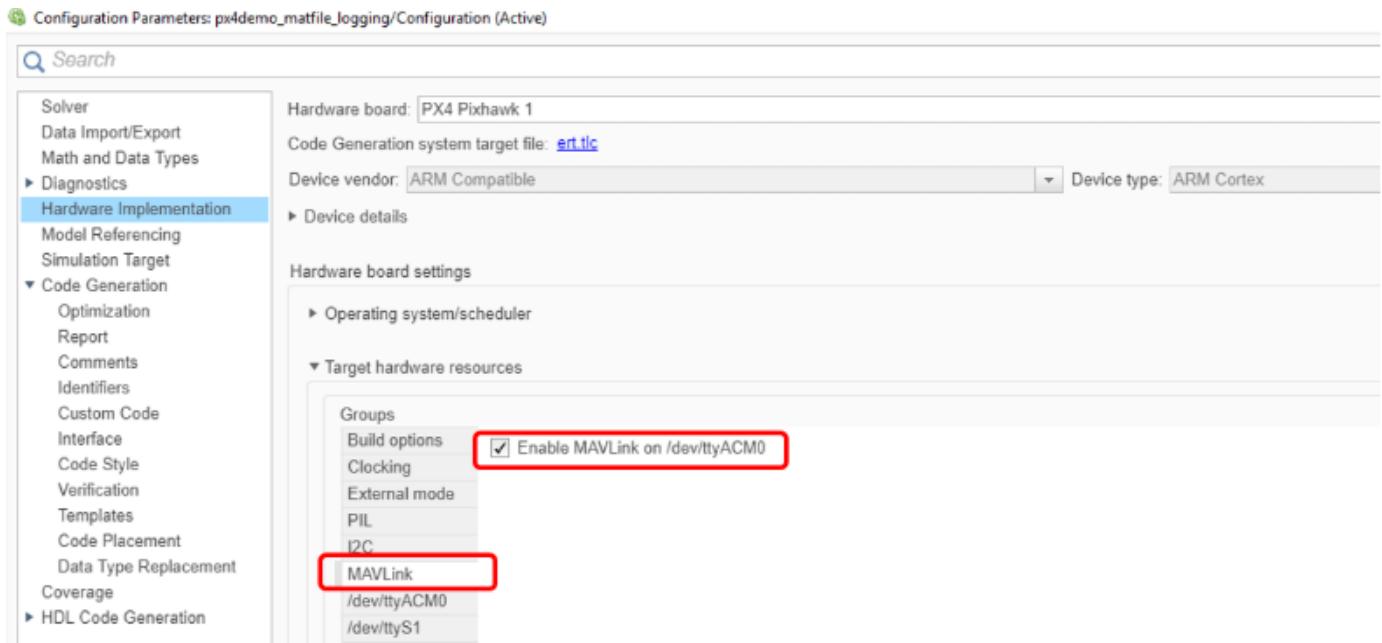
Step 1 - Configure a Simulink Model for MAT-File Logging

1. Open the px4demo_log model. This Simulink model is pre-configured for the Pixhawk 1 with MAT-File logging enabled. In this example, signals from the 'To Workspace' block are logged.

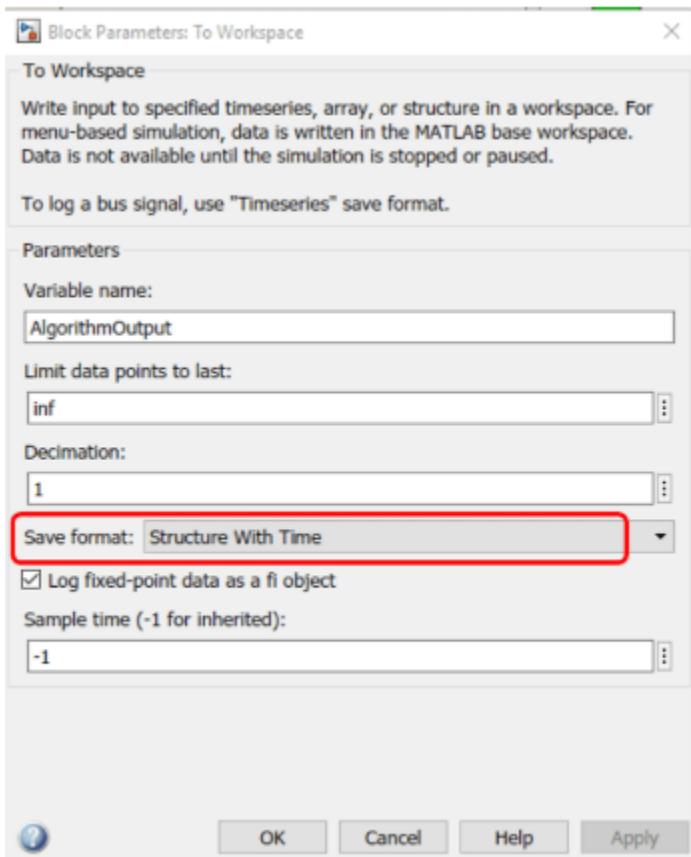


Copyright 2020-2022 The MathWorks, Inc.

2. Open the Model Configuration Parameters dialog box (Go to **Modeling > Model Settings** on the Simulink toolbar).
3. Go to the **Hardware Implementation** pane, and select the name of the target hardware from the **Hardware board** list. If required, you can change the Hardware board selection other than the pre-configured Pixhawk 1 board.
4. MAVLink needs to be enabled to retrieve the log files from SD card inserted on the Pixhawk hardware board. In the Configuration Parameters dialog box, go to **Hardware Implementation > Target hardware resources > MAVLink**, and select **Enable MAVLink on /dev/ttyACM0**.



5. Browse to **Code Generation > Interface > Advanced Parameters**, or type MAT-file logging in the search box.
6. Select the **MAT-file logging** option and click **Apply** to save the changes.
7. Click **OK** to close the dialog box.
8. In the Simulink model, double-click the *To workspace* blocks, and open the Configuration Properties dialog box.
9. Select the Save format as **Array or Structure with Time or Structure**, and click **Apply** and then **OK**.



10. Leave the **Limit data points to last** field as it is.

11. On the **Modeling** tab of Simulink toolbar, enter a value for the stop time. This is the duration for which the signal is logged. However, the model continues to run on the hardware and it is not affected by the time specified.

For example, in this demo model (SD Card Logging), the stop time is 30. The signal will be logged for 30 seconds. If the stop time is Inf, the signal will be logged till the Micro SD card is full or the board is disconnected.

Step 2 - Prepare Model for Simulation and Deployment

1. Execute the px4PrepareModelForMATFileLogging function in the MATLAB® command prompt to set the Limit Data points to last in such a way that there will be no data drops. For more information, see “Prepare Model for Simulation and Deployment”

```
px4PrepareModelForMATFileLogging('px4demo_log')
```

2. After you run this function, a dialog box opens requesting your permission to fix the values for memory optimization. Click **Fix it** to optimize the memory.



3. After you click **Fix it**, the value for **Limit data points to last** for all the **To Workspace** blocks in your model is changed. You will see the below warning about Simulink signals taking up lot of memory.

```
>> px4PrepareModelForMATFileLogging('px4demo_log')
Warning: The memory required to log the selected signals to SD card may cause any of the below
memory issues:
1. Linker RAM overflow issues.
2. Less RAM available to run the application. This may increase the chances of a stack/heap
collision that will cause a hard fault on the AutoPilot.
Use any one or a combination of the following methods to reduce the memory required.
1. Reduce the number of signals to be logged.
2. Change the data type of the signal to be logged to a data type of smaller size.
3. Increase the sample time at which the signals are logged.
4. Increase the decimation of the signals which are logged.
5. Set the Save Format to 'Structure' or 'Array', instead of 'Structure with Time'. For each
signal with 'Structure with Time' format, there will be a duplication of time variable created.
6. Combine the signals to be logged using a Mux, and feed to a 'To Workspace', or 'Scope'
block to avoid overheads.
Continue to tweak the model according to the above suggestions until this warning message no
longer appears.
Refer the link for more information on how to log Simulink signals into SD Card.
> In codertarget.pixhawk.registry.staticMemorySizeforSDCard (line 285)
In px4PrepareModelForMATFileLogging (line 7)
```

In the next step, we shall see how to optimize the memory required for logging.

Step 3 - Optimize the Simulink Model to Reduce the Memory Required for Logging and Deploy on Pixhawk 1 Hardware

Optimize the Simulink model

In this section, we refer to some of the steps mentioned in “Troubleshooting Memory Limitations for MAT-file Logging” section to reduce the memory required for logging.

1. Reduce the number of signals in the constant block from 30 to 20. You can manually change the values or execute the below command:

```
set_param('px4demo_log/Constant','Value','1:20')
```

2. Execute the function `px4PrepareModelForMATFileLogging` again to check the warning.

```
px4PrepareModelForMATFileLogging('px4demo_log')
```

The warning still appears, which implies that we have to optimize more.

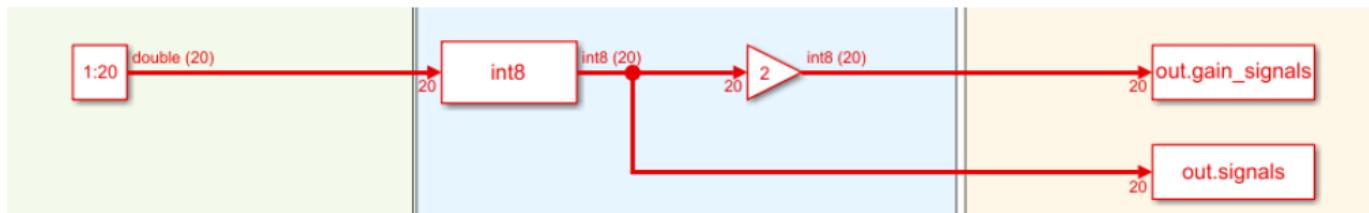
There are four *To Workspace* blocks executing at the rate of 10ms, and all of them have the Save format set to Structure With Time. This will make the 'Time' data for 10ms be logged by multiple blocks and it is redundant.

3. Set the Save Format of *To Workspace1*, *To Workspace2* and *To Workspace3* blocks to **Structure**. Let the **Save Format** of *To Workspace4* remain as **Structure with Time** as is. You can manually set them or execute the below commands:

```
set_param('px4demo_log/To Workspace1','SaveFormat','Structure')
set_param('px4demo_log/To Workspace2','SaveFormat','Structure')
set_param('px4demo_log/To Workspace3','SaveFormat','Structure')
```

4. Execute the function `px4PrepareModelForMATFileLogging` again. Always click on **Fix It** when the pop up appears. The warning can still be seen.

5. To reduce memory further, look at the data types. The constant block has values 1 to 20, but the data type is double. We can use the int8 data type for these values, which will reduce the memory required. Insert a Data Type Conversion block and set its data type to int8.



6. Execute the function `px4PrepareModelForMATFileLogging` again. Notice that there are no warnings.

You can also try other methods described in the “Troubleshooting Memory Limitations for MAT-file Logging” section such as decimation and changing the Sample Time of the signal selected for logging, to reduce memory.

Deploy the Simulink model to Pixhawk 1 Hardware

In the Simulink model, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. The build process for the model starts and it is deployed on the Pixhawk 1 Hardware board.

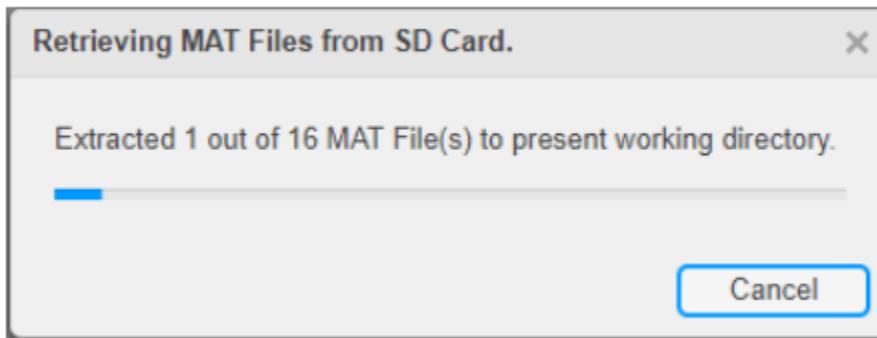
Step 4 - Import the MAT-files from the SD Card on Pixhawk 1 Hardware

In this section, we refer the “Import MAT-Files into MATLAB” section.

1. The Stop time for the Simulink model is set to 30s. After 30s, execute the `getMATFilesFromPixhawk` command to import the MAT-Files from the SD card to the present working directory of MATLAB.

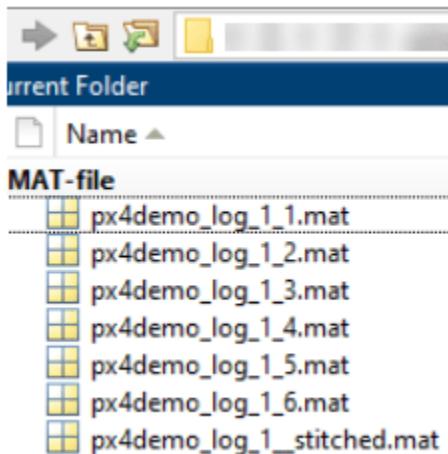
```
getMATFilesFromPixhawk('px4demo_log','DeleteAfterRetrieval',true)
```

A MATLAB window showing the status of the files being imported opens.



2. Once all the MAT-Files are imported into the present working directory, execute the px4MATFilestitcher command to stitch all the MAT-files together.

```
px4MATFilestitcher()
```



3. Load the data to the base workspace and analyze the data.

```
load('px4demo_log_1_stitched.mat')
```

For example, if you want to see the values of rt_AlgorithmOutput, you can execute the below command:

```
plot(rt_counter.time,rt_AlgorithmOutput.signals.values)
```

Related Topics

- “Log Signals on an SD Card”

Reading Accelerometer Values from an I2C based Sensor Connected to a PX4 Autopilot

This example shows how to use UAV Toolbox Support Package for PX4 Autopilot to configure and read accelerometer values from an I2C based sensor.

Introduction

UAV Toolbox Support Package for PX4 Autopilot enables you to use the I2C interface to communicate with I2C based devices connected to the PX4 Autopilots. In this example, you will learn how to communicate with sensor MPU9250. The MPU9250 is a 9 degree of freedom (DOF) inertial measurement unit (IMU) used to read acceleration, angular velocity, and magnetic field in all three dimensions. This sensor is interfaced with the PX4 Autopilot using the I2C bus. For more information on the device, refer to the MPU9250 datasheet.

This example shows how to program the PX4 Autopilot to read the accelerometer values from the sensor using the I2C bus. It also illustrates how to program the PX4 Autopilot to initialize the sensor with some configuration settings.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.

Required Hardware

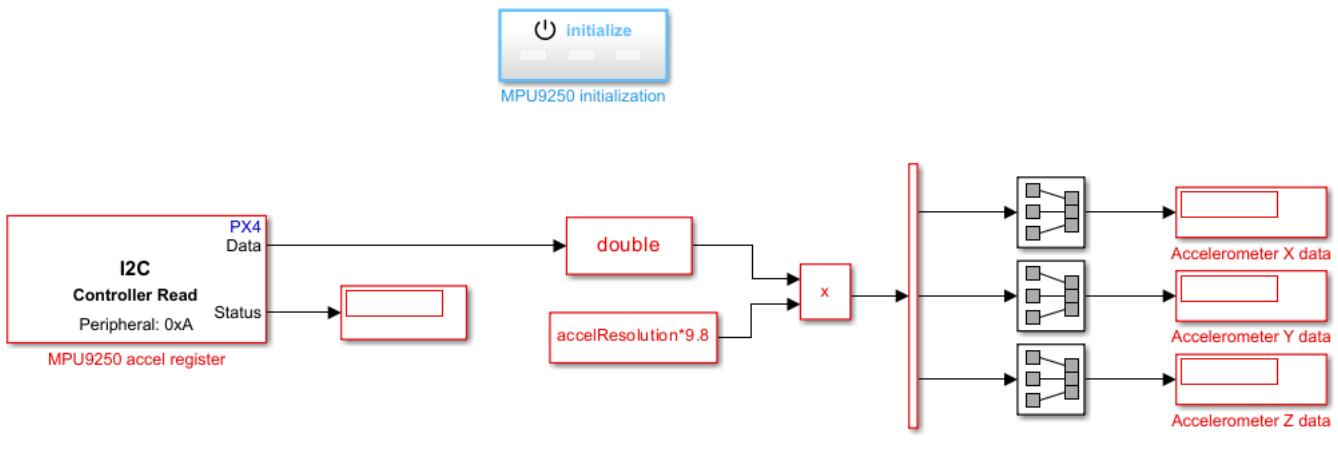
To run this example, you will need the following hardware:

- Supported PX4 Autopilot
- MPU9250
- USB cable
- Pixhawk connectors

Model

Open the px4demo_I2C_MP9250 model.

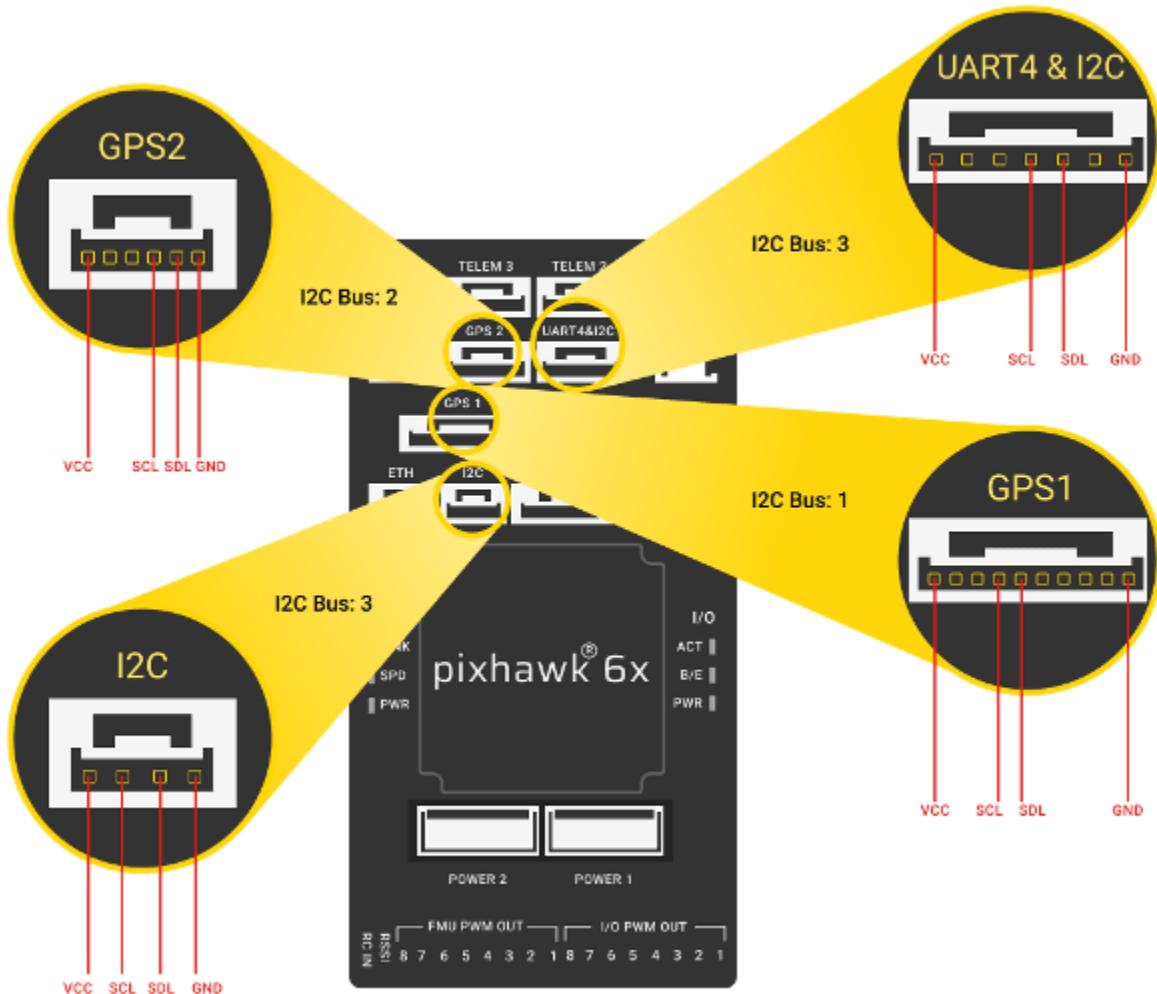
Read Accelerometer values from MPU9250 connected to PX4 Autopilot



Copyright 2020-2021 The MathWorks, Inc.

Step 1 - Connect the MPU9250 Sensor to the Pixhawk Hardware

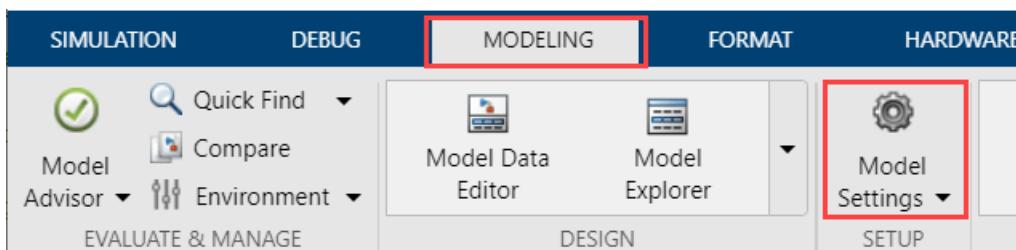
In this section, connect the MPU9250 sensor to the Pixhawk board. Decide the bus you want to connect to the I2C device. Refer to the “I2C Bus Port Numbers for Labels on PX4 Autopilots” on page 5-2 for finding buses and pinout for your board. For example, consider Pixhawk 6x as the hardware board, then the bus mapping will be similar to the following image. It is recommended to connect the I2C device to the I2C specific port, that is I2C. In this example, corresponding I2C bus number is 3. If you are using a different board, see “I2C Bus Port Numbers for Labels on PX4 Autopilots” on page 5-2 for buses and pinouts. Connect the sensor to corresponding pin of the I2C port. You might require a corresponding Pixhawk connector for that port. Refer to the following image for pinouts.



Step 2 - Configure the Model for Supported Pixhawk Hardware

In this section, you will configure the model for the supported Pixhawk hardware.

1. Open the px4demo_I2C_MP9250 model. The model is configured for Pixhawk 6x. If you have a different supported Pixhawk hardware follow the remaining steps to select the required hardware.
2. In your Simulink model, Open the **Modeling** tab and press **CTRL+E** to open Configuration Parameters dialog box.

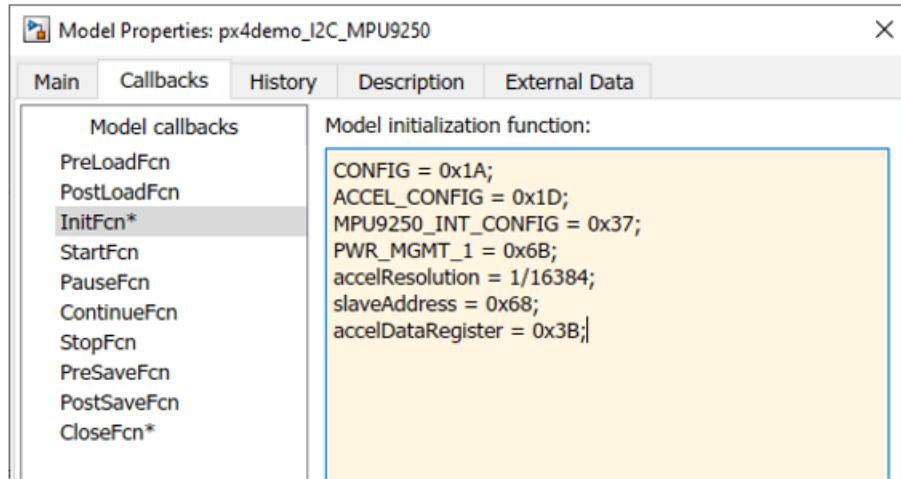


3. Select the **Hardware Implementation** pane and select your required Pixhawk hardware from the Hardware board parameter list. Do not change any other settings.

4. Click OK.

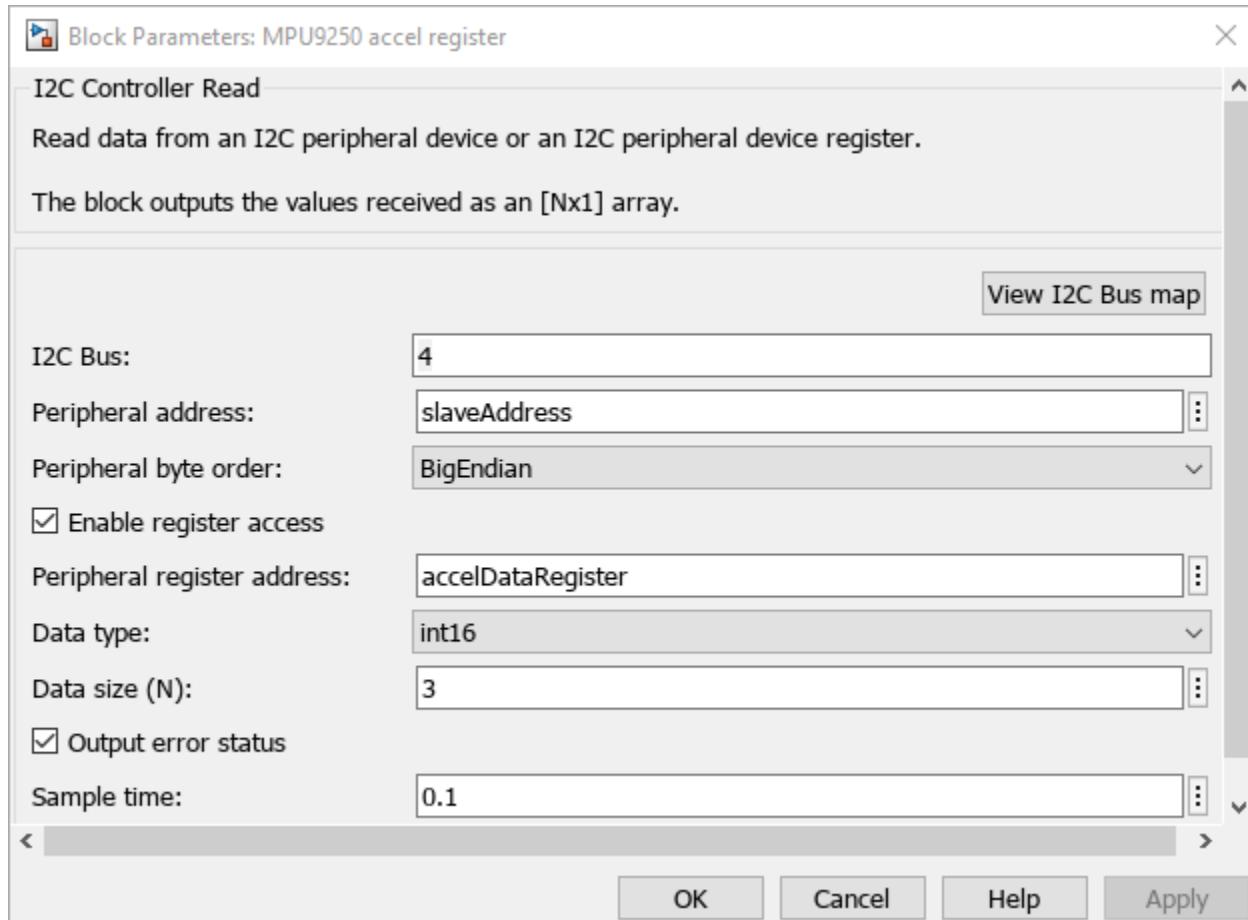
Step 3 - Configure the Model to Read Accelerometer Values Using the I2C Read Block

The configuration values and registers used in this example are defined in InitFcn callback present in the Model Properties of the example model. These values are chosen by referring to the MPU9250 datasheet.



1. Configure I2C Read block.

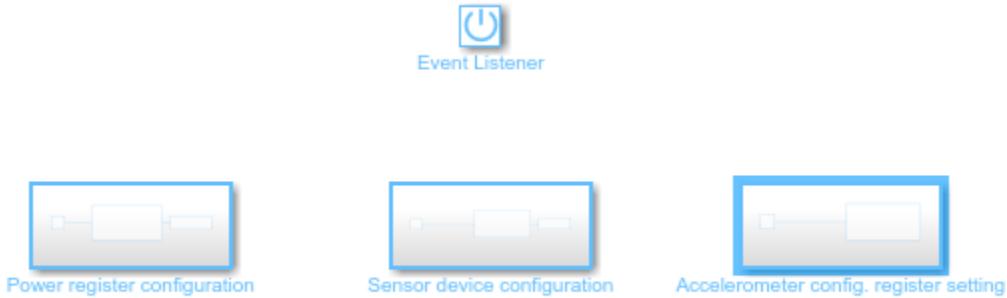
- Open the I2C Read block. Notice that the **Peripheral register address** parameter of the block is set to `accelDataRegister`. This corresponds to a 7-bit address of 1101000 (0x68 in hexadecimal) according to the MPU9250 datasheet.



- Accel values along x, y, and z are stored in 3 consecutive 16 bit registers starting from 0x3B. Enter the **Peripheral register address** as `accelDataRegister` (0x3B). Select the **Data type** as `int16`, **Peripheral byte order** as `BigEndian`, set **Data size (N)** to 3, and **Sample Time** to `0.1`. This initiates an I2C read request every 0.1 second.
- Click **OK**.

2. Sensor Initialization

- The Initialization block is used for initializing sensor configuration registers. This is a onetime operation, and initialization block can be used to do such one-time configurations.
- In the MPU9250 initialization block there are three different subsystems power register configuration, Sensor device configuration, accelerometer register configuration.



- In Power register configuration, PWR_MGMT_1 register of MPU9250 is configured using I2C Controller Write block.
- In Sensor device configuration, MPU9250_INT_CONFIG register of MPU9250 is configured using I2C Controller Write block.
- In Accelerometer register configuration, ACCEL_CONFIG register of MPU9250 is configured using I2C Controller Write block.

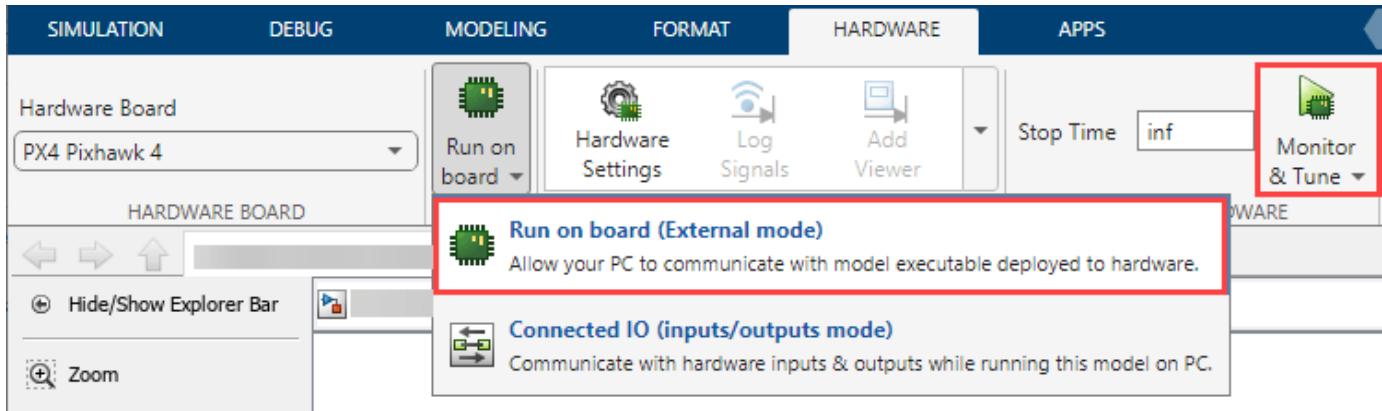
3. Notice the following points in the model:

- The Data Type Conversion block is used to convert the read data to double for multiplying with the accelerometer resolution(`accelResolution`).
- The `accelresolution` is chosen assuming the accelerometer range is 2g and is, multiplied using a product block.
- The Demultiplexer block used to extract acceleration along 3 different axes into three different signals from a 3X1 vector.

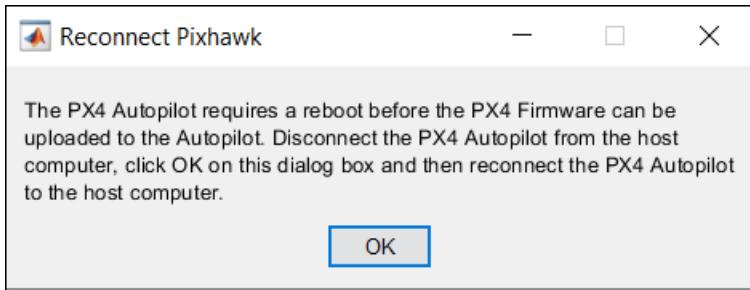
Step 4 - Configure and Run the Model

In this section, you will monitor the accelerometer values by signal monitoring and parameter tuning. You can also run the model using the Connected IO option.

1. In the **Simulation** tab, specify the stop time for parameter tuning simulation. The default value for the **Stop time** parameter is 10.0 seconds. To run the model for an indefinite period, enter **inf**.
2. Run the model using one of the following options.
 - **Monitor & Tune:** To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

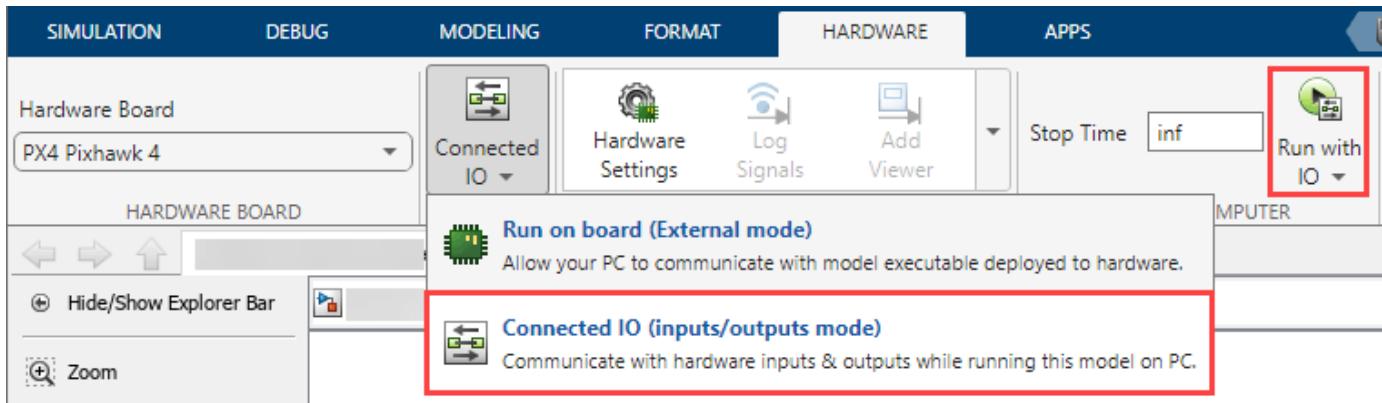


Wait for the code generation to be completed. When the dialog box appears instructing you to reconnect the flight controller to the serial port, click **OK** and then reconnect the PX4 Autopilot on the host computer.



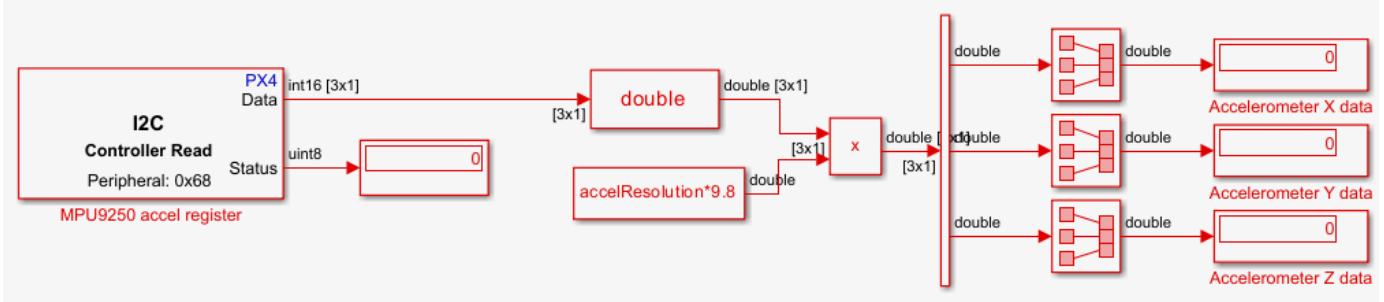
The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

- **Connected IO:** To run this model in the Connected I/O mode, on the **Hardware** tab, in the Mode section, select Connected IO and then click Run with IO.



If the Connected IO firmware is not deployed on the hardware, then the dialog box instructing you to reconnect the flight controller to the serial port appears otherwise the dialog box is not displayed. If the dialog box appears, click **OK** and then reconnect the PX4 Autopilot on the host computer.

3. Notice that the Display blocks Accelerometer X data, Accelerometer Y data, Accelerometer Z data, in the model shows the accelerometer reading in m/s² along x, y, and z direction respectively.



Note: If you are not able to read data, the display block connected to the Status port can be used to understand the read status.

0 - indicates a success

1 - indicates a failure in opening the bus.

2 - indicates a failure in read operation, which may be due to a loose connection between sensor and the PX4 Autopilot.

4. Try moving the sensor and observe the changes in values.

5. In the **Hardware** tab, click **Stop** to stop the process.

Other Things to Try

- Read values from gyroscope and magnetometer.
- In this example configuration for accelerometer range and accelerometer bandwidth are not set, so the default settings of the sensor are used. Try configuring the sensor for a different accelerometer range and accelerometer bandwidth.
- You can try this example to read accelerometer values from an I2C based sensor in Connected I/O simulation. For more information on enabling and using Connected I/O simulation, see “Getting Started with Connected IO for PX4 Autopilot” on page 1-199.

Summary

This example showed how to program your PX4 Autopilot to configure and read accelerometer values from an I2C based sensor. In this example you also learned how to:

- Communicate with an I2C based sensor.
- Program the PX4 Autopilot to write data to and read data from specific registers on the I2C based sensor.
- Initialize the sensor with configuration settings.
- You can use this example as a reference to access other I2C based sensors.

Getting Started with Connected IO for PX4 Host Target

This example shows you how to enable and use Connected IO simulation to read data from PX4 Host Target during normal mode simulation.

Introduction

The UAV Toolbox Support Package for PX4® Autopilots enables you to communicate with the PX4 Host Target during normal mode simulation by enabling Connected IO. Connected IO is not based on Simulink® code generation, hence Embedded Coder® license is not required to use Connected IO simulation. For more information and list of supported blocks see “Communicate with Hardware Using Connected I/O”.

In this example, you will:

- Enable Connected IO simulation
- Observe the accelerometer and GPS data from PX4 Host Target by running the model in Normal mode simulation

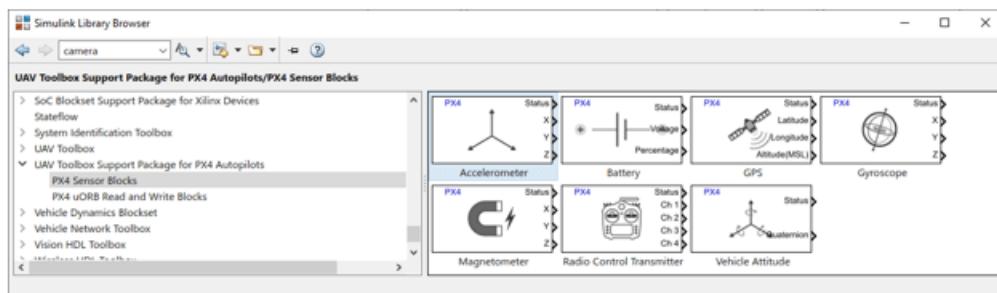
Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using the Hardware Setup screens. In the Hardware Setup screens, ensure that you select **PX4 Host Target** for **PX4 Autopilot board** option.

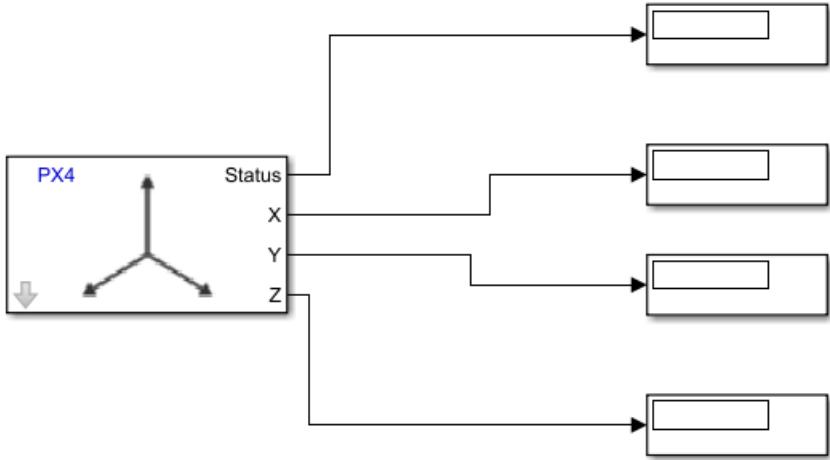
Task 1 - Configure and Run a Model to Read Accelerometer Data Using Accelerometer Block

In this task, you will configure a Simulink model to read accelerometer sensor data from PX4 Host Target using an Accelerometer block. You will also observe the data over Connected IO simulation.

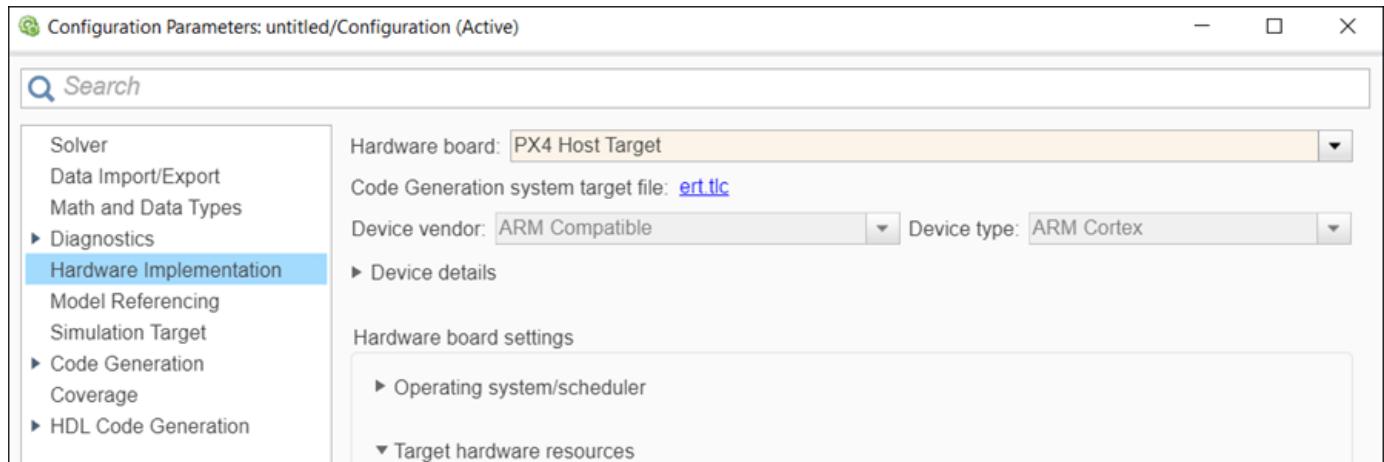
1. From the MATLAB toolbar, select **Home > New > Simulink Model** to open the Simulink Start Page. Click **Blank Model** to launch a new Simulink model.
2. On the **Simulation** tab, click **Library Browser**. In the Library Browser, select **UAV Toolbox Support Package for PX4 Autopilots > PX4 Sensor Blocks**, and then add an Accelerometer block to the model.



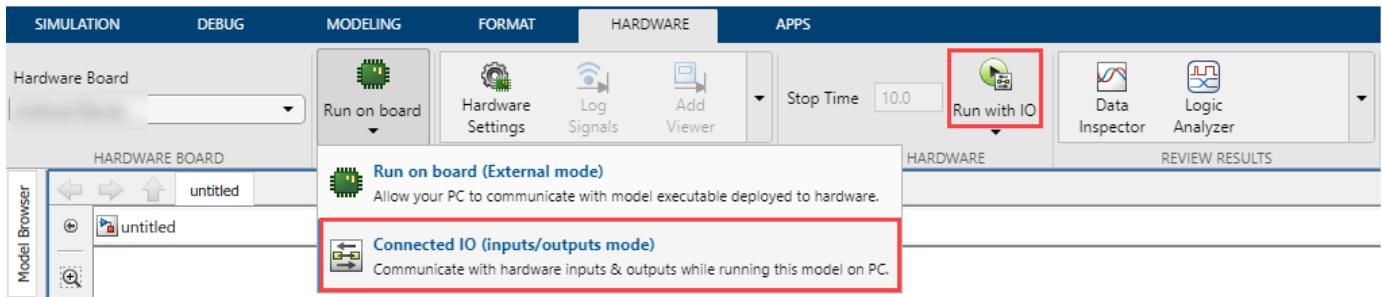
3. In the Library Browser, Select **Simulink >**, then drag and drop four Display blocks to the model. Connect the three outputs of the Accelerometer block to the three Display blocks. Connect the Status output to the fourth Display block as shown in the image below.



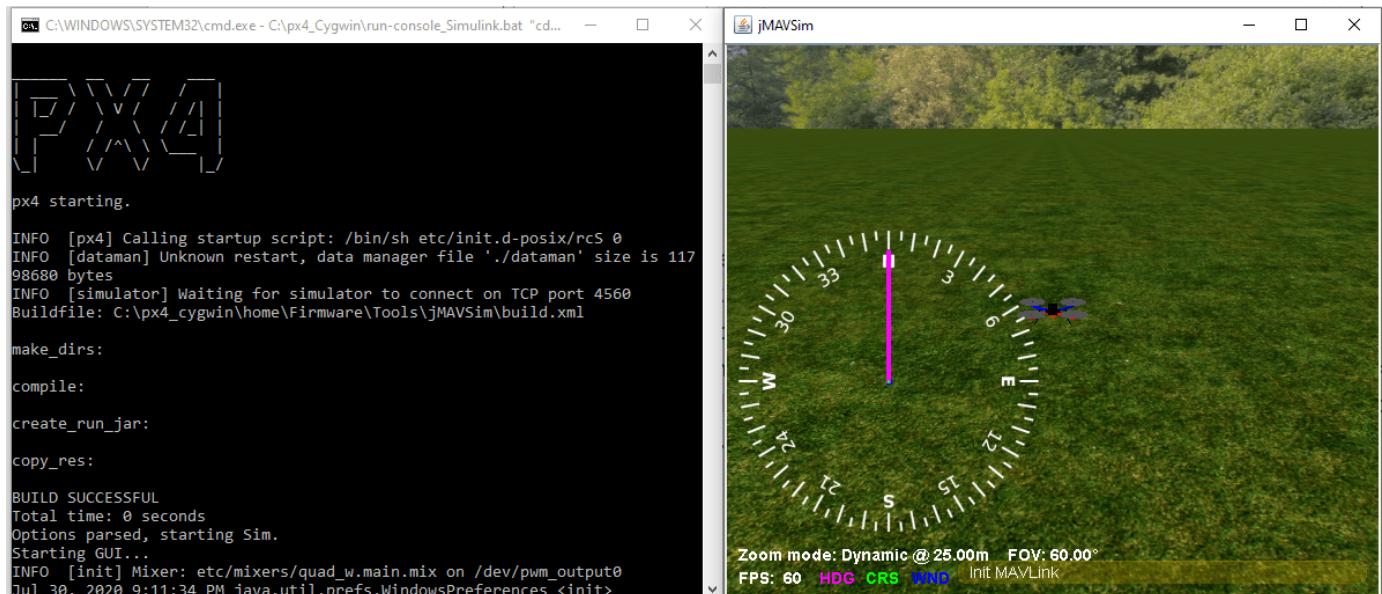
4. In the **Modeling** tab, click **Model Settings** to open the **Configuration Parameters** dialog box.
5. In the **Configuration Parameters** dialog box, select **Hardware Implementation**.
 - Verify that the **Hardware board** parameter is set to **PX4 Host Target**.



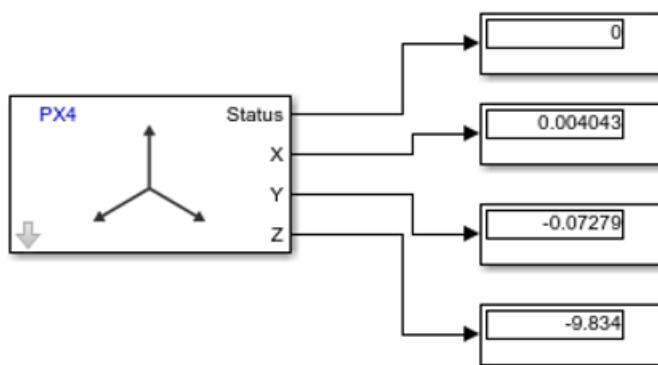
6. On the **Modeling** tab, set the **StopTime** to **inf**.
7. To run this model in the Connected IO mode, on the **Hardware** tab, in the Mode section, select **Connected IO** and then click **Run with IO**.



- Ensure the PX4 Host target and jMAVSIM Simulator launches.



- Once the Simulation begins, you can view the values in the display. Z accel = -9.8 (acceleration value due to gravity. This is because the drone is on the ground and in horizontal position).

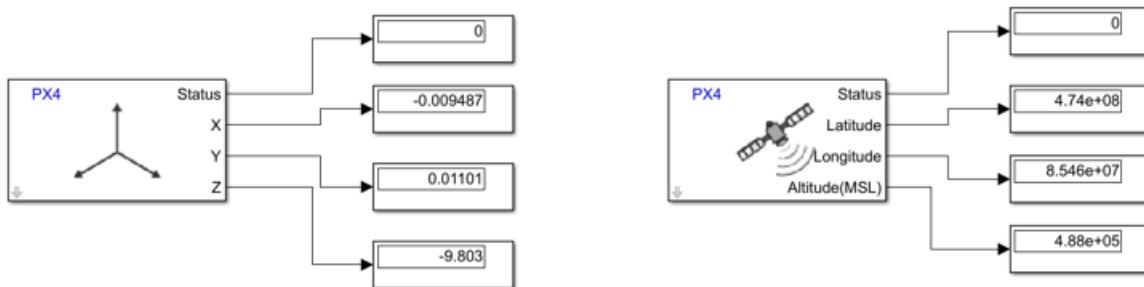


8. Stop the simulation after observing values. Do not close the PX4 Host Target and jMAVSIM windows.

Task 2 - Add PX4 GPS Block to the Configured Model to Read GPS Values

Since Connected IO is not based on Simulink code generation, the PX4 Host Target launched during Task 1 is independent of the model. If you add or delete blocks or modify the model, Simulink connects to the already launched PX4 Host Target and reads data for updated model. For example, add PX4 GPS block to the earlier configured model in task 1 and run the simulation again.

1. On the **Simulation** tab, click **Library Browser**. In the Library Browser, Select **UAV Toolbox Support Package for PX4 Autopilots > PX4 Sensor Blocks**, then add a GPS block to the model.
2. In the Library Browser, Select **Simulink** and then drag and drop four Display blocks to the model. Connect the three outputs of the GPS block to the three Display blocks. Connect the Status output to the fourth Display block as shown in the image below.

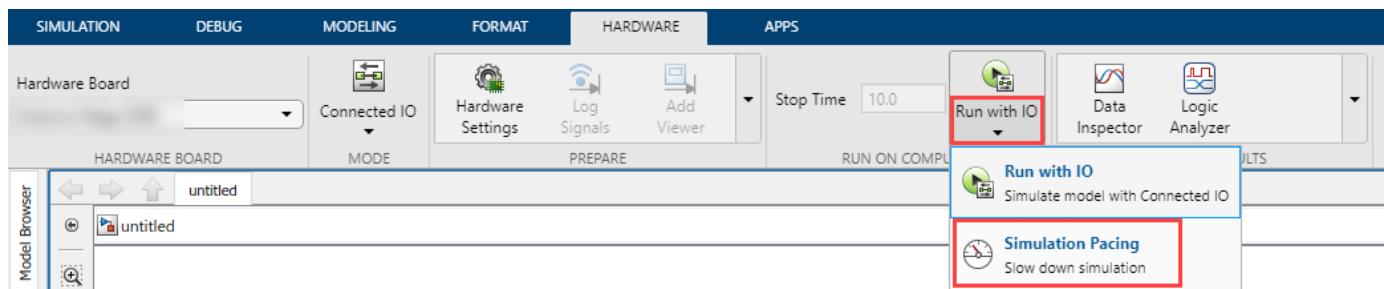


- The GPS values displayed are of Zurich, Switzerland. These GPS values are the default values provided in GPS simulator module of PX4.

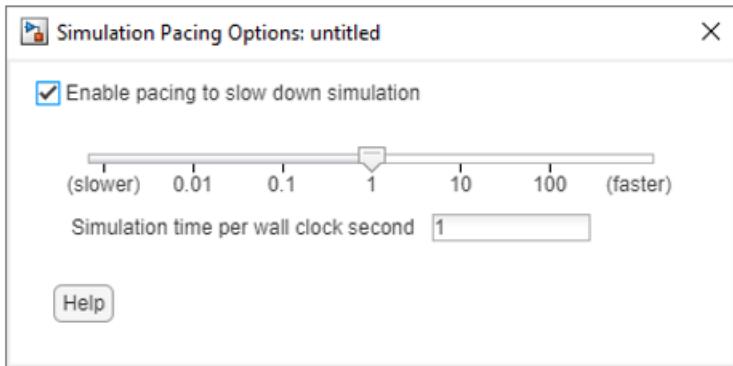
Task 3 - Run Connected IO Simulation Close to Real-Time Using Simulation Pacing Option

In tasks one and two, Simulink runs slower than the wall clock. This section helps you to run the Connected IO simulation close to real-time using Simulation Pacing option. Perform these steps to run the model using Simulation Pacing option.

1. On the **Hardware** tab, select **Simulation Pacing**.



2. In the **Simulation Pacing** Options dialog, select **Enable pacing to slow down simulation**. On enabling, the specified pace gets automatically applied to the simulation. Set the value for **Simulation time per wall clock second** as 1.



3. Along with enabling Simulink pacer on and setting the Simulation time per wall clock second as 1, sample time for the block has to be chosen specifically for getting close to real-time simulation. There is a two-way communication between Simulink and the Host Target while running Connected IO simulation. This introduces a inherent time overhead for a particular block execution. Thus, there is a specific minimum time required for a block execution. This varies with the machine performance and can be around 15-20 milliseconds. Now the sample time of these blocks should be more than this block execution time for close to real-time connected IO simulation. Assuming the block Execution time is 20 milliseconds for block, since we have two blocks here, the minimum sample time of the model we can keep to get close to real-time behavior is $20*2 = 40$ milliseconds. Set the sample time of the blocks as 40 milliseconds.

4. Simulate the model. Observe that Simulink time is close to wall clock.

- The host target does not launch again. Simulink connects to the existing host target launched previously and reads the data.

Other Things to Try

You can use preconfigured Simulink models that helps you to read and write to uORB topics. You can run Connected IO simulation in the preconfigured models. For more information on uORB blocks, see “Getting Started with uORB Blocks for PX4 Autopilots Support Package” on page 1-80.

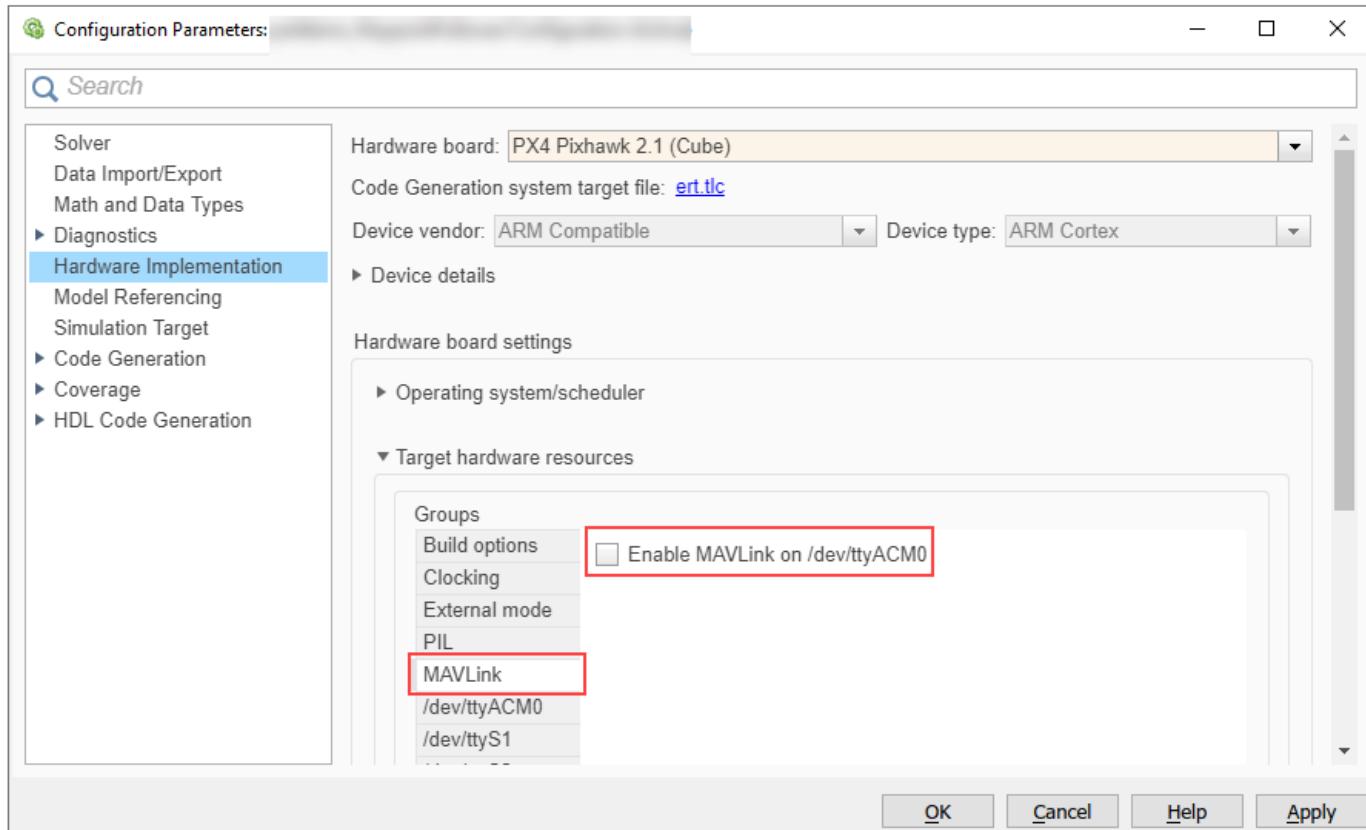
Connected IO simulation is not suitable for designing closed loop controllers and systems with dynamic feedback, where blocks must run at much higher rate. Hence it is recommended not to try the controller examples with Connected IO.

Connecting to NSH Terminal for Debugging

This topic shows you how to use the NSH terminal for the debugging of Pixhawk Series controllers.

After you deploy the model created using UAV Toolbox Support Package for PX4® Autopilots, you can use the NSH terminal for the debugging of Pixhawk Series controllers.

NSH is accessed using MAVLink. Therefore, ensure that MAVLink is enabled over USB (for details, see “Enabling MAVLink in PX4 Over USB”.



Note: If you enable MAVLink over USB (/dev/ttyACM0), then you cannot use the External mode over USB. You can choose any other serial port (for example, /dev/ttyS6) to run the Simulink model in External mode. However, in this case, you may need additional serial to USB convertor.

Accessing NSH from MATLAB

UAV Toolbox Support Package for PX4 Autopilots provides a pre-defined class named `HelperPX4` that helps you to access NSH and perform certain actions.

Find the serial port on the host computer to which the PX4 Autopilot is currently connected and create a `HelperPX4` object to access NSH. For example:

```
shellObj = HelperPX4('COM9');
```

Here COM9 value is used as an example. Change it to the actual host serial port.

The `HelperPX4` class provides different methods to send NSH commands and obtain response.

The `system` method helps you to send the commands to NSH and get the shell response. To see the list of available NSH commands under `system` method, use the `help` command:

```
[shellResp, status] = shellObj.system('help')
```

The status value 1 indicates a successful execution.

If the `listener` command is listed, you can use it in the `system` method to listen to any uORB topic. For example:

```
[shellResp, status] = shellObj.system('listener vehicle_status')
```

The `getFile` method helps you to get the fault log or any other file in the SD card mounted on the PX4 Autopilot target, without removing SD card from the target. To do this (for fault logs):

1. See the list of fault logs available in the SD card using the `system` method:

```
shellObj.system('ls /fs/microsd')
```

2. Observe the log list and decide which log file you want to copy to the host computer. Get the file to host computer using the `getFile` method. For example:

```
shellObj.getFile('/fs/microsd/fault_2019_10_24_10_49_04.log');
```

Tip: You can use the `getFile` method to edit the startup script `rc.txt` that you have copied to the SD card (for details about `rc.txt`, see “Performing PX4 System Startup from SD Card”. In this case, you can also use the `putFile` method to copy the modified startup script back to the SD card.

After you make all the changes, you can delete the `HelperPX4` object that you created:

```
delete(shellObj);
```

Accessing NSH using QGroundControl (QGC)

For accessing NSH using QGroundControl (QGC), refer to this link

Monitor and Tune PX4 Host Target Flight Controller with Simulink-Based Plant Model

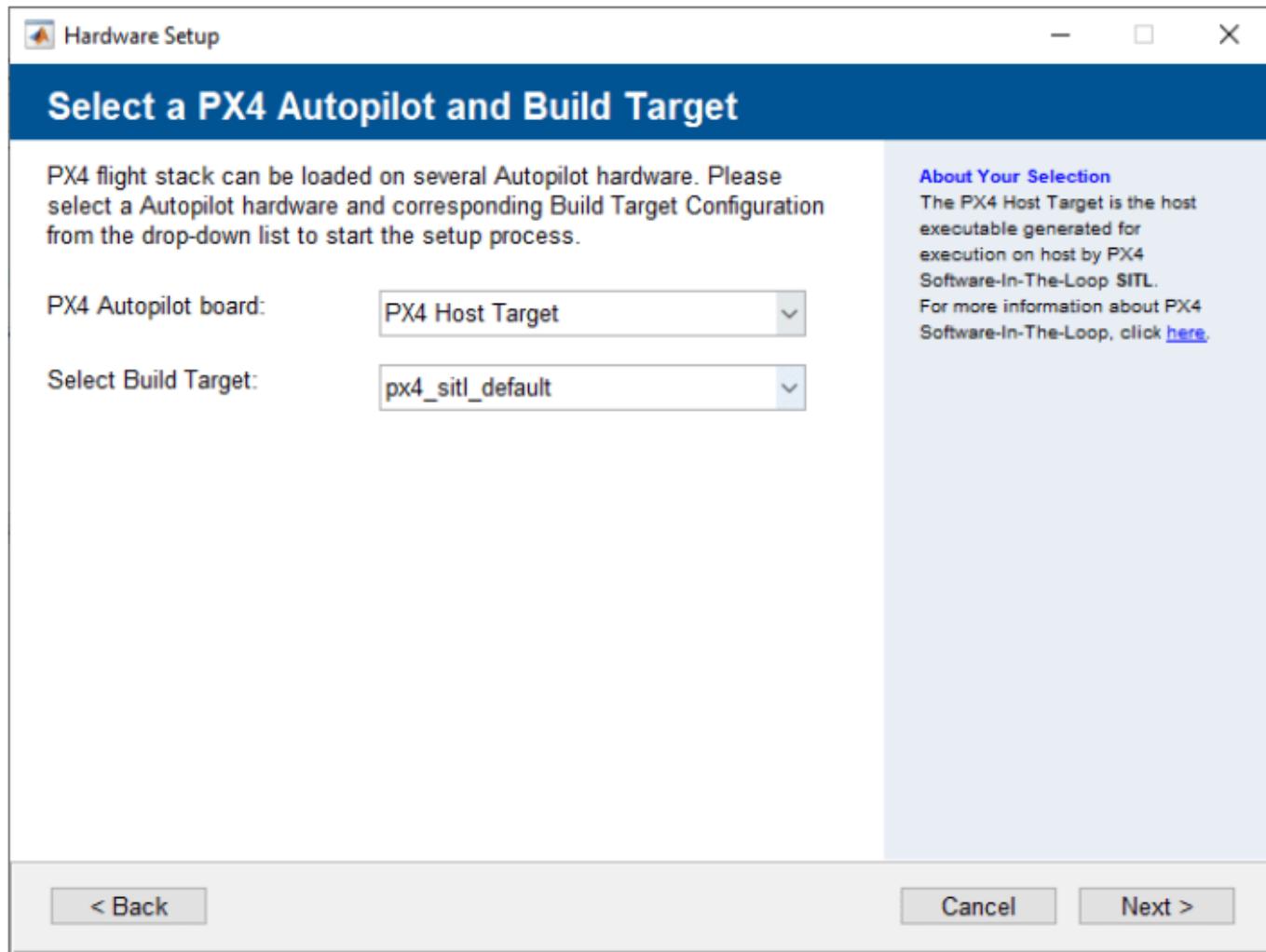
This example shows how to use the UAV Toolbox Support Package for PX4 Autopilots to verify the controller design using PX4 Host Target versus the simulator designed in Simulink®.

The UAV Toolbox Support Package for PX4 Autopilots enables you to use Simulink to design a flight controller algorithm to stabilize the vehicle based on the current vehicle attitude, position, and velocity and also track the desired attitude using Simulink. The MAVlink blocks in the UAV Toolbox enable you to read and write the MAVLink HIL_* messages and design the plant dynamics.

This example shows how to validate a position controller design on a medium-sized quadrotor plant using a single Simulink model, and then take the same controller and plant model and simulate it with the PX4 source code in, what the PX4 community calls it, Software In The Loop (SITL) simulation.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens. In the **Select a PX4 Autopilot and Build Target** screen, select **PX4 Host Target** as the PX4 Autopilot board from the drop-down list.



- For more information on how to verify the controller design using the jMAVSIM simulator, see “Deployment and Verification Using PX4 Host Target and jMAVSIM/Simulink” and “Position Tracking for X-Configuration Quadcopter Using Rate Controller” on page 1-160.
- For more information on designing the controller model and verifying it using the simulator plant model designed in Simulink, see “Integrate Simulator Plant Model Containing MAVLink Blocks with Flight Controller Running on PX4 Host Target”.

Model

To get started, launch the Simulink Px4DemoHostTargetWithSimulinkPlant project.

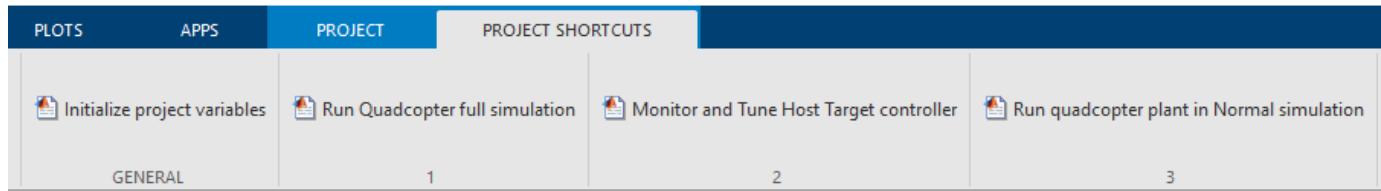
Model Architecture and Conventions

This project consists of the following models:

- The `Quadcopter_controller` is the harness model that contains the flight controller. The model is set up to run with the PX4 source code by creating a PX4 Host Target executable.

- The Quad_UAV_dynamics harness model contains the UAV plant dynamics. This harness model is set up to run in normal simulation pacing mode in lockstep with the Quadcopter_controller harness model.

The Project shortcuts guide you through the three tasks as you progress through the example.

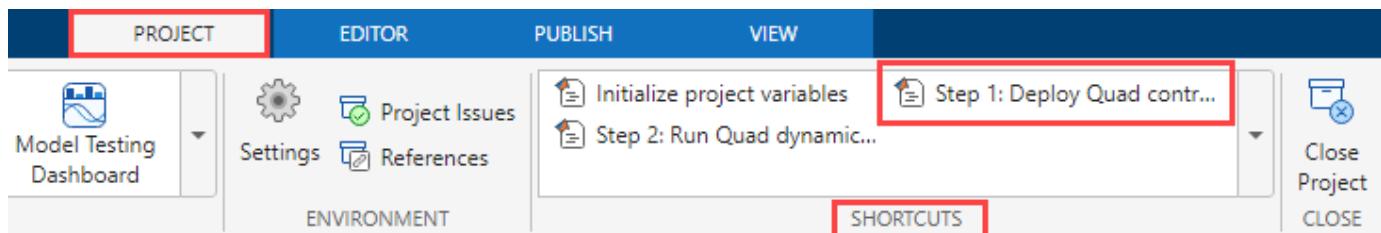


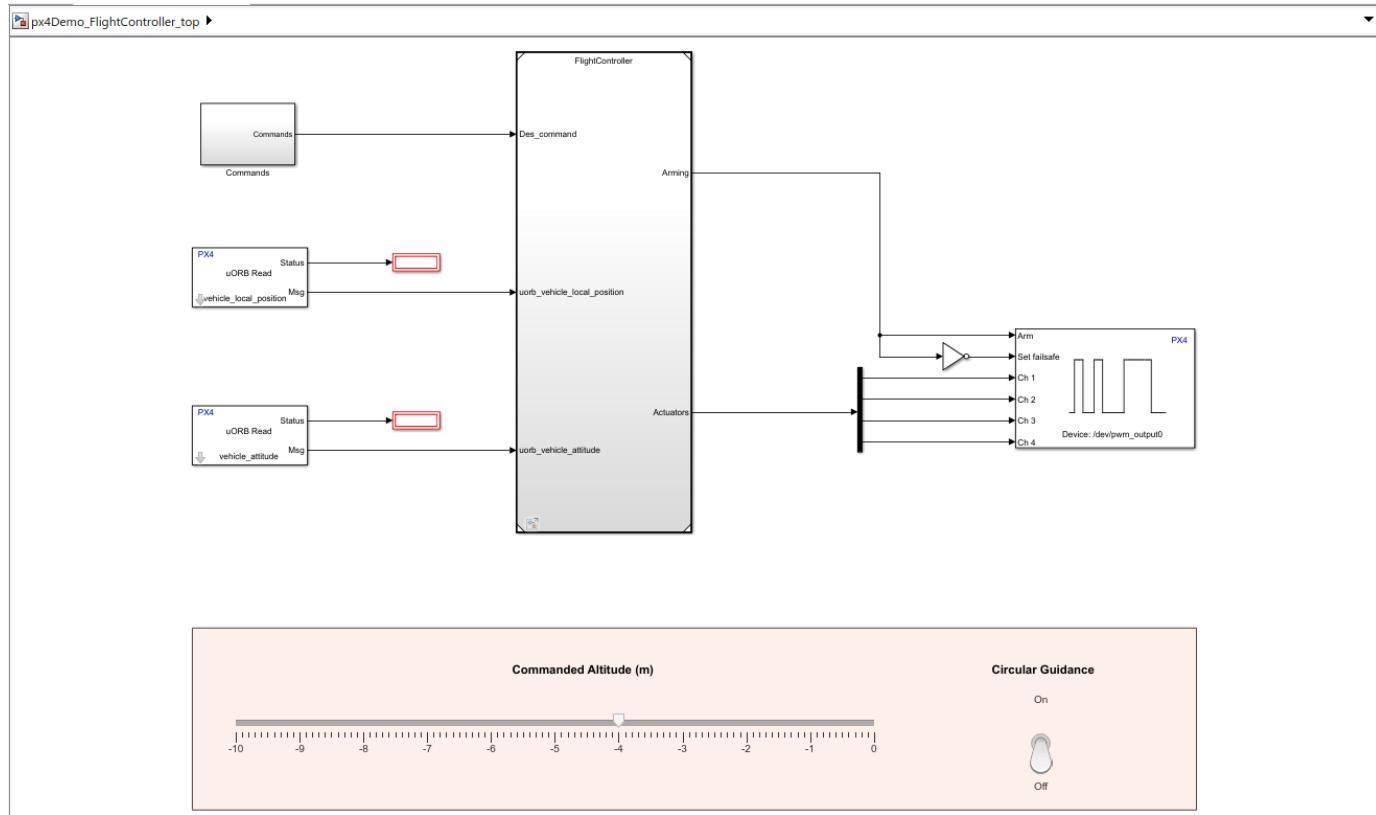
Task 1: Deploy Controller as Host Target and Run Plant Model in Simulink

In this task, we use the controller designed in Task 1 and run it with PX4 source code using the PX4 Host Target feature (for more information, see “Integrate Simulator Plant Model Containing MAVLink Blocks with Flight Controller Running on PX4 Host Target”). Use two separate instances of MATLAB to run controller and plant model.

Step 1: Deploy Quad controller on Host Target

- Click **Step 1: Deploy Quad controller on Host Target** in the Project Shortcuts tab to open the | Quadcopter_controller | model.



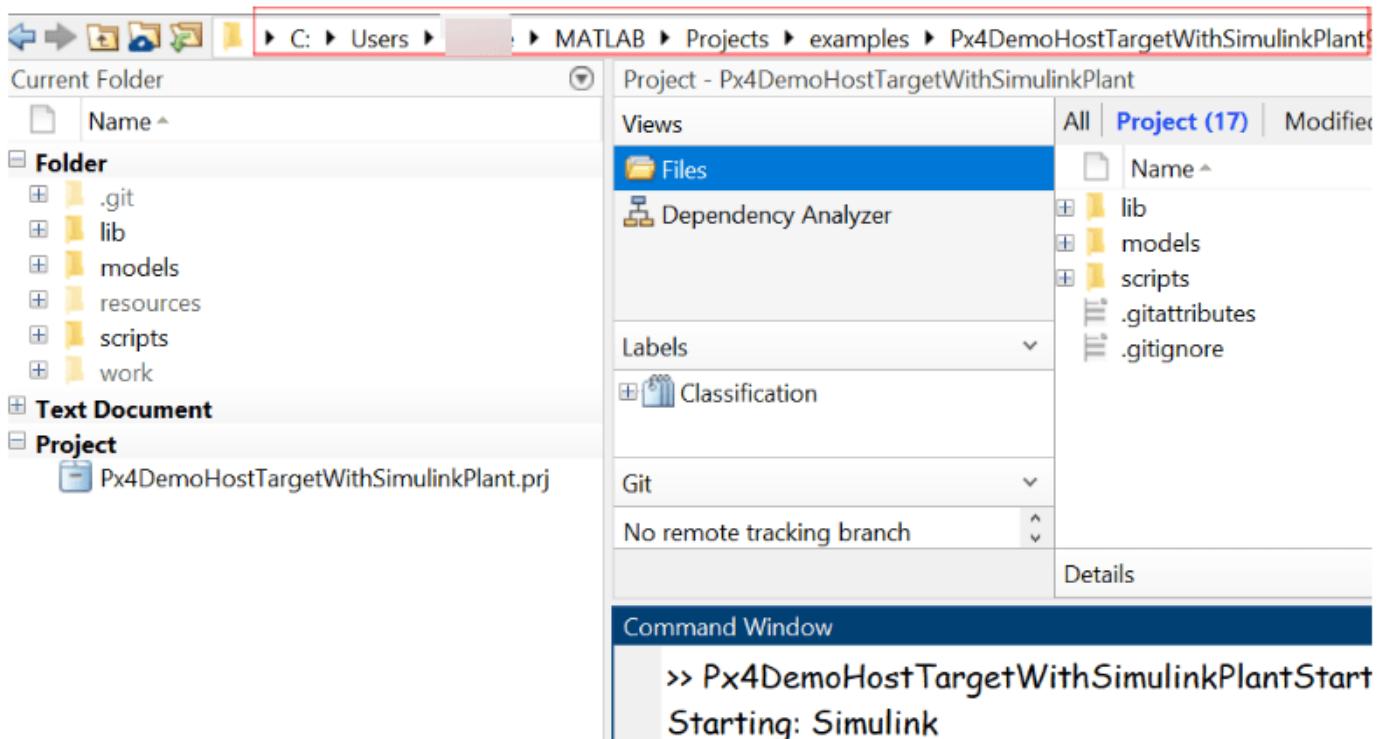


The uORB Read blocks are used to subscribe to the `vehicle_local_position` and `vehicle_attitude` topics. These topics contain data sent by the simulator plant model using `HIL_STATE_QUATERNION` and `HIL_GPS` messages.

The Controller subsystem designs the Rate controller and the Position and Attitude controller as explained in “Position Tracking for X-Configuration Quadcopter Using Rate Controller” on page 1-160.

The `FlightController` model outputs the actuator values that are then fed to the PX4 PWM Output block.

2. Copy the MATLAB Project Path to clipboard.

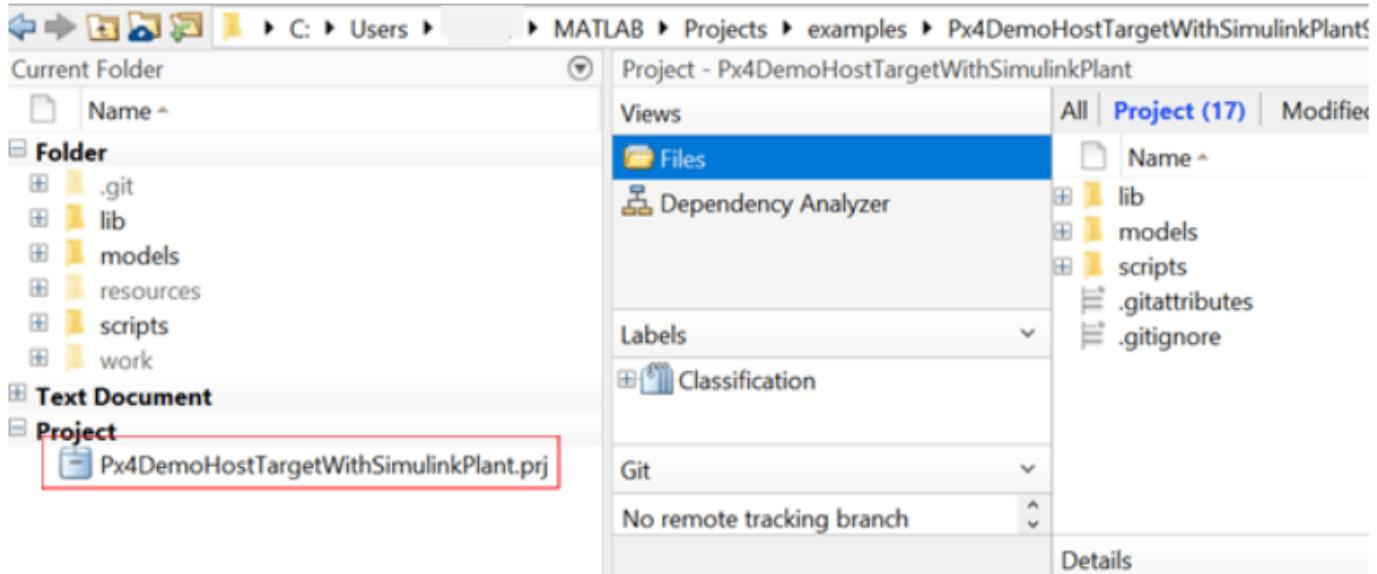


Step 2: Launch Simulink Plant model in second MATLAB

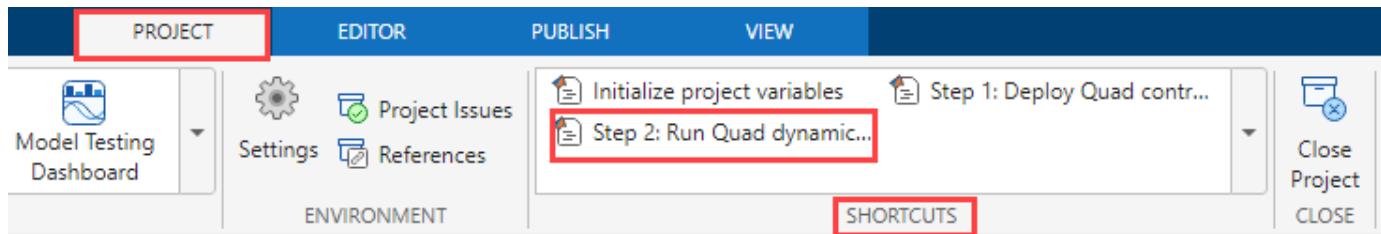
1. Open the second instance of the same MATLAB version.
2. Navigate to the project path previously copied in current MATLAB.

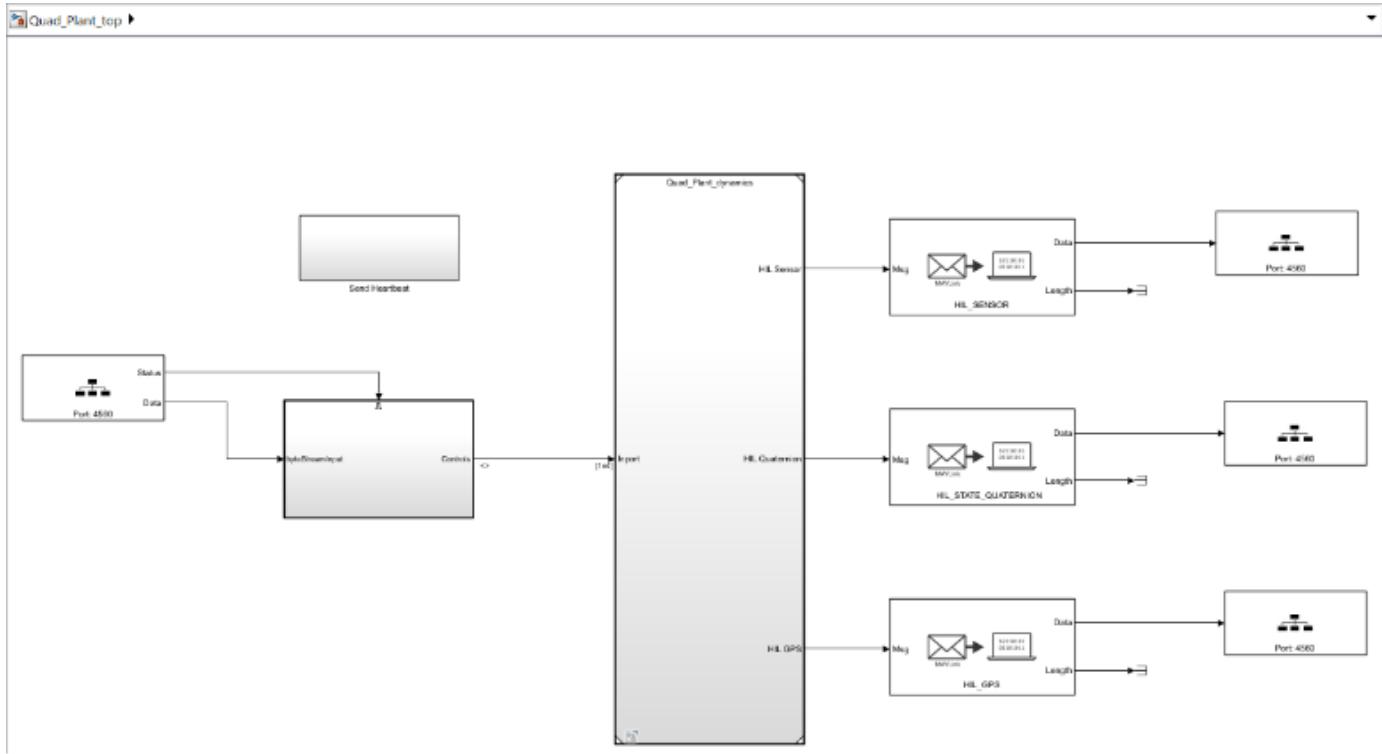
```
fx >> cd C:\Users\ \MATLAB\Projects\examples\Px4DemoHostTargetWithSimulinkPlant
```

3. Click on the .prj file to launch the same Project in current MATLAB.



4. Click **Step 2: Run Quad dynamics simulation** in the Project Shortcuts tab to open the Quad_UAV_dynamics model.

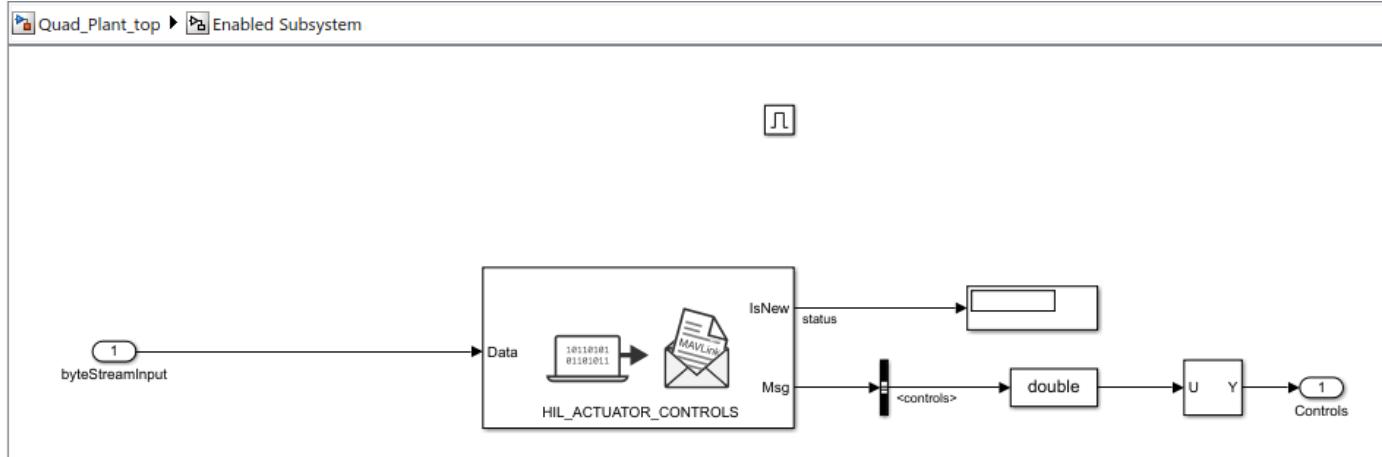




Ensure that the **Simulation Pacing** option for this model is enabled, as described in “Integrate Simulator Plant Model Containing MAVLink Blocks with Flight Controller Running on PX4 Host Target”.

The `TCP read from PX4 Host Target` MATLAB System block is used to read the MAVLink data sent from the `px4Demo_FlightController_top` model.

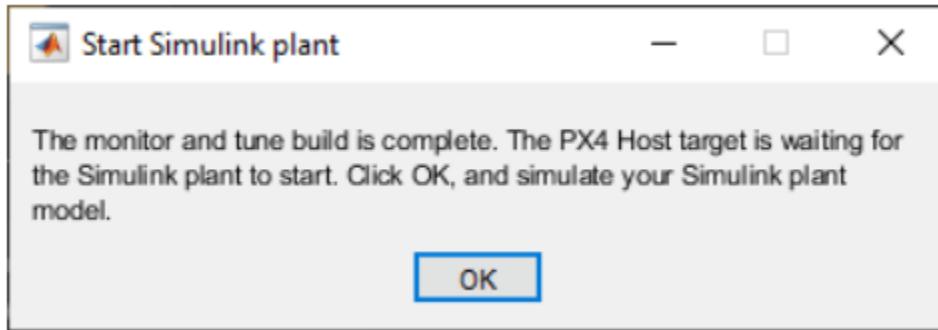
The Enabled subsystem has the `MAVLink Deserializer` block that extracts the `HIL_ACTUATOR_CONTROLS` message.



The TCP write to PX4 Host Target MATLAB System block sends the HIL_SENSOR, HIL_GPS, and HIL_STATE_QUATERNION MAVLink messages to the px4Demo_FlightController_top model.

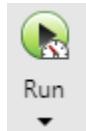
Step 3: Deploy controller model over Monitor & Tune and run Plant Simulation

1. In Configuration parameters > Hardware Implementation, set the **Hardware board** parameter to **PX4 Host Target**.
2. Under **Target hardware resources > Build Options**, set **Simulator** to **Simulink**.
3. In the **Simulation** tab, set the Simulation **Stop time** to *inf*.
4. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.
5. Wait for Simulink to complete the code generation. The following dialog box appears. **Do not click OK yet**.



In the plant model launched in second MATLAB, follow the below steps.

1. In the **Simulation** tab, set the Simulation **Stop Time** to *inf*.
2. Click **Run** on the **Simulation** Tab.



After the simulation starts in the plant model, click **OK** in the dialog box of the first model.

Getting Started with Connected IO for PX4 Autopilot

This example shows you how to enable and use Connected IO simulation to read data from PX4 Autopilot during normal mode simulation.

Introduction

The UAV Toolbox Support Package for PX4 Autopilots enables you to communicate with the PX4 Autopilot during normal mode simulation by enabling Connected IO. Connected IO is not based on Simulink code generation, hence Embedded coder license is not required to use Connected IO simulation. For more information and list of supported blocks see “Communicate with Hardware Using Connected I/O”.

In this example, you will:

- Enable Connected IO simulation
- Observe the accelerometer and Read Parameter data from PX4 Autopilot by running the model in Normal mode simulation

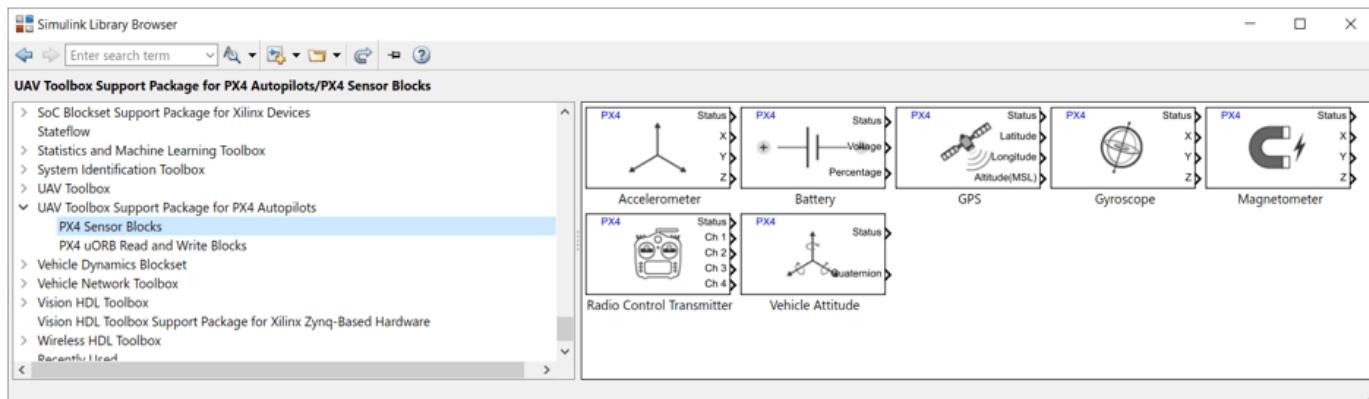
Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the “Install UAV Toolbox Support Package for PX4 Autopilots in Linux” using the Hardware Setup screens.

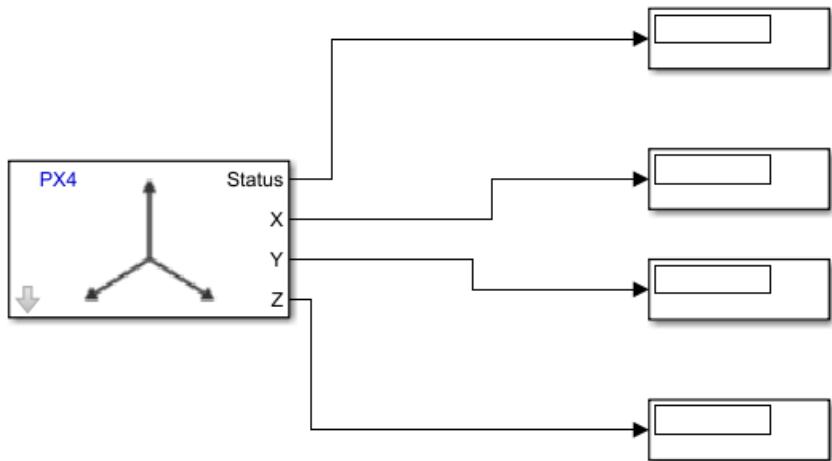
Task 1 - Configure and Run a Model to Read Accelerometer Data Using Accelerometer Block

In this task, you will configure a Simulink model to read accelerometer sensor data from PX4 Autopilot using an Accelerometer block. You will also observe the data over Connected IO simulation.

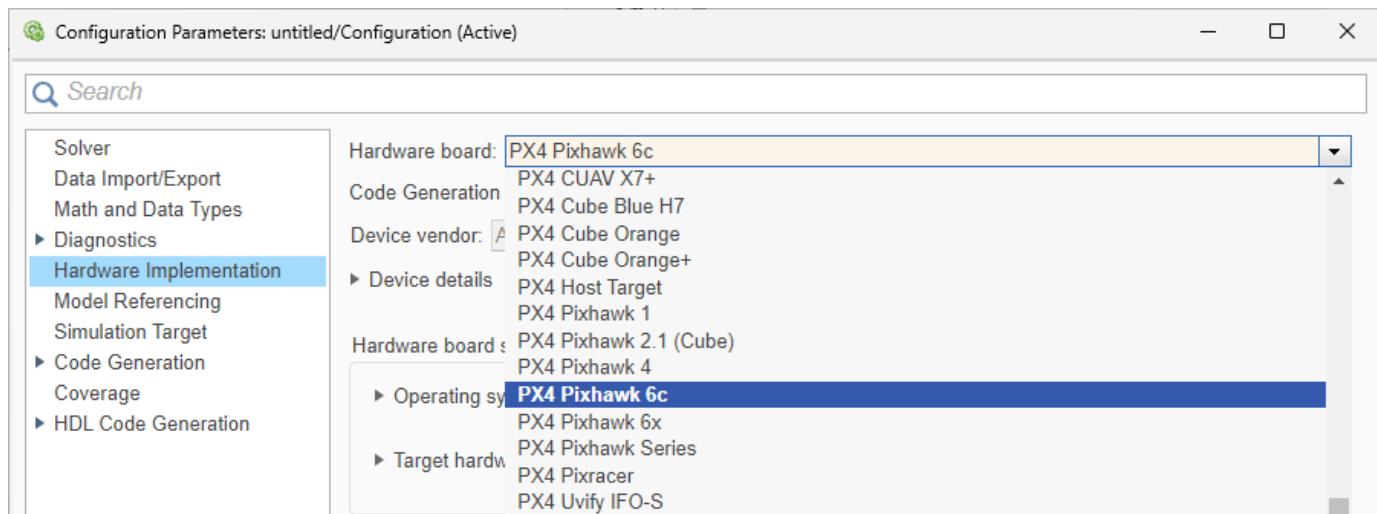
1. From the MATLAB toolbar, select **Home > New > Simulink Model** to open the Simulink Start Page. Click **Blank Model** to launch a new Simulink model.
2. On the **Simulation** tab, click **Library Browser**. In the Library Browser, select **UAV Toolbox Support Package for PX4 Autopilots > PX4 Sensor Blocks**, and then add an Accelerometer block to the model.



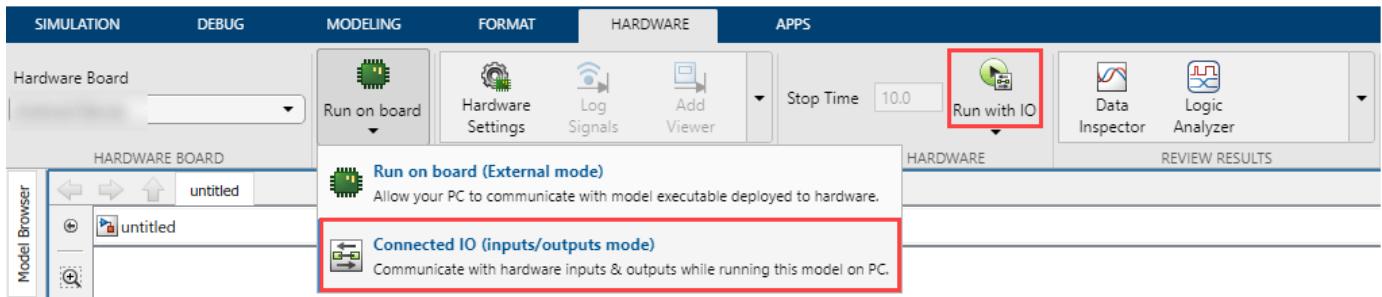
3. In the Library Browser, Select **Simulink >**, then drag and drop four Display blocks to the model. Connect the three outputs of the Accelerometer block to the three Display blocks. Connect the Status output to the fourth Display block as shown in the image below.



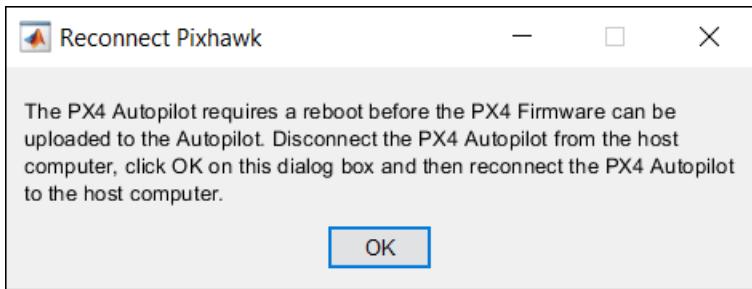
4. On the **Modeling** tab, click **Model Settings** to open the **Configuration Parameters** dialog box.
5. In the **Configuration Parameters** dialog box, select **Hardware Implementation**.
- Verify that the **Hardware board** parameter is set to a required PX4 board.



6. On the **Modeling** tab, set the **StopTime** to **inf**.
7. To run this model in the Connected IO mode, on the **Hardware** tab, in the Mode section, select **Connected IO** and then click **Run with IO**.

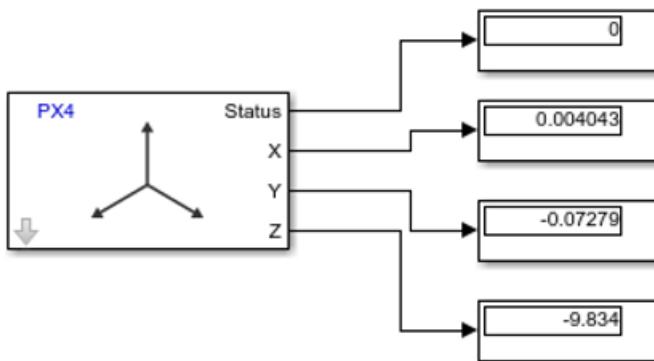


- Wait for the code generation to be completed. If the dialog box appears instructing you to reconnect the flight controller to the serial port, ensure that you click **OK** on the dialog box within **5 seconds** after reconnecting the flight controller. This dialog box will not appear if the connected IO server is flashed onto the hardware.



After the Pixhawk series controller is flashed, wait for the signal monitoring to start.

- Once the Simulation begins, you can view the values in the display.



8. Stop the simulation after observing values. Do not close the PX4 Autopilot and jMAVSim windows.

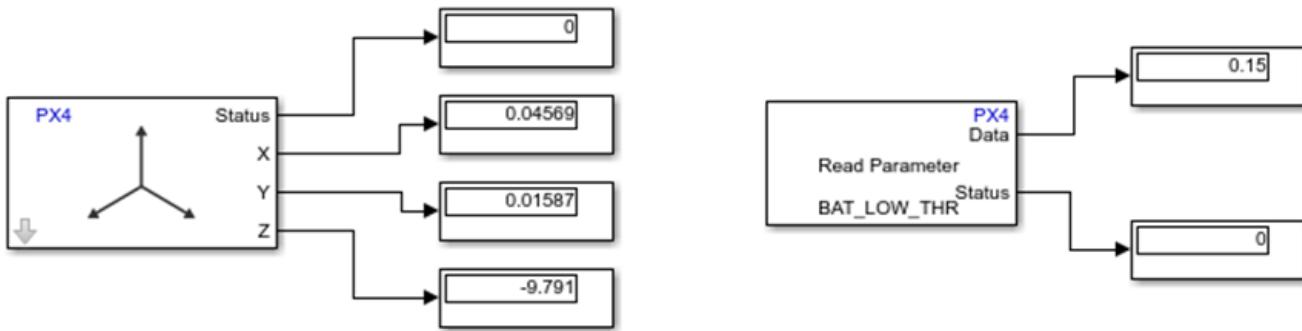
Task 2 - Add PX4 Read Parameter Block to the Configured Model to View Read Parameter Values

Since Connected IO is not based on Simulink code generation, the PX4 Autopilot connected during Task 1 is independent of the model. If you add or delete blocks or modify the model, Simulink connects to the already connected PX4 Autopilot and reads data for updated model. For example, add PX4 Read Parameter block to the earlier configured model in task 1 and run the simulation again.

1. On the **Simulation** tab, click **Library Browser**. In the Library Browser, Select **UAV Toolbox Support Package for PX4 Autopilots** and then add a Read Parameter block to the model.

2. Double-click the Read Parameter block to open the Block Parameters dialog box. Change the **Parameter Name** to BAT_LOW_THR and then click **Apply** and **OK**.

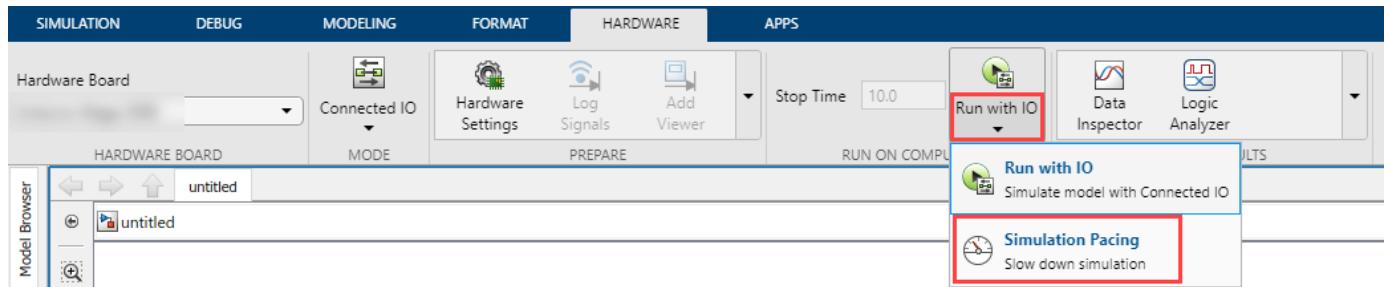
3. In the Library Browser, Select **Simulink** and then drag and drop two Display blocks to the model. Connect the outputs of the Read Parameter block to Display blocks as shown in the image below.



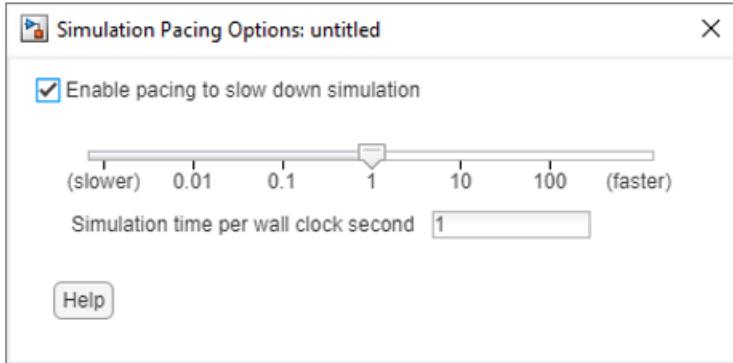
Task 3 - Run Connected IO Simulation Close to Real-Time Using Simulation Pacing Option

In tasks one and two, Simulink runs slower than the wall clock. This section helps you to run the Connected IO simulation close to real-time using Simulation Pacing option. Perform these steps to run the model using Simulation Pacing option.

1. On the **Hardware** tab, select **Simulation Pacing**.



2. In the **Simulation Pacing** Options dialog, select **Enable pacing to slow down simulation**. On enabling, the specified pace gets automatically applied to the simulation. Set the value for **Simulation time per wall clock second** as 1.



3. Along with enabling Simulink pacer on and setting the Simulation time per wall clock second as 1, sample time for the block has to be chosen specifically for getting close to real-time simulation. There is a two-way communication between Simulink and the Autopilot while running Connected IO simulation. This introduces a inherent time overhead for a particular block execution. Thus, there is a specific minimum time required for a block execution. This varies with the machine performance and can be around 20-30 milliseconds. Now the sample time of these blocks should be more than this block execution time for close to real-time connected IO simulation. Assuming the block Execution time is 30 milliseconds for block, since we have two blocks here, the minimum sample time of the model we can keep to get close to real-time behavior is $30*2 = 60$ milliseconds. Set the sample time of the blocks as 60 milliseconds.

4. Simulate the model. Observe that Simulink time is close to wall clock.

- Simulink connects to the PX4 Autopilot and reads the data.

Other Things to Try

You can use preconfigured Simulink models that helps you to read and write to uORB topics. You can run Connected IO simulation in the preconfigured models. For more information on uORB blocks, see “Getting Started with uORB Blocks for PX4 Autopilots Support Package” on page 1-80.

Connected IO simulation is not suitable for designing closed loop controllers and systems with dynamic feedback, where blocks must run at much higher rate. Hence it is recommended not to try the controller examples with Connected IO.

Code Execution Profiling on PX4 Target in Monitor & Tune Simulation

This example shows how to use the UAV Toolbox Support Package for PX4 Autopilots to profile the real-time execution of the generated code running as an executable on a PX4 target hardware with XCP on Serial Interface.

Introduction

Sample times you specify in the Simulink models determine the time schedule for running generated code on target hardware. With enough computing power on the hardware, the code runs in real-time according to the specified sample times. With real-time execution profiling, you can check if the generated code meets your real-time performance requirements. This support package supports the code execution profiling on any supported PX4 hardware.

At the end of the Simulink model code profile execution, you can:

- View a report of code execution times.
- Access and analyze execution time profiling data.

Prerequisite

- If you are new to Simulink, watch the Simulink Quick Start video.

Required MathWorks Products

- Simulink®
- Embedded Coder®

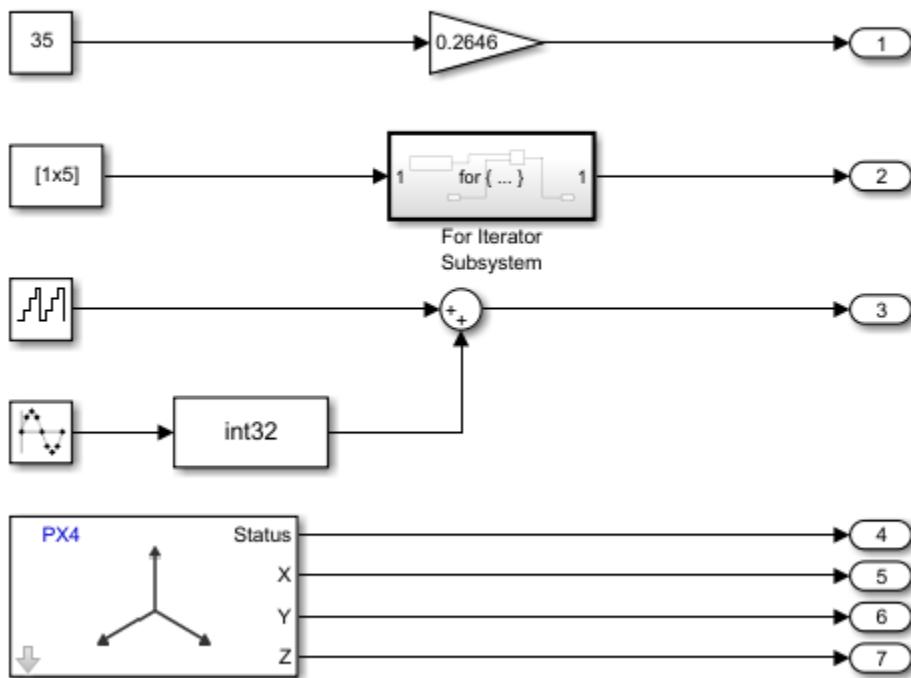
Required Hardware

One of these “Supported PX4 Autopilots”.

Model

Open the pre-configured px4demo_profiling model.

Profiling with XCP External Mode



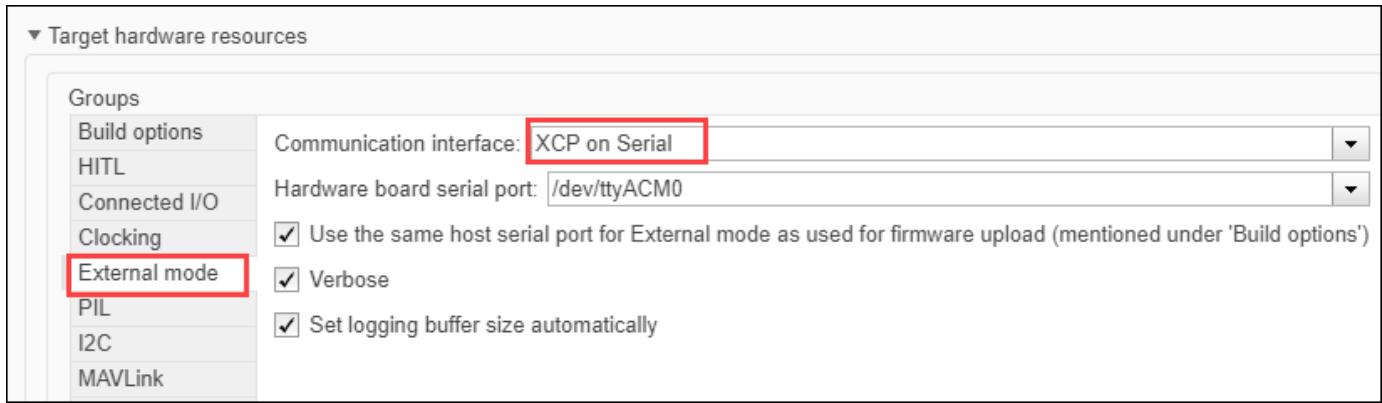
Copyright 2021 The MathWorks, Inc.

Configuring the Model

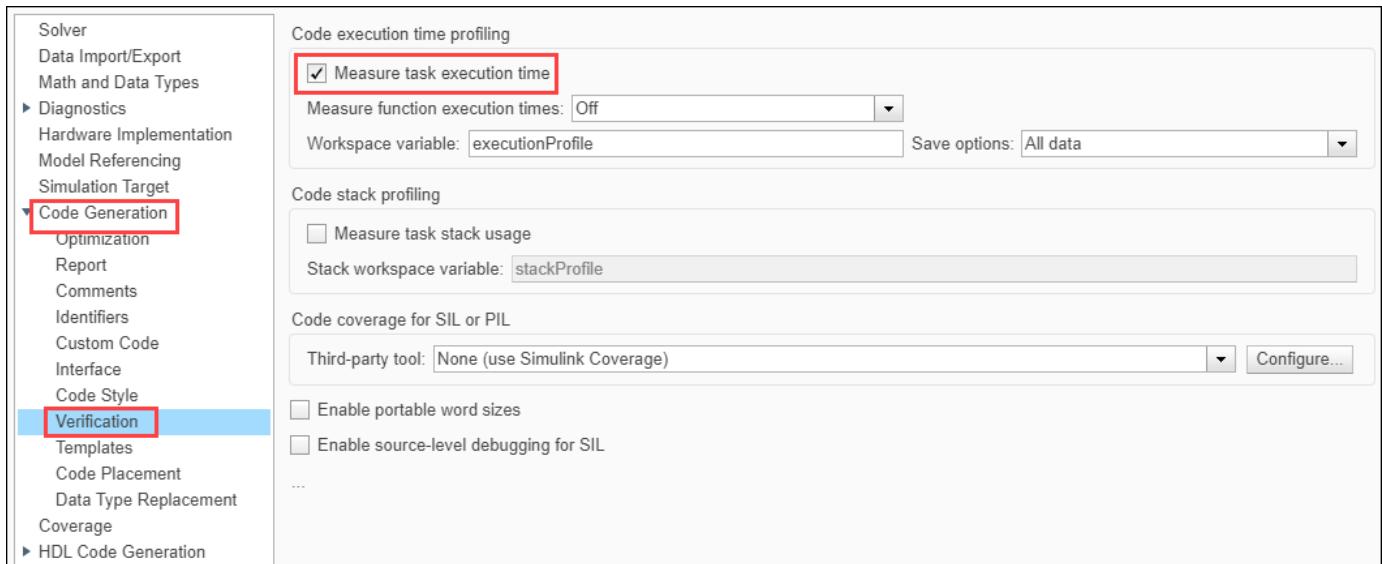
In this example, you will configure a Simulink model and enable profiling.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

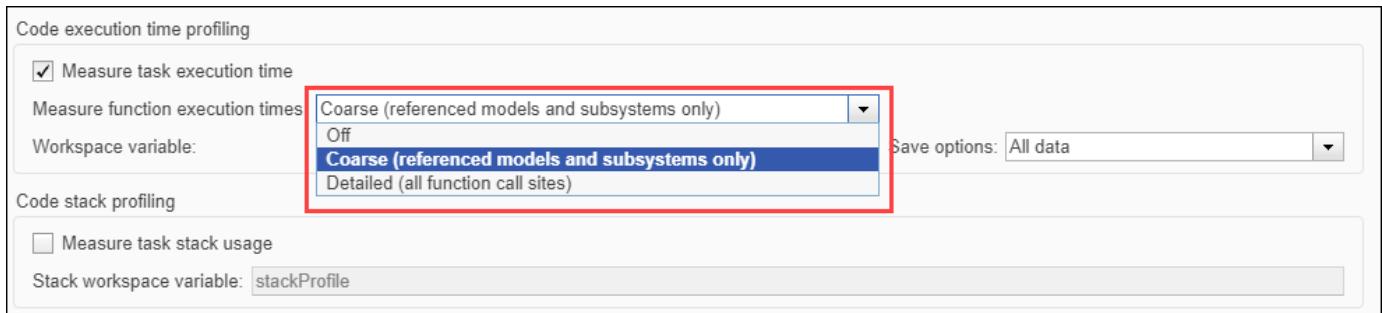
1. Open the model.
2. Go to **Modeling > Model Settings** to open the Configuration Parameters dialog box.
3. Open the **Hardware Implementation** pane, and select the required PX4 hardware from the list in **Hardware board** parameter.
4. Expand **Target hardware resources** for that board.
5. Go to **External mode** tab and choose **XCP on Serial** as the **Communication interface**.

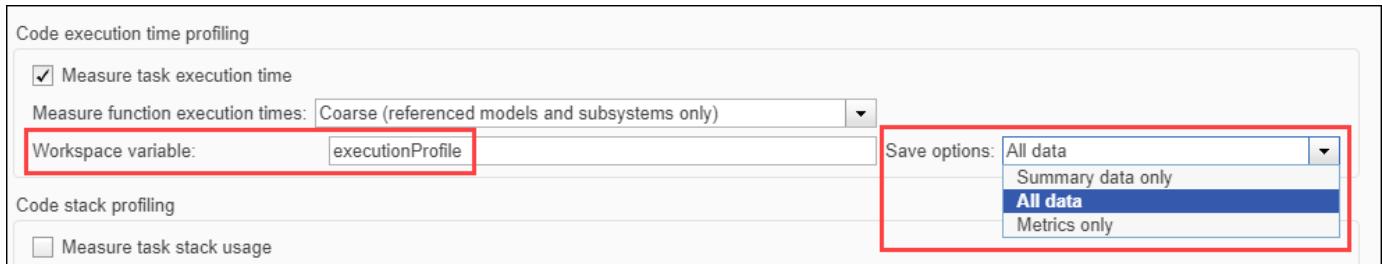


6. Go to **Code Generation > Verification > Code execution time profiling** and select **Measure task execution time**.



7. Select the required option for **Measure function execution times**, **Workspace Variable** and **Save Options**. For information on different save options, see "Save Options".





8. Click **Apply** and **OK**.

Initiate Monitor and Tune Action for the Model

On the **Hardware** tab of the Simulink toolbar, click **Monitor & Tune** to monitor signals and tune parameters.

This generates a profiling report with profiling metrics of different tasks/functions that are being profiled. This also contains links to plot the data on SDI, bar charts, pie charts and CPU Utilization which provides further analysis of the data.

All data report

Code Execution Profiling Report

Find: Match Case

Code Execution Profiling Report for px4demo_profiling

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See [Code Execution Profiling](#) for more information.

1. Summary

Total time	1087144000
Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	1e+06
Profiling data created	30-Nov-2021 13:25:50

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
px4demo_profiling_step0 [0.001 0]	30000	20982	30000	20982	39305	
px4demo_profiling_step1 [0.01 0]	67000	49214	67000	49214	3930	
px4demo_profiling_step2 [0.025 0]	62000	25638	62000	25638	1572	
px4demo_profiling_step3 [0.05 0]	94000	36531	94000	36531	786	
px4demo_profiling_terminate	17000	17000	17000	17000	1	

3. CPU Utilization [hide]

Task	Average CPU Utilization	Maximum CPU Utilization
px4demo_profiling_step0 [0.001 0]	2.098%	3%
px4demo_profiling_step1 [0.01 0]	0.4921%	0.67%
px4demo_profiling_step2 [0.025 0]	0.1026%	0.248%
px4demo_profiling_step3 [0.05 0]	0.07306%	0.188%
Overall CPU Utilization	2.766%	4.106%

4. Definitions

CPU Utilization: Percentage of CPU time assigned to a task. Computed by dividing task execution time by sample time.

Execution Time: Time between start and end of code section.

Self Time: Execution time, excluding time in child sections.

Code Execution Profiling Report Details

This report provides the following details:

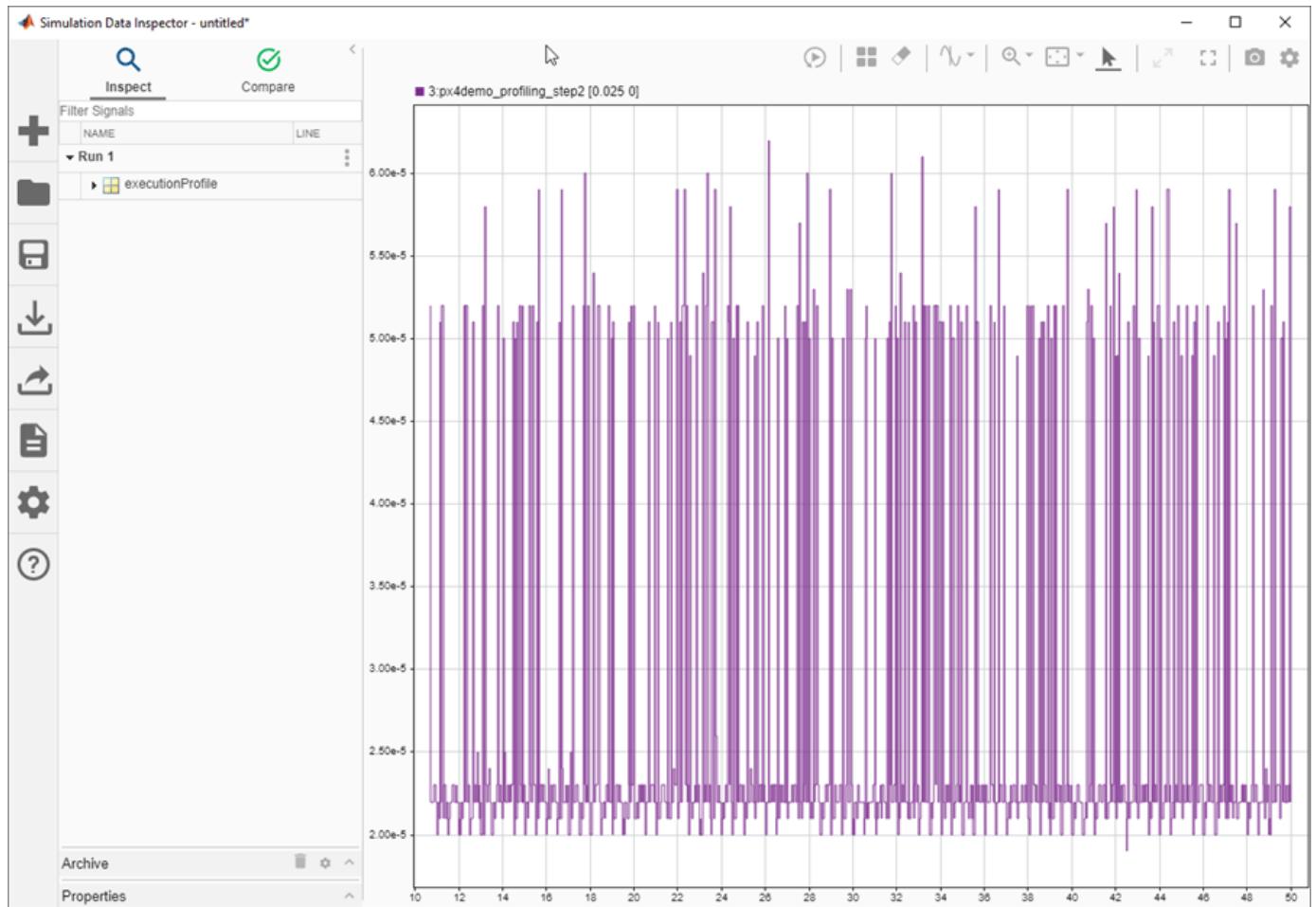
1. A detailed summary
2. Information about profiled code sections, which includes time measurements.
3. Average and maximum CPU core utilization.

Note: CPU Utilization in profiling report is computed by dividing the execution time taken by task by the sample time of the periodic task. This CPU utilization does not consider other PX4 modules to give the actual utilization of the processor.

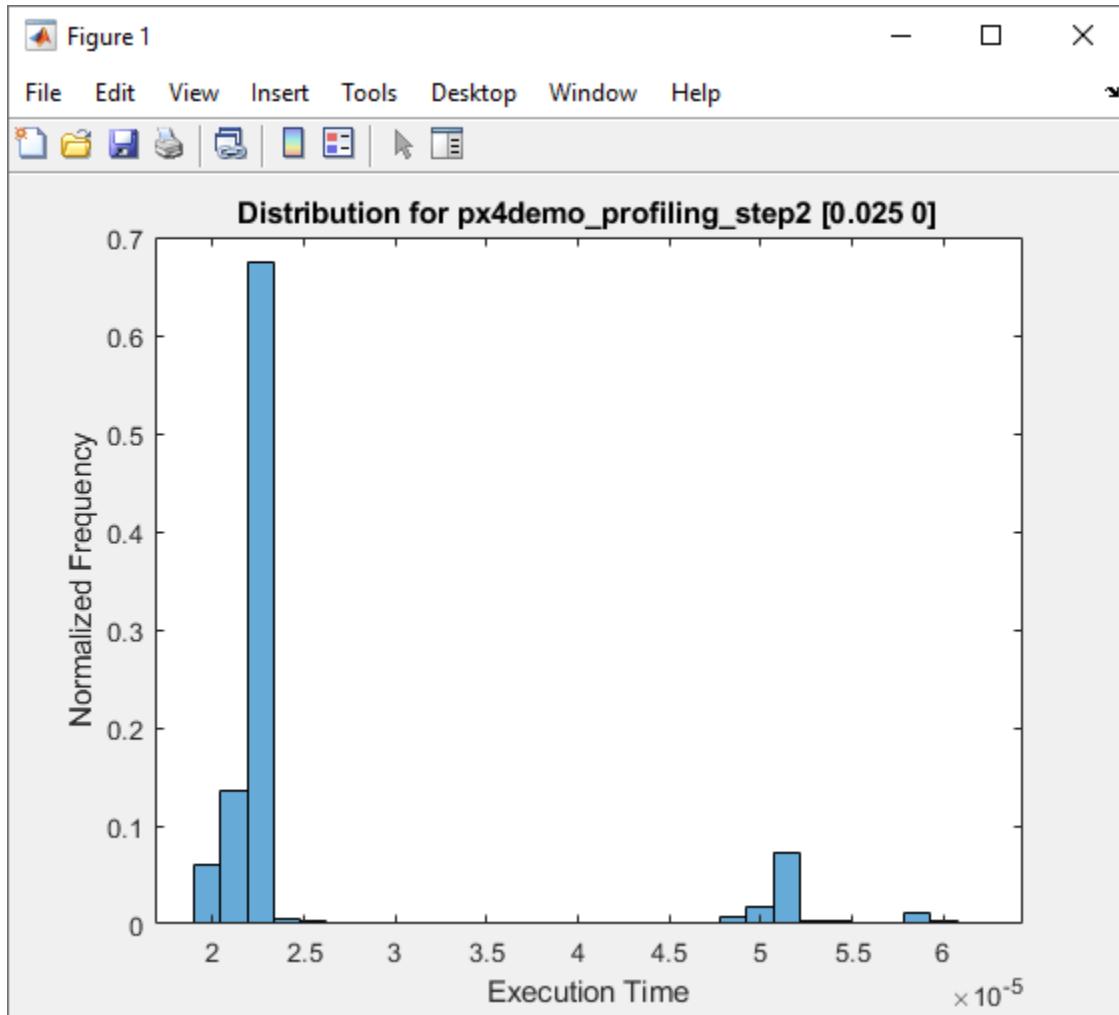
4. Definition of metrics.

The report has these sections:

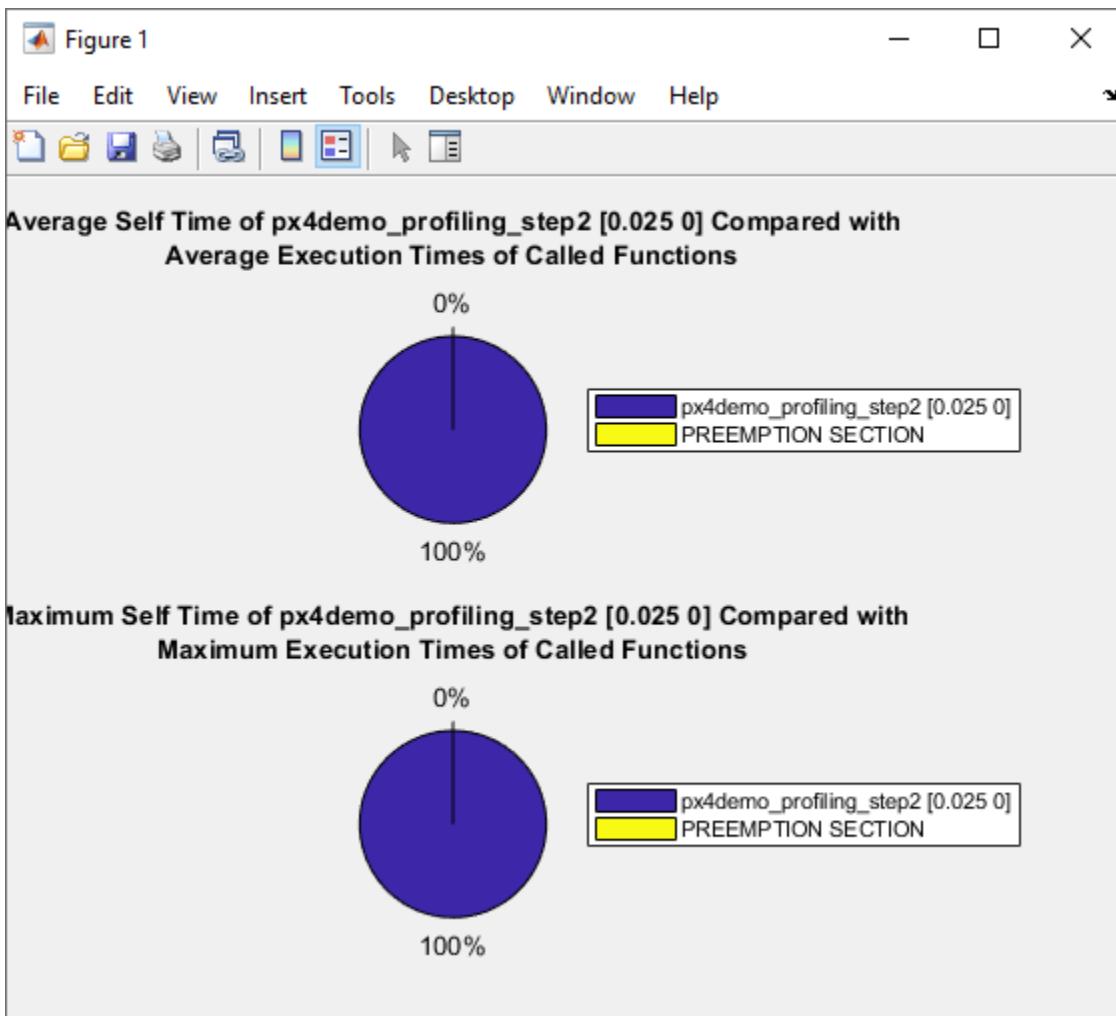
1. **Section:** Name of function from which code is generated.
2. **Maximum Execution Time in ns:** Longest time between start and end of code section.
3. **Average Execution Time in ns:** Average time between start and end of code section.
4. **Maximum Self Time in ns:** Longest execution time, excluding time spent in child sections.
5. **Average Self Time in ns:** Average execution time, excluding time spent in child sections.
6. **Calls:** Number of calls to the code section.
7. **MATLAB icon:** Icon that you click to display the profiled code section.
8. **SDI icon:** Icon that you click to visualize the profiling data over the duration of simulation.



9. **Bar chart icon:** Icon that you click to display the normalized frequency of the execution times of the task/function.



10. **Pie chart icon:** Icon that you click to display the time taken by a function and its child functions.



Metrics only report

A screenshot of the "Code Execution Profiling Report" window. The title bar says "Code Execution Profiling Report". The main area contains the following sections:

Code Execution Profiling Report for px4demo_profiling

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See [Code Execution Profiling](#) for more information.

Profiling data was stored on the target and uploaded to Simulink only at the end of the SIL/PIL execution. The hierarchy of calls to code sections cannot be identified, so the call structure in this report is flattened.

1. Summary

Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%.0f');
Timer frequency (ticks per second)	1e+06
Profiling data created	30-Nov-2021 13:38:36

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Calls
px4demo_profiling_initialize	27000	27000	1
px4demo_profiling_step0 [0.001 0]	25000	15083	50001
px4demo_profiling_step1 [0.01 0]	50000	43228	5000
px4demo_profiling_step2 [0.025 0]	20000	16194	2000
px4demo_profiling_step3 [0.05 0]	17000	14913	1000
px4demo_profiling_terminate	13000	13000	1

3. CPU Utilization [hide]

Task	Average CPU Utilization	Maximum CPU Utilization
px4demo_profiling_step0 [0.001 0]	1.508%	2.5%
px4demo_profiling_step1 [0.01 0]	0.4323%	0.5%
px4demo_profiling_step2 [0.025 0]	0.06478%	0.08%
px4demo_profiling_step3 [0.05 0]	0.02983%	0.034%
Overall CPU Utilization	2.035%	3.114%

4. Definitions

CPU Utilization: Percentage of CPU time assigned to a task. Computed by dividing task execution time by sample time.

OK Help

This report contains only summary metrics of Average/Maximum execution times of each task/function being profiled. Also, it contains the number of times each function/task is called during simulation.

Note: Profiling with **All data** or **Summary data** save options and **Detailed** option selected for Measure function execution times requires high bandwidth. This is because large amount of data must be sent in real-time. If the target does not have enough bandwidth to stream data in real-time, then select **Metrics Only** save option. Running profiling with this option stores summary profiling data on the target and will only send data to the host at the end of simulation.

Other Things to Try

Profile other Simulink models from the UAV Toolbox Support Package for PX4 Autopilots. Observe the time taken for implementing steps in the Simulink model that helps to improve its efficiency.

Related Topics

- “Code Execution Profiling on PX4 Targets”

PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics in Simulink

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to verify the controller design by deploying on the Pixhawk® hardware board, in HITL mode with UAV Dynamics contained in Simulink®.

The UAV Toolbox Support Package for PX4 Autopilots enables you to use Simulink to design a flight controller algorithm to follow the mission set in the QGroundControl (QGC). Additionally, you can design the UAV Dynamics in Simulink and simulate sensor values and use the same to communicate with Autopilot in HITL mode instead of third party simulators.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Go through the “PX4 Hardware-in-the-Loop System Architecture” document to understand the physical connections required to setup the Pixhawk and Simulink for HITL Simulation.
- Configure and set up Pixhawk in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any Pixhawk Series board as the PX4 Autopilot board and corresponding build target having `_multicopter` keyword from the drop-down list. This example uses Pixhawk 6x as autopilot board and `px4_fmu-v6x_multicopter` build target.

Required Third-Party Software This example requires this third-party software:

- QGroundControl (QGC)

Required Hardware To run this example, you will need the following hardware.

- Pixhawk Series flight controller
- Micro USB (Universal Serial Bus) type-B cable
- Micro-SD card
- Serial-to-USB FTDI converter
- Mini USB cable for FTDI
- Pixhawk serial port connectors

Methods to Run HITL Simulation from Simulink

This example has two different Simulink models; one for the Controller to be deployed on Autopilot and another containing the UAV Dynamics and sensor simulation. There are two methods which you can use to execute these two models.

- In the first method, the Controller is deployed on the Autopilot and the UAV Dynamics model will run on MATLAB® on host computer communicating with Autopilot.

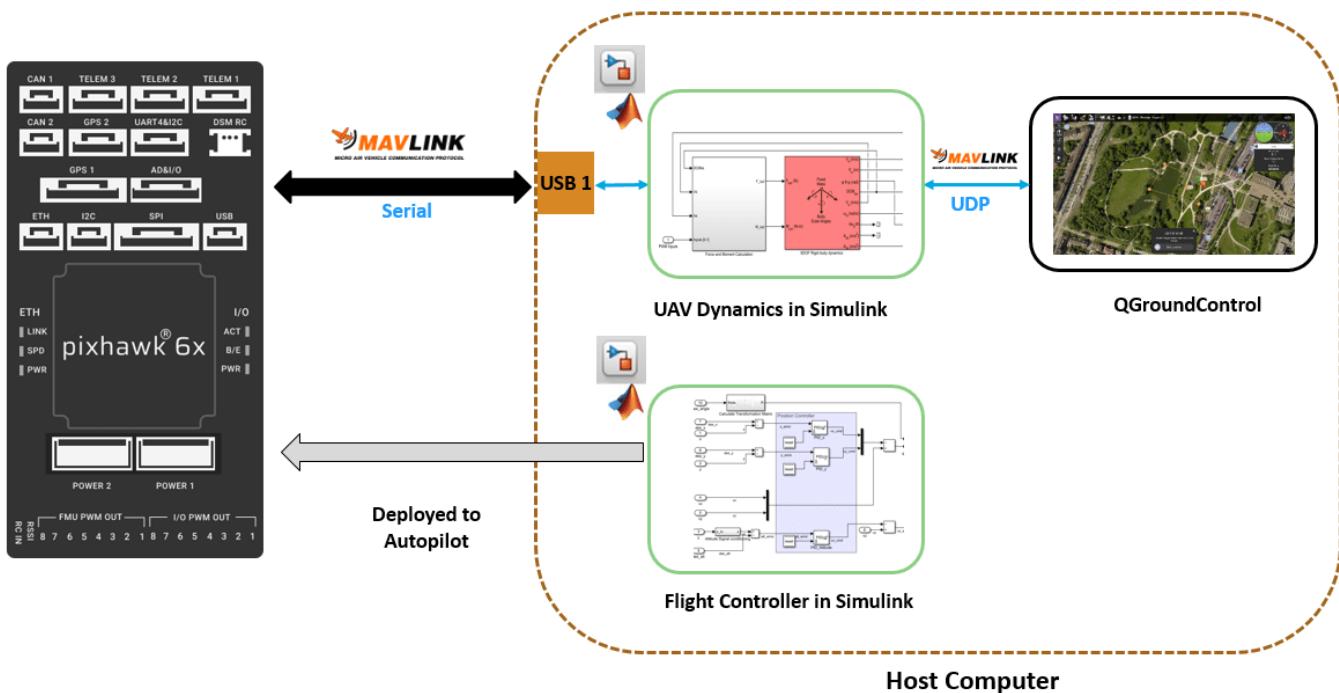
- In the second method, the Controller is deployed on Autopilot and will communicate with MATLAB on host computer using Monitor & Tune Simulation. The UAV Dynamics model will run on MATLAB on host computer communicating with Autopilot.

For the second method, running two Simulink models in same MATLAB can cause degraded performance of MATLAB. It is recommended to launch two separate sessions of same MATLAB, each one running one Simulink model. In the Method 1, the Controller model is deployed and does not run Monitor & Tune and hence one MATLAB instance is sufficient.

Method 1: Controller Deployed on Autopilot over Normal Mode (No Monitor & Tune)

Step 1: Make Hardware Connections and setup the Pixhawk in HITL mode

The diagram below illustrates the HITL setup and the physical communication between various modules.



1. Connect your Pixhawk board to the host computer using the USB cable.
2. Ensure that you have configured the Pixhawk board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
3. Ensure that you have setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.

Step 2: Open MATLAB Project and Controller and UAV Dynamics model

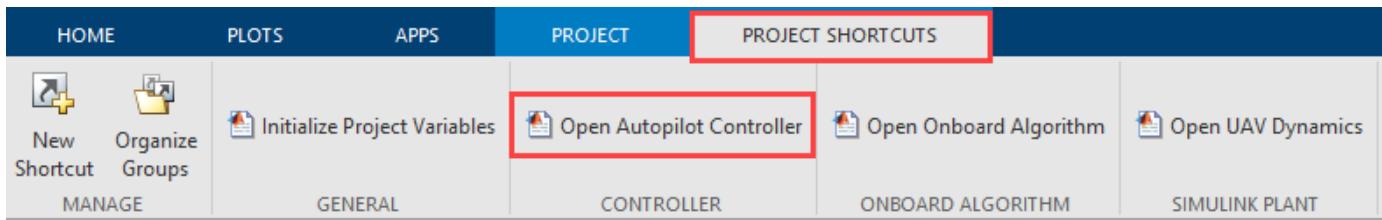
The support package includes an example project having PX4 controller and the UAV to follow the mission set in the QGroundControl (QGC).

1. Open MATLAB.

2. Open the example MATLAB project by executing this command at the MATLAB command prompt:

```
openExample('px4/PX4HTLSimulationSimulinkPlantExample')
```

3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open Autopilot Controller** to launch PX4 Controller named *Quadcopter_ControllerWithNavigation*.

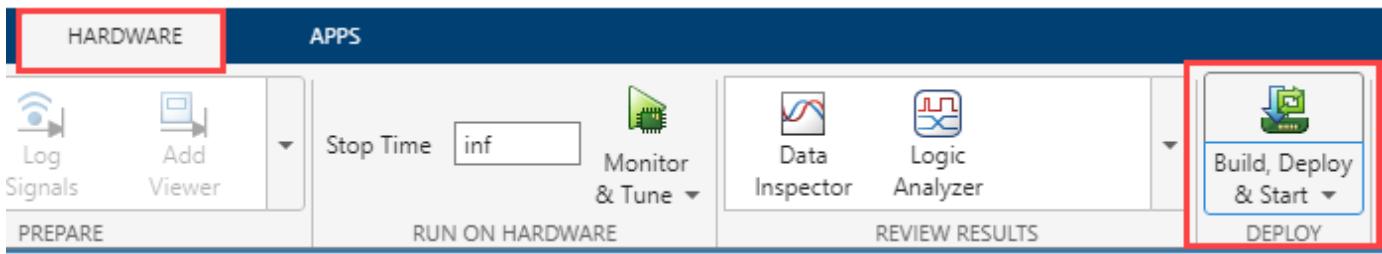


Step 3: Configure Simulink Controller model for HITL mode

1. Follow the instructions mentioned in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

2. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*.



The code will be generated for the Controller model and the same will be automatically deployed to the Pixhawk board (Pixhawk 6x in this example).

After the deployment is complete, QGroundControl will be automatically launched.

Note: If you are using Ubuntu®, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

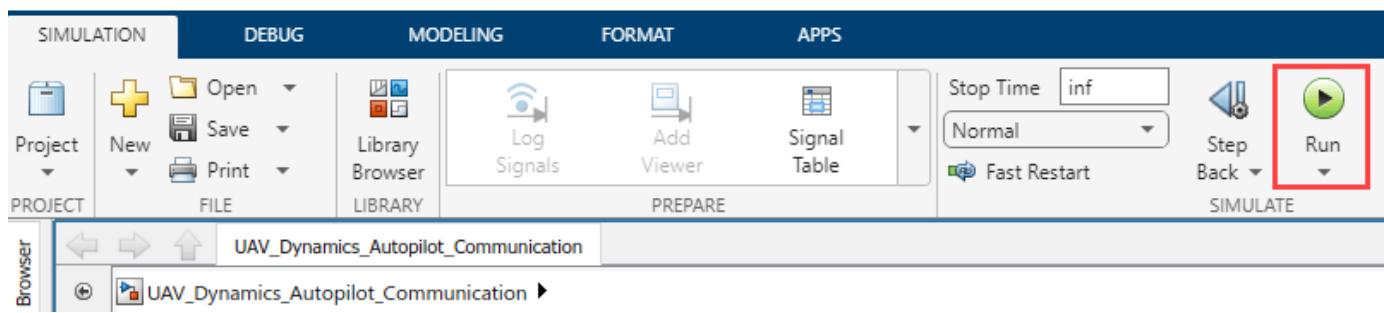
```
./QGroundControl.AppImage
```

Step 4: Run the UAV Dynamics model, upload Mission from QGroundControl and fly the UAV

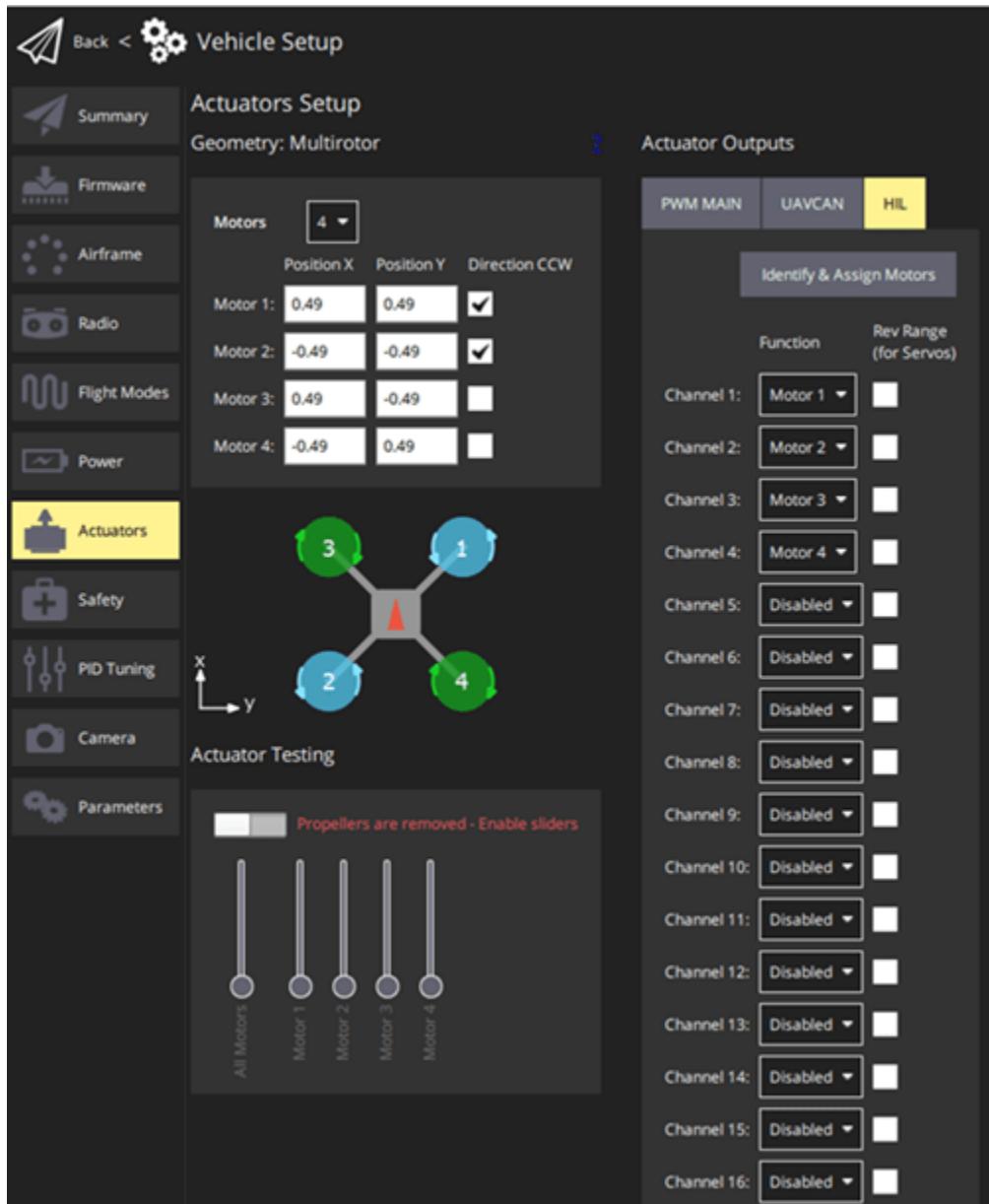
1. In the **Project Shortcuts** tab, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.

2. In this model, ensure that the COM ports are set in the MAVLink Bridge Source and MAVLink Bridge Sink blocks. To set the COM port:

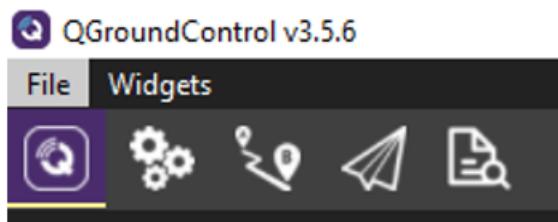
- Double-click the block to open the Block Parameters dialog box and then specify the value for the **Serial Port (Specify manually)** parameter.
3. In the Simulink toolbar of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model.



4. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



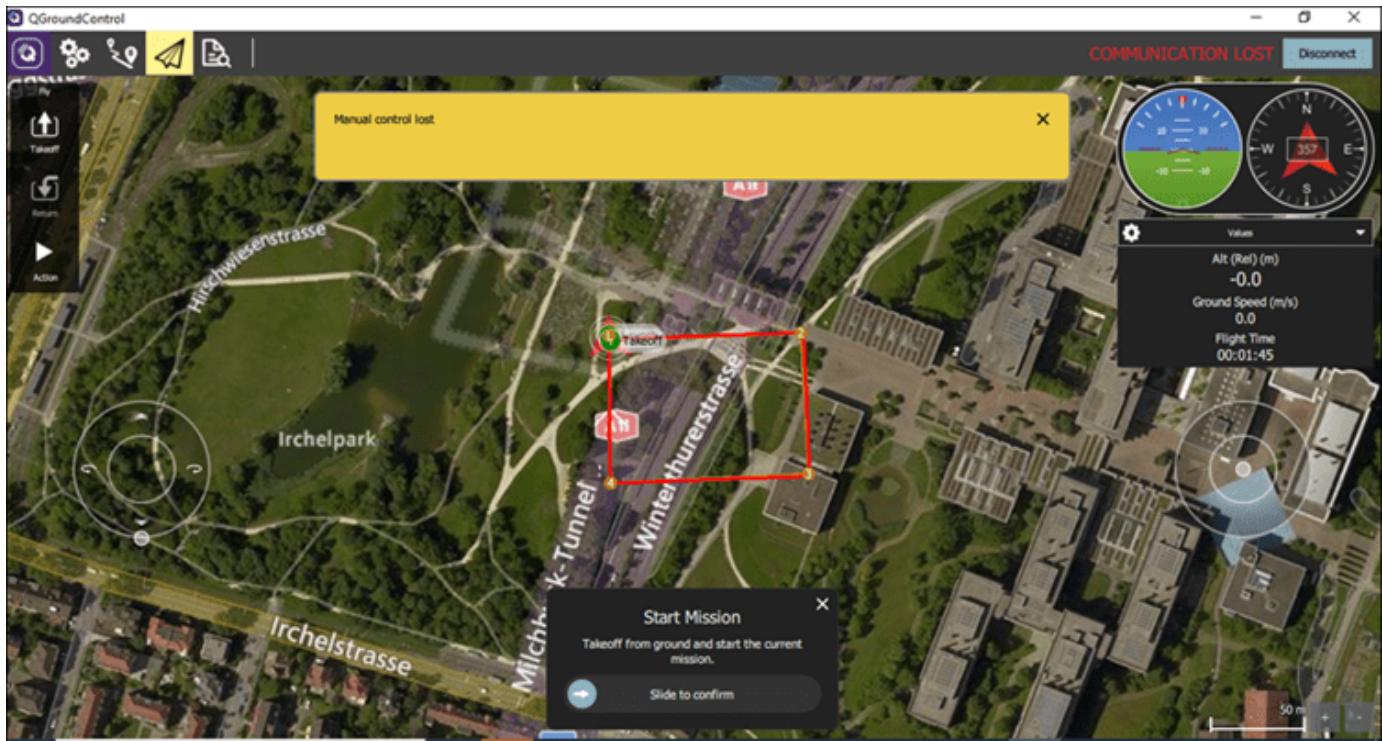
5. Navigate to the *Plan View*.



6. Create a mission. For information on creating a mission, see Plan View.

After you create a mission, it is visible in QGC.

7. Click Upload button in the QGC interface to upload the mission from QGroundControl.
8. Navigate to *Fly View* to view the uploaded mission.
9. Start the Mission in QGC. The UAV should follow the mission path.

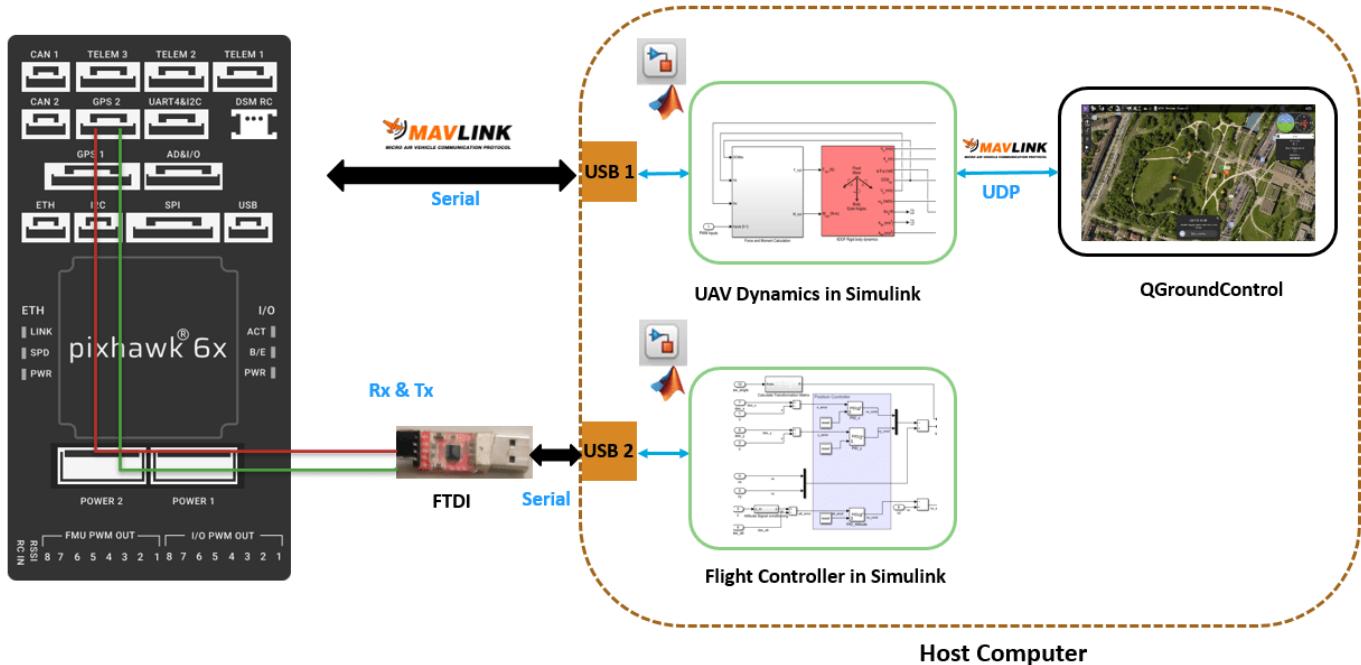


Method 2: Controller Deployed on Autopilot for Monitor & Tune Simulation

In this method, two separate sessions of the same MATLAB, each one running one Simulink model, are launched. In this method, you can run “Monitor and Tune the Model Running on PX4 Autopilots” Simulation to tune the algorithm in real time on the autopilot and log the real time signals in Simulink. In this case, an additional serial port is required to establish Monitor & Tune communication with the Autopilot. Since the USB port is already occupied for transferring MAVLink data between host computer and Autopilot in HITL mode, you need to use any of the other available serial ports on the Autopilot. In this example, GPS2 (dev/ttyS7) is used on Pixhawk 6x to communicate with Simulink running on Host Computer using Monitor & Tune Simulation.

Step 1: Make Hardware Connections and setup the Pixhawk in HITL mode

The diagram below illustrates the HITL setup and the physical communication between various modules. Note that the flight controller now communicates to the Pixhawk board GPS2 serial port over FTDI for Monitor & Tune Simulation.



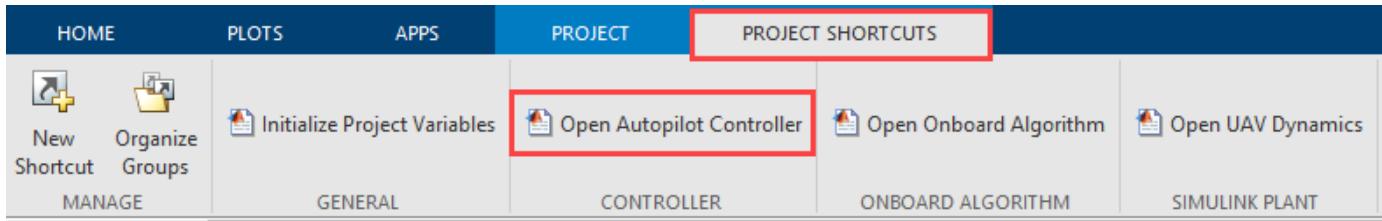
1. Connect your Pixhawk board to the host computer using the USB cable.
2. Connect the Rx & Tx of GPS2 port to the Tx and Rx of the FTDI (Serial to USB Converter) device which is connected to host computer over another USB port. For more information, see “Running Monitor & Tune Simulation over FTDI with Pixhawk 6x”.
3. Determine the COM Port for the FTDI device on host computer. This can be seen from the Device Manager.
4. Ensure that you have configured the Pixhawk board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
5. Ensure that you have setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.

Step 2a: Open first instance of MATLAB and launch MATLAB project and Controller model

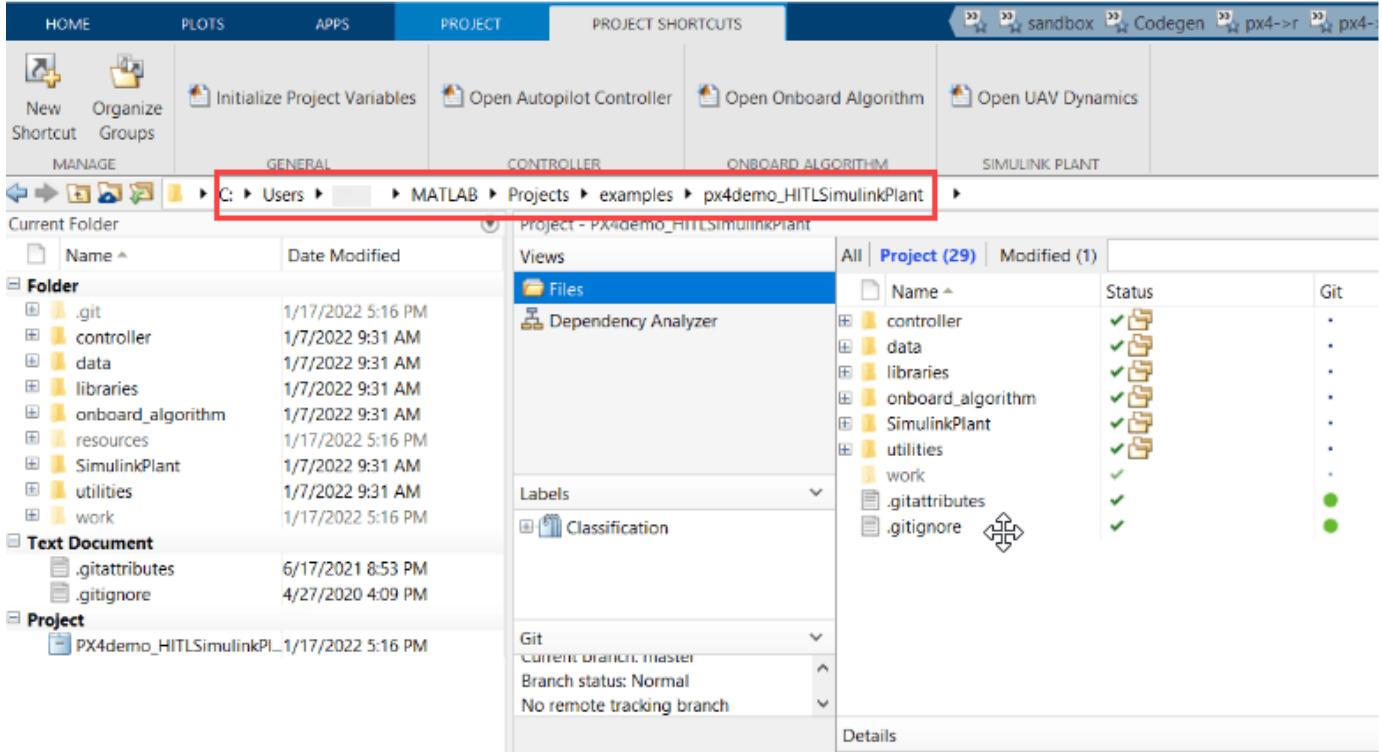
The support package includes an example project having PX4 controller and the UAV to follow the mission set in the QGroundControl (QGC).

1. Open MATLAB.
2. Open the example MATLAB project by executing this command at the MATLAB command prompt:

```
openExample('px4/PX4HITLSimulationSimulinkPlantExample')
```
3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open Autopilot Controller** to launch PX4 Controller named *Quadcopter_ControllerWithNavigation*.



4. Copy the MATLAB Project Path to clipboard.

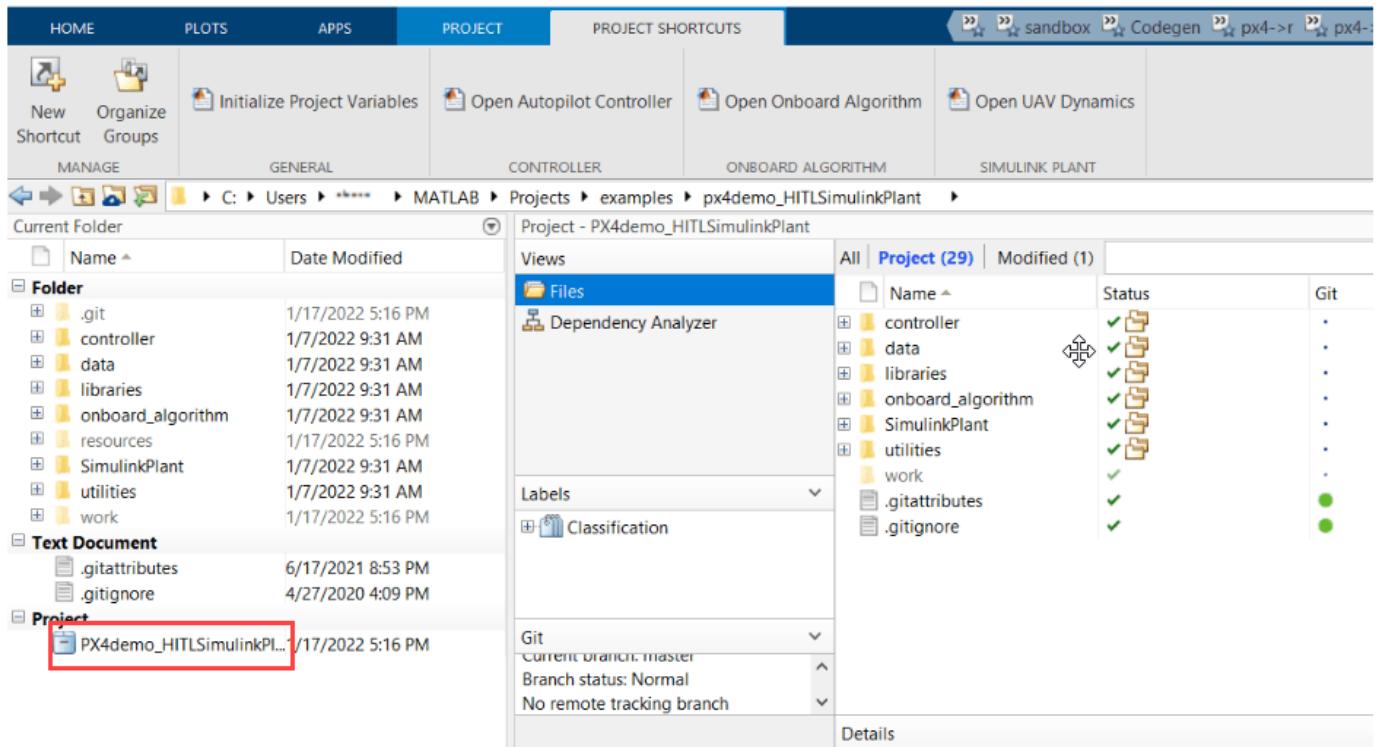


Step 2b: Open second instance of MATLAB and launch the previous MATLAB Project and UAV Dynamics model

1. Open the second instance of the same MATLAB version.
2. Navigate to the project path previously copied in Step 2a in current MATLAB.

```
fx >> cd C:\Users\...\MATLAB\Projects\examples\px4demo_HILSimulinkPlant
```

3. Click on the .prj file to launch the same Project in current MATLAB.



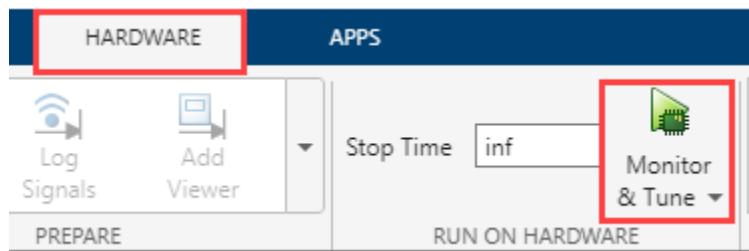
4. In the **Project Shortcuts** tab, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.

Step 3: Configure Simulink Controller model for Monitor & Tune Simulation

1. For the Controller model launched in first MATLAB, follow the instructions mentioned in “Configure Simulink Model for Monitor & Tune Simulation with Hardware-in-the-Loop (HITL)” document.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

2. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*



3. The code will be generated for the Controller model and the same will be automatically deployed to the Pixhawk board (in this case Pixhawk 6x). Pixhawk should start communicating with host over Monitor & Tune Simulation.

After the deployment is complete, QGroundControl will be automatically launched. **Note:** If you are using Ubuntu, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

```
./QGroundControl.AppImage
```

Step 4: Run the UAV Dynamics model, upload Mission from QGroundControl and fly the UAV

Follow Step 4 from Method 1, to run the UAV Dynamics model, create and upload mission from QGC and fly the UAV.

Scenario Simulation and Flight Visualization with PX4 Hardware-in-the-Loop (HITL) and UAV Dynamics in Simulink

This example demonstrates 3D scenario Simulation and Flight visualization with PX4® Hardware-in-the-Loop (HITL) and UAV Dynamics contained in Simulink®. Unreal Engine® simulation environment is used for the 3D scenario Simulation and visualization. For more information, see Unreal Engine Simulation for Unmanned Aerial Vehicles.

Limitation: The Unreal Engine simulation environment is supported only on Microsoft® Windows® system.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Go through the “PX4 Hardware-in-the-Loop System Architecture” document to understand the physical connections required to setup the Pixhawk® and Simulink for HITL Simulation.
- Configure and set up Pixhawk in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any Pixhawk Series board as the PX4 Autopilot board and corresponding build target having `_multicopter` keyword from the drop-down list. This example uses Pixhawk 6x as autopilot board and `px4_fmu-v6x_multicopter` build target.
- Familiarize with the co-simulation framework for UAVs, see Unreal Engine Simulation for Unmanned Aerial Vehicles

Required Third-Party Software

This example requires this third-party software:

- QGroundControl (QGC)

Required Hardware

To run this example, you will need the following hardware:

- Host PC Configured with MATLAB supported GPU. It is recommend to use GPU with compute capability of more than 5.
- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card
- Pixhawk serial port connectors.

Workflow to Run Unreal Engine Flight Visualization Model Along with Pixhawk in HITL Mode

This example uses three different Simulink models.

- Simulink model for Flight Controller to be deployed on PX4 Autopilot.
- Simulink model for UAV Dynamics and sensor simulation.
- Simulink model for flight visualization with Unreal Engine Simulation for Unmanned Aerial Vehicles.

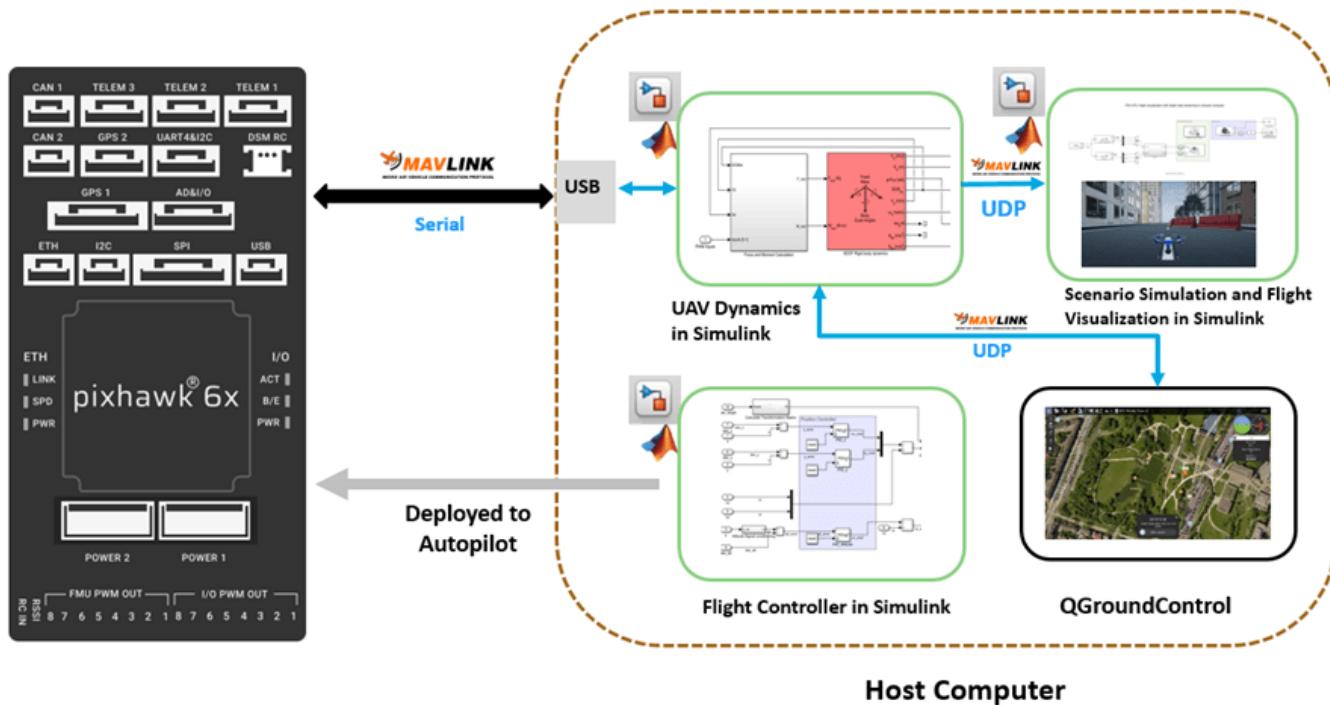
To avoid performance degradation in MATLAB® due to three different Simulink models running at the same time, launch two separate sessions of same MATLAB.

In first session of MATLAB, the Flight Controller will be deployed on the Autopilot and the UAV Dynamics model will be running on host computer communicating with Autopilot.

In second session of MATLAB, the Simulink model for flight visualization with Unreal Engine Simulation will be running.

Step 1: Make Hardware Connections and Setup the Pixhawk in HITL Mode

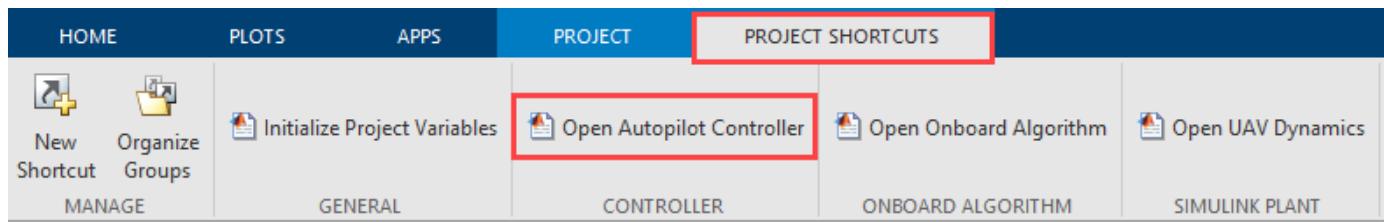
The below diagram illustrates the HITL setup and the physical communication between various modules.



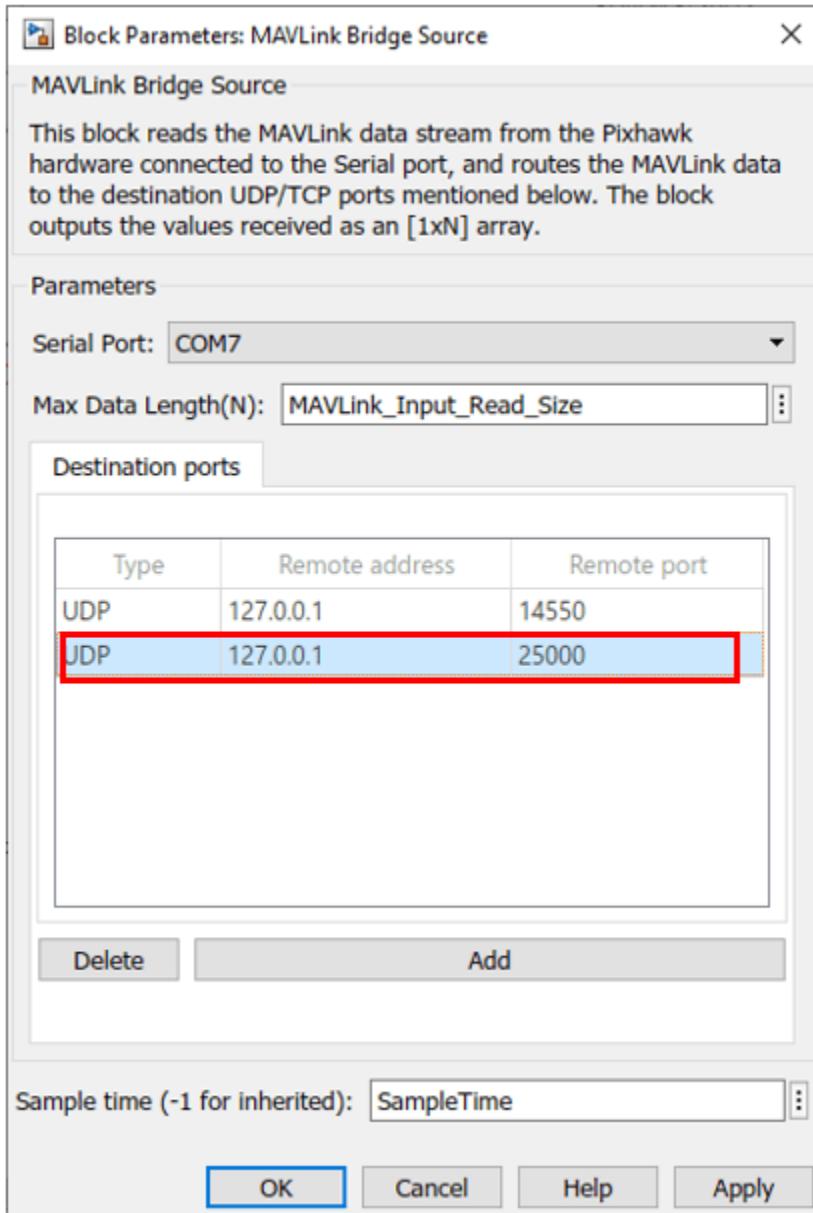
1. Connect your Pixhawk board to the host computer using the USB cable.
2. Ensure that you have configured the Pixhawk board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
3. Ensure that you have setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.

Step 2: Launch First Session of MATLAB and the MATLAB Project

1. Open MATLAB.
2. Open the px4demo_HardwareInLoopWithSimulinkPlant MATLAB project.
3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open Autopilot Controller** to launch PX4 Controller named *Quadcopter_ControllerWithNavigation*.



4. In the **Project Shortcuts** tab, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.
5. Open the Simulink Plant model *UAV_Dynamics_Autopilot_Communication* and a connection for the flight visualization in the MAVLink Bridge Source block. Double-click the **MAVLink Bridge Source** block to open the block Parameters dialog box.

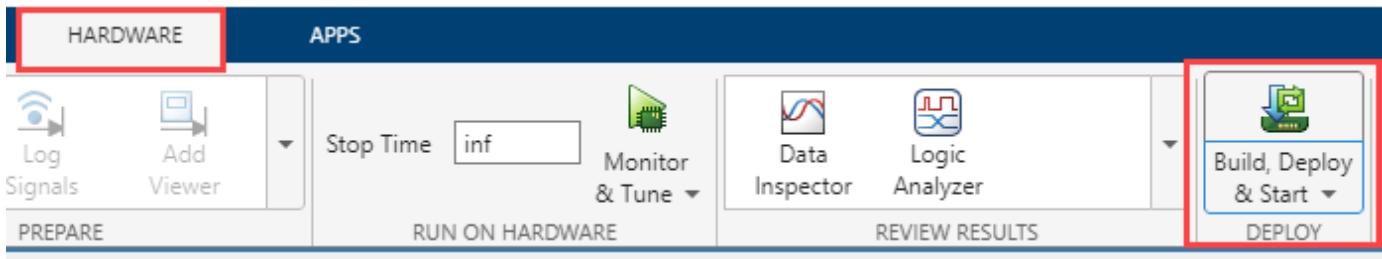


Step 3: Configure Simulink Controller Model for HITL Mode

1. Follow the instructions mentioned in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

2. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*.

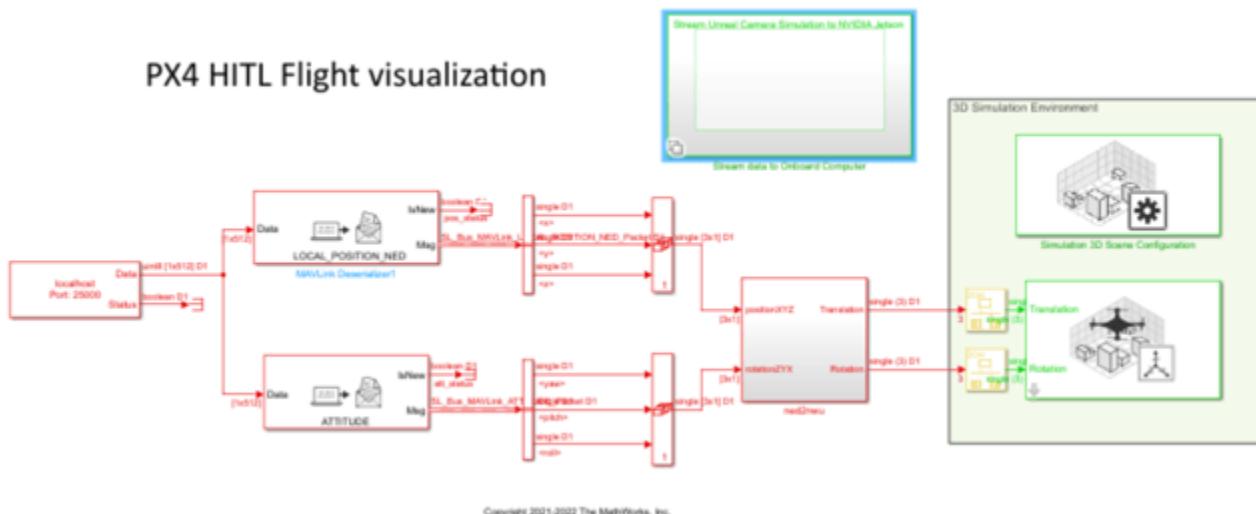
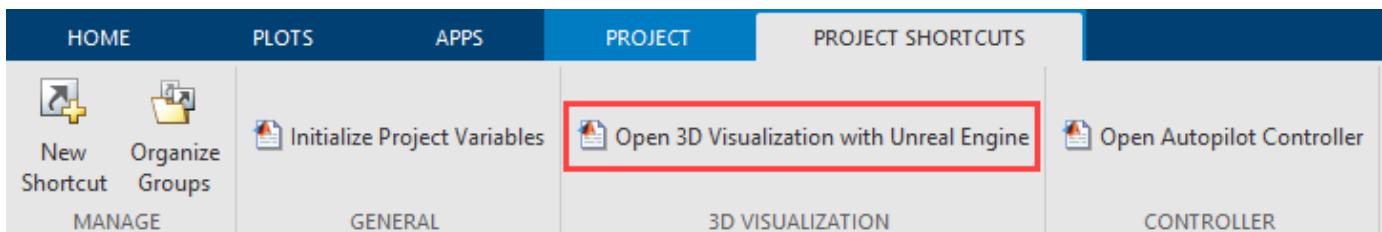


The code will be generated for the Controller model and the same will be automatically deployed to the Pixhawk board (Pixhawk 6x in this example).

After the deployment is complete, QGroundControl will be automatically launched.

Step 4: Launch Second Session of MATLAB and Open the Flight Visualization Model

1. Launch second session of MATLAB.
2. Open the px4demo_HardwareInLoopWithSimulinkPlant MATLAB project.
3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open 3D Visualization with Unreal Engine** to launch the onboard model named *Unreal_3DVisualization*.



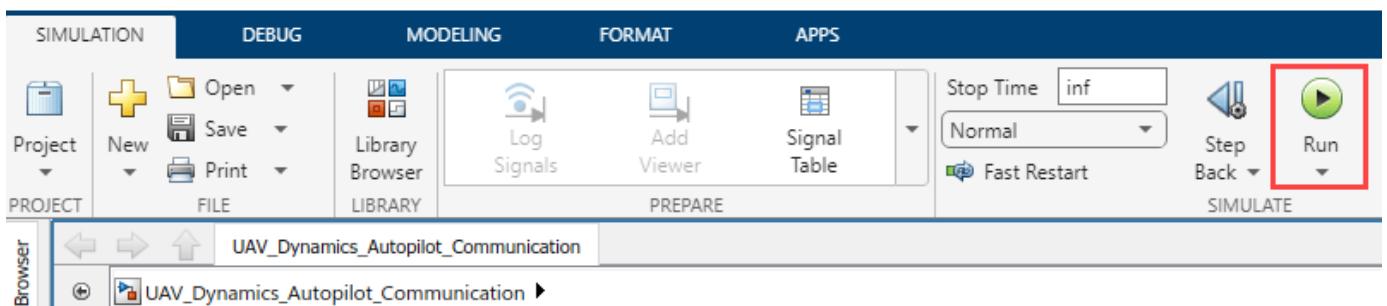
In this model, The MAVLink data from the PX4 Autopilot is received over UDP (port : 25000) and is used to decode the position and attitude data of the UAV. After coordinate conversion, it is then passed to the Simulation 3D UAV Vehicle block for flight visualization.

4. On the **Simulation** tab, click **Run** to simulate the model. Once the model starts running, you will see the Unreal simulation environment getting launched. A sample screen is shown below.

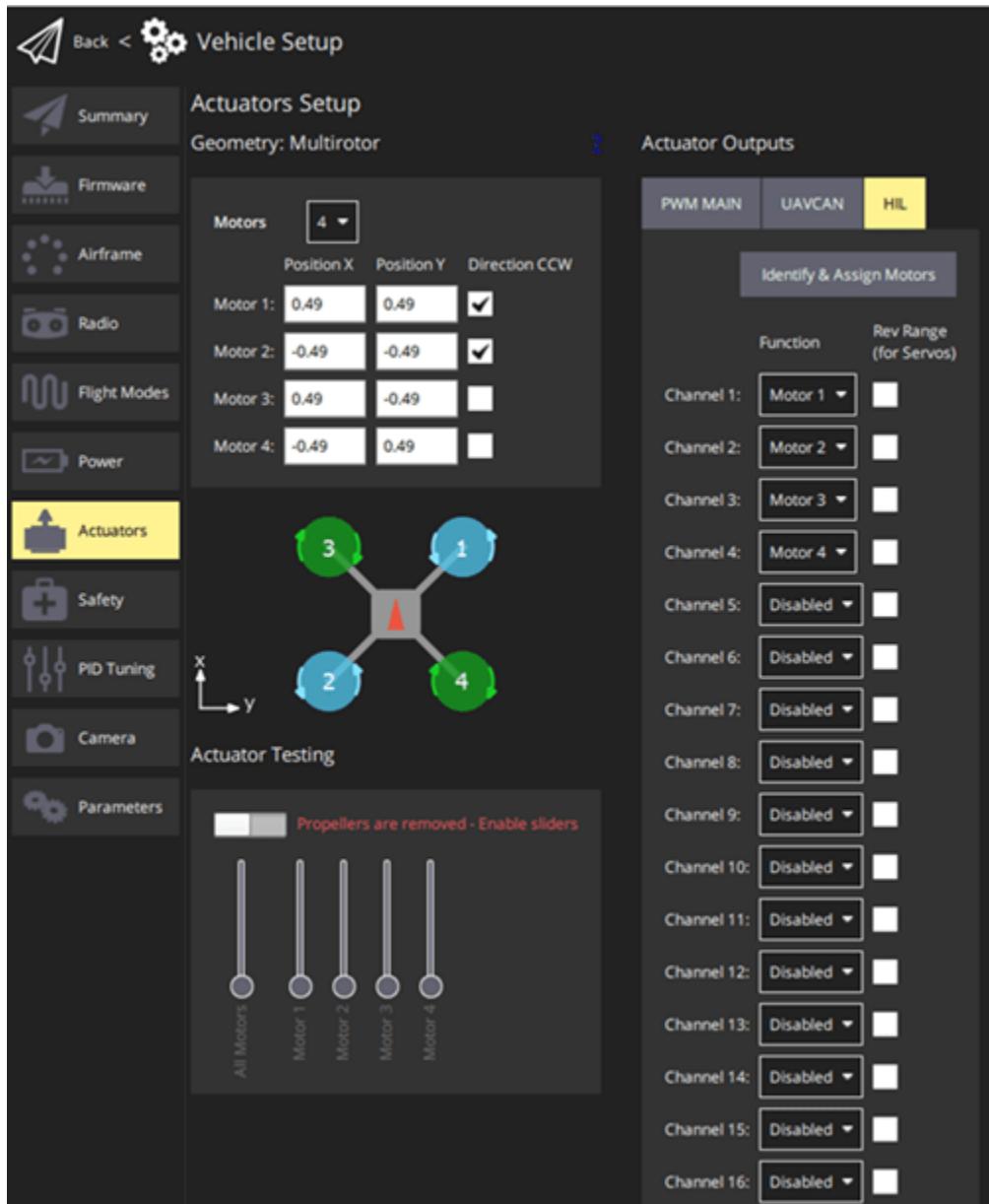


Step 5: Run the UAV Dynamics Model, Upload Mission from QGroundControl and Fly the UAV

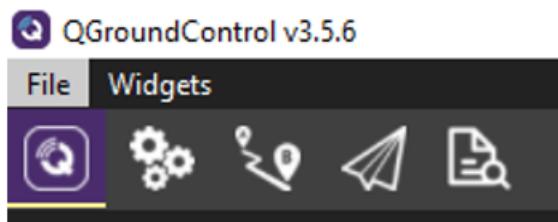
1. In the Simulink toolbar of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model.



2. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



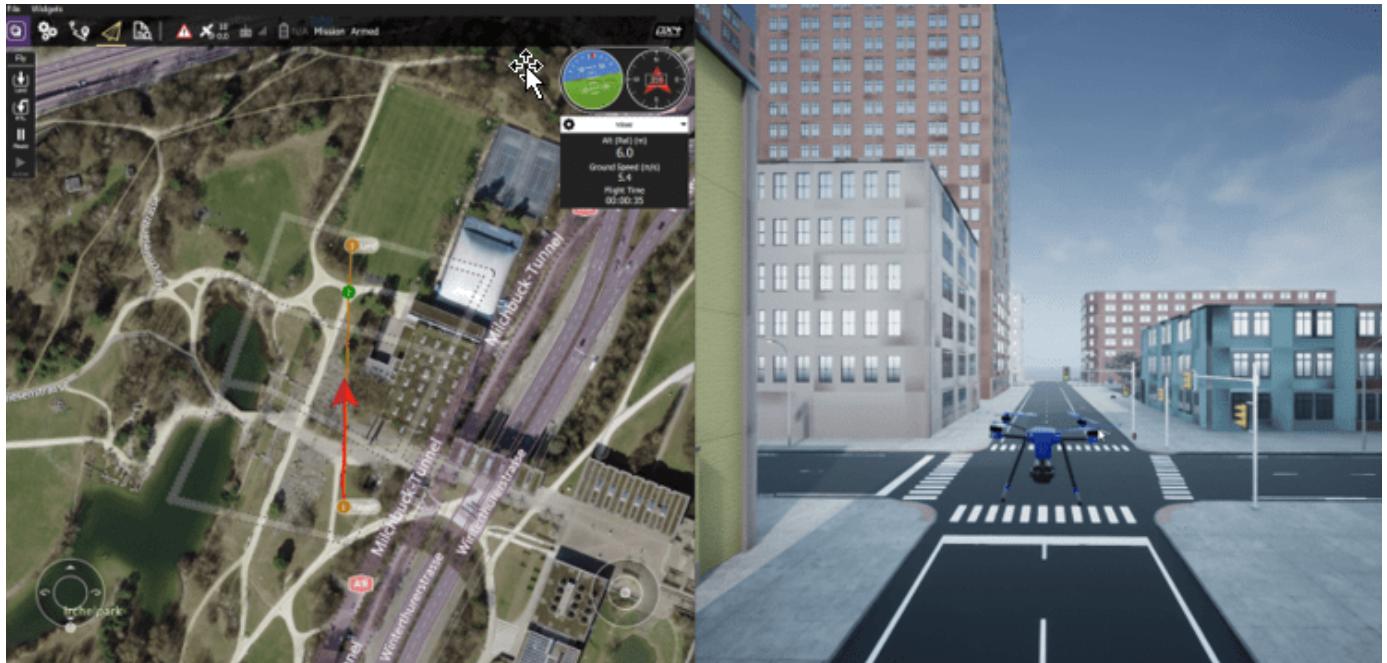
3. Navigate to the *Plan View*.



4. Create a mission. For information on creating a mission, see Plan View.

After you create a mission, it is visible in QGC.

5. Click Upload button in the QGC interface to upload the mission from QGroundControl.
6. Navigate to *Fly View* to view the uploaded mission.
7. Start the Mission in QGC. The UAV should follow the mission path and the flight will be simulated in the Unreal environment.



Troubleshooting

- While Simulating the visualization model in Step 4, you might get any eSTD exceptionc errors such as, "some module could not be found". Change the compiler to Microsoft Visual C++ 2019 using `mex --setup C++` command to fix the issue.
- If the Unreal environment simulation is very slow, ensure that your Host PC is Configured with MATLAB supported GPU (GPU with compute capability of more than 5).

PX4 Autopilot and NVIDIA Jetson in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics Modeled in Simulink

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to verify an autonomous algorithm deployed on NVIDIA® Jetson™ as Onboard Computer along with Pixhawk® hardware board, in HITL mode with UAV Dynamics contained in Simulink®.

For Autonomous algorithms which are computationally intensive, you can use an Onboard Computer on the drone along with the Autopilot. NVIDIA Jetson is a commonly used Onboard Computer for drones. Using Simulink, you can design a complex Autonomous algorithm and deploy the same on NVIDIA Jetson.

This example also showcases 3D scenario simulation during the flight and streaming the image to the onboard computer using the simulated camera sensor. Unreal Engine® simulation environment is used for the 3D scenario Simulation and visualization. This image can be received in NVIDIA Jetson and can be used for scene aware intelligent path planning.

In summary, in this example you,

- Enable onboard computer workflows with PX4 HITL.
- Implement a PX4 path planning interface in Simulink and deploy on NVIDIA Jetson.
- Enable flight visualization with PX4 HITL and stream simulated camera image to NVIDIA Jetson.
- Run and complete a UAV mission with onboard computer.

Limitation: The Unreal Engine simulation environment is supported only on Microsoft® Windows® system. If you are using a Linux® system, skip adding the 3D scenario simulation step and still be able to complete this example.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Go through the “PX4 Hardware-in-the-Loop System Architecture” document to understand the physical connections required to setup the Pixhawk and Simulink for HITL Simulation.
- Configure and set up Pixhawk in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any Pixhawk Series board as the PX4 Autopilot board and corresponding build target having `_multicopter` keyword from the drop-down list. This example uses Pixhawk 6x as autopilot board and `px4_fmu-v6x_multicopter` build target.
- Familiarize with the co-simulation framework for UAVs, see “Unreal Engine Simulation for Unmanned Aerial Vehicles”

It is recommended to understand the “PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics in Simulink” on page 1-214 example.

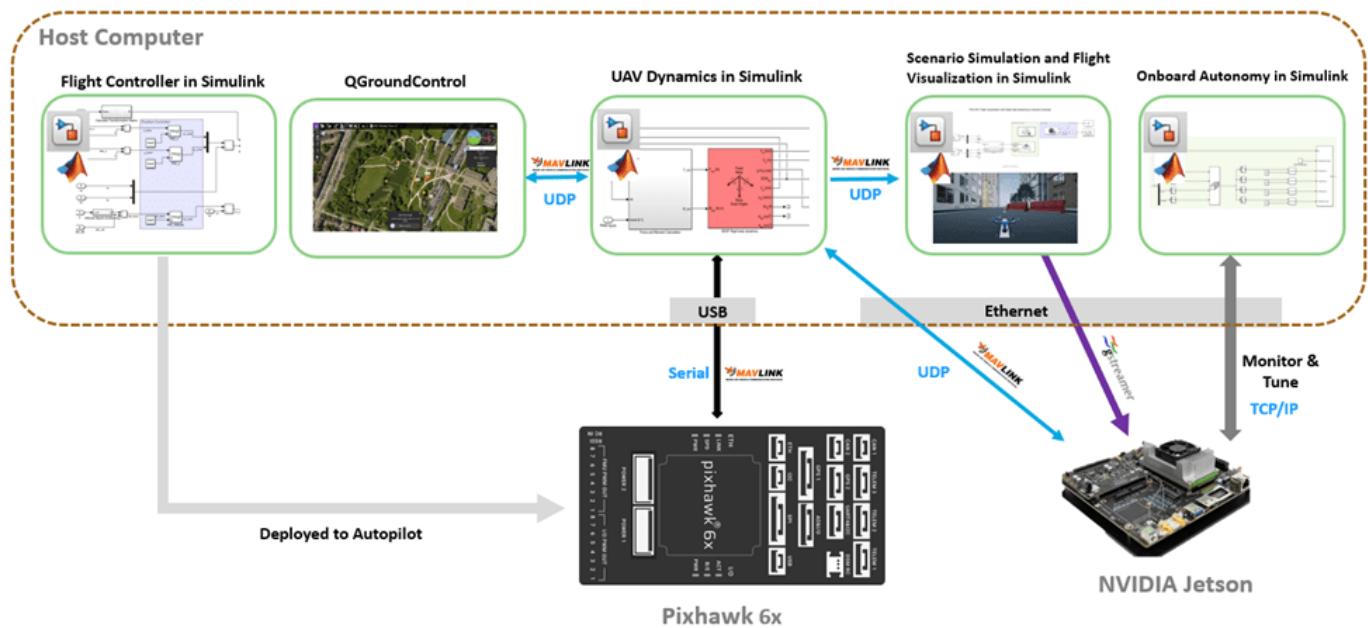
Required Third-Party Software This example requires this third-party software:

- QGroundControl (QGC)

Required Hardware To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card
- Pixhawk serial port connectors
- NVIDIA Jetson & power adaptor
- Host PC Configured with “GPU Computing Requirements” (Parallel Computing Toolbox). It is recommended to use GPU with compute capability of more than 5.

Workflow to Run Model on NVIDIA Jetson Along with Pixhawk in HITL Mode



The above diagram illustrates the PX4 and NVIDIA Jetson HITL setup and the physical communication between various modules.

This example uses four different Simulink models.

- Simulink model for Flight Controller to be deployed on PX4 Autopilot.
- Simulink model for UAV Dynamics and sensor simulation.
- Simulink model for Autonomous algorithm to be deployed on NVIDIA Jetson.
- Simulink model for flight visualization with Unreal Engine Simulation for Unmanned Aerial Vehicles.

To avoid performance degradation in MATLAB® due to three different Simulink models running at the same time, launch three separate sessions of same MATLAB.

- In the first session of MATLAB, the Flight Controller is deployed on Autopilot and the UAV Dynamics model will run on host computer communicating with Autopilot.
- In the second session of MATLAB, the Simulink model for flight visualization with Unreal Engine Simulation will be running. This can be skipped if you opt to not add the flight visualization.
- In the third session of MATLAB, the Simulink model for NVIDIA Jetson communicates with MATLAB on host computer using Monitor & Tune Simulation.

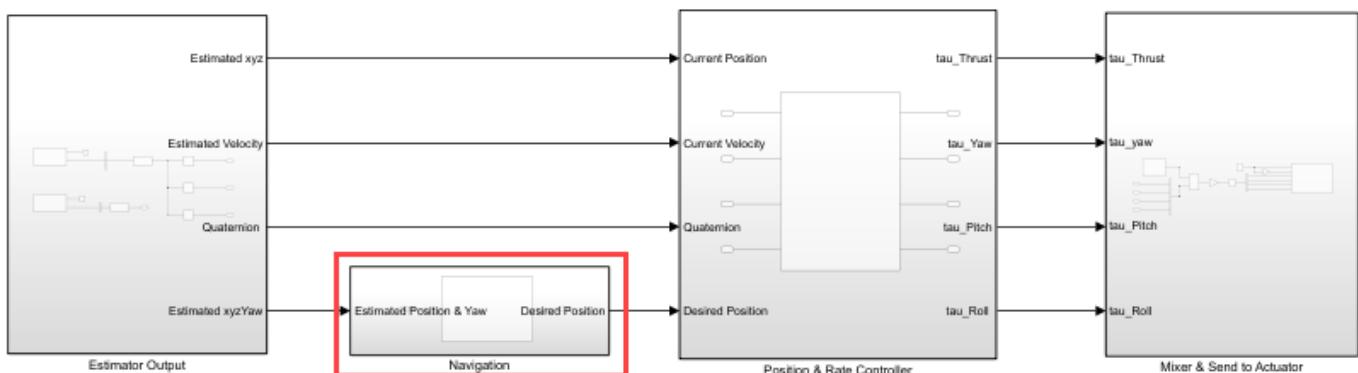
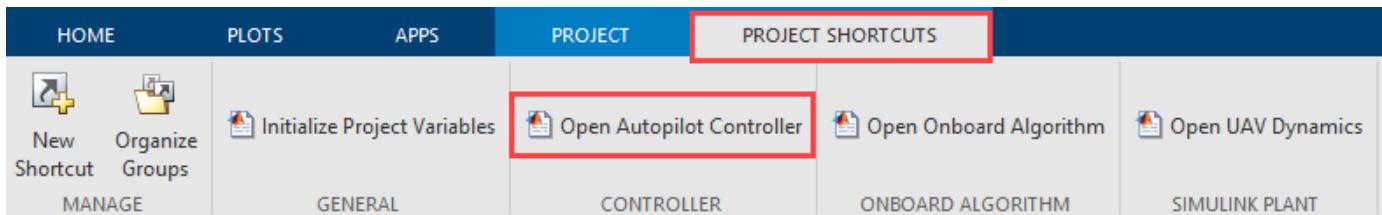
Step 1: Make Hardware Connections and Set Up the Pixhawk in HITL mode

1. Connect your Pixhawk board to the host computer using the USB cable.
2. Ensure that you have configured the Pixhawk board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
3. Ensure that you have setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.
4. Set up and configure your NVIDIA Jetson on network using “MATLAB Coder Support Package for NVIDIA Jetson and NVIDIA DRIVE Platforms” (MATLAB Coder).

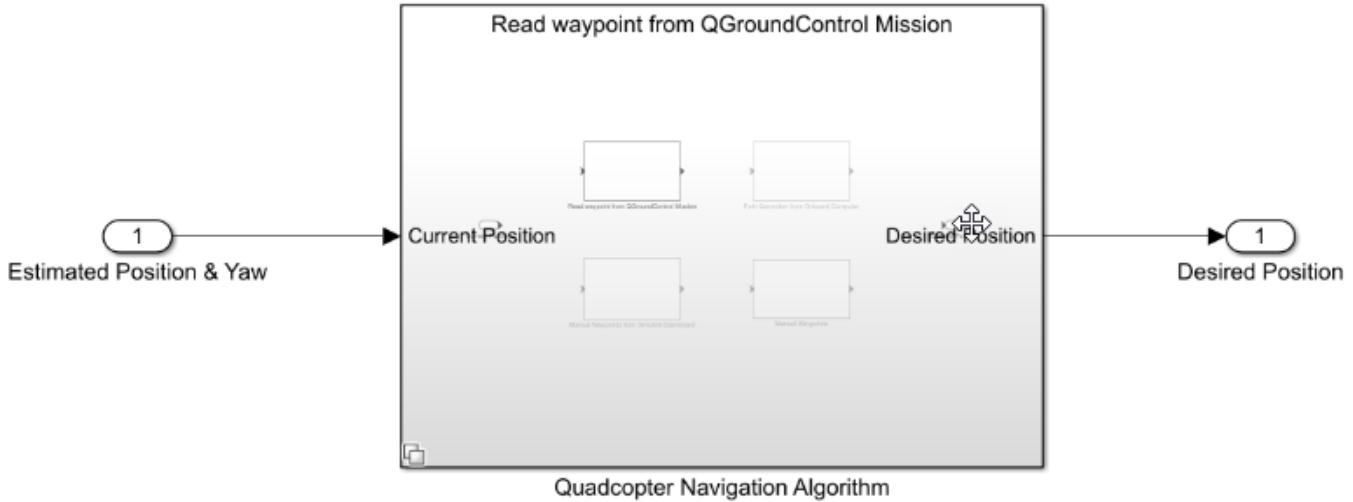
Step 2: Launch First Session of MATLAB and the MATLAB Project

The support package includes an example MATLAB project having the PX4 flight controller and the UAV to follow the mission set in the QGroundControl (QGC).

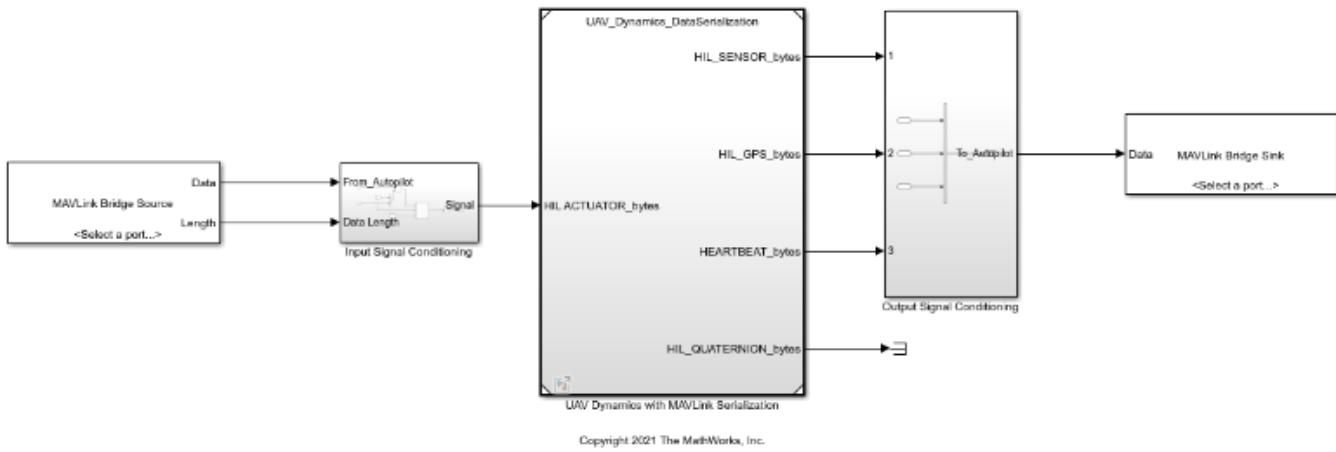
1. Open MATLAB.
2. Open the px4demo_HardwareInLoopWithSimulinkPlant MATLAB project.
3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open Autopilot Controller** to launch PX4 Controller named *Quadcopter_ControllerWithNavigation*.



4. Navigate to Navigation subsystem. This is a “Implement Variations in Separate Hierarchy Using Variant Subsystems” (Simulink) with guidanceType as the variant control variable. Define guidanceType = 1 in the global workspace to choose the navigation subsystem for this example.



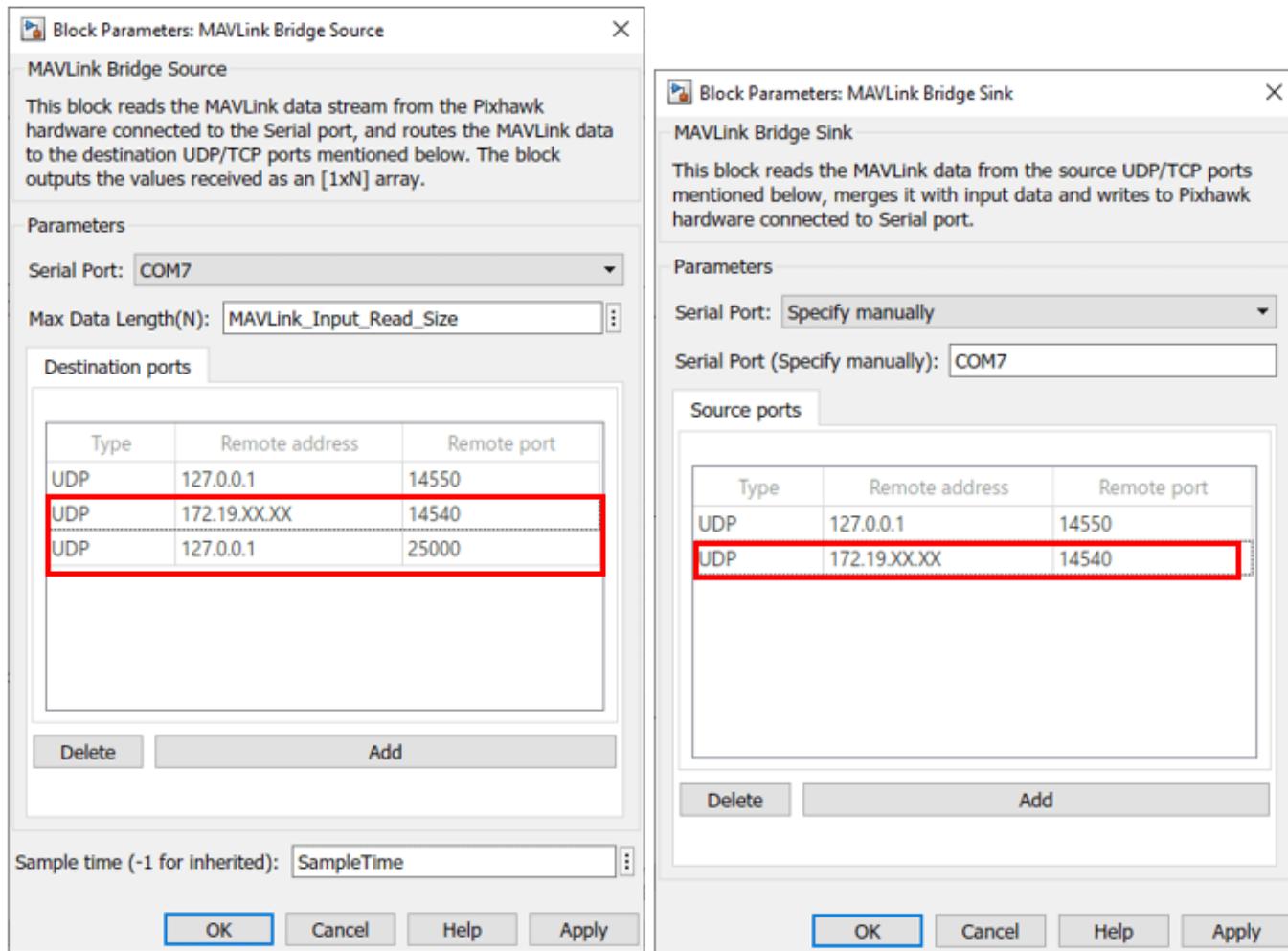
5. In the **Project Shortcuts** tab, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.



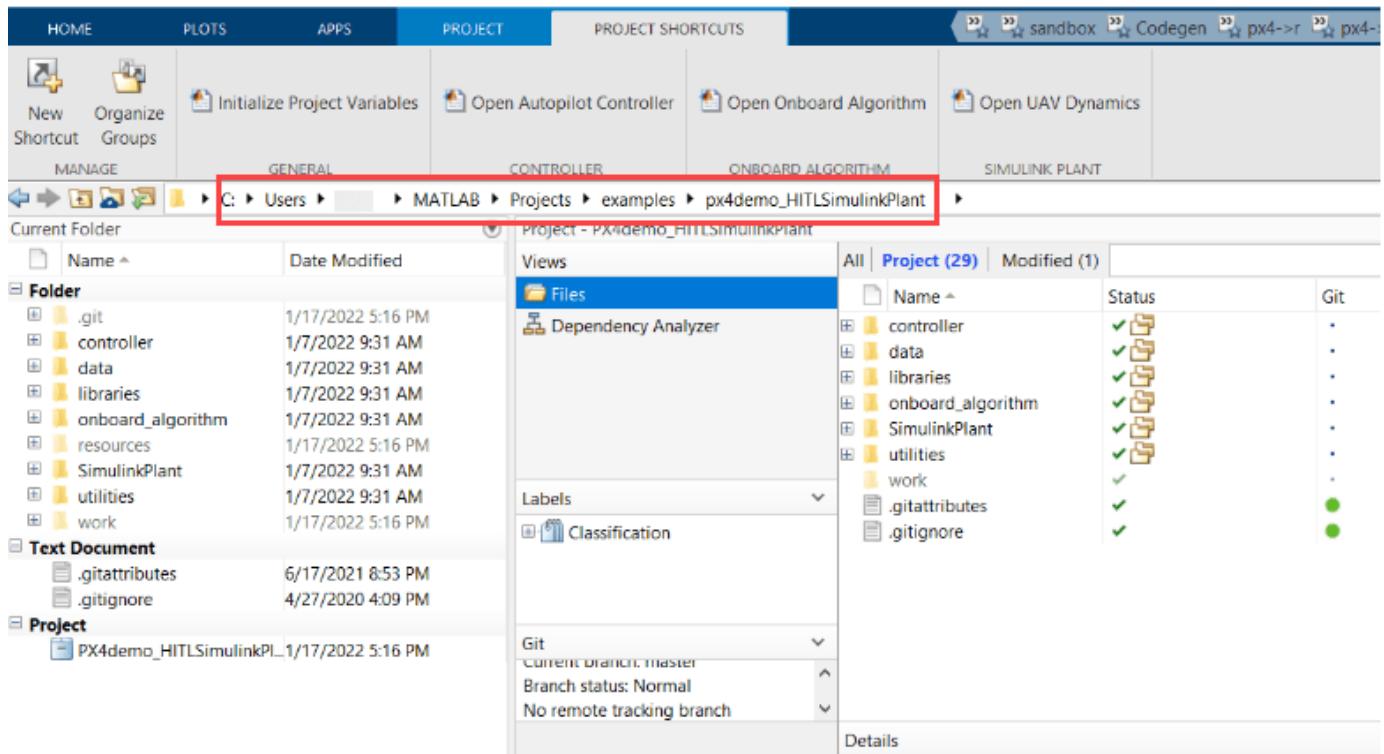
6. Open the Simulink Plant model *UAV_Dynamics_Autopilot_Communication* and configure the serial port. Select the serial port of Pixhawk which is connected to the host computer. Add the following UDP connections of onboard computer in the MAVLink Bridge blocks. For more information, see “MAVLink Connectivity for QGC, On-board Computer and Simulink Plant”. Double-click the MAVLink Bridge blocks to open the block Parameters dialog box.

a. Add the IP address of onboard computer (NVIDIA Jetson) in the MAVLink Bridge Source and MAVLink Bridge Sink blocks. Ensure that you can ping the NVIDIA Jetson successfully from the host PC. Enter the port number as 14540.

b. Add localhost connection for the flight visualization in the MAVLink Bridge Source block. Enter the port as 25000. Skip this if you are opting to not add the flight visualization.



7. Copy the MATLAB Project Path to clipboard.

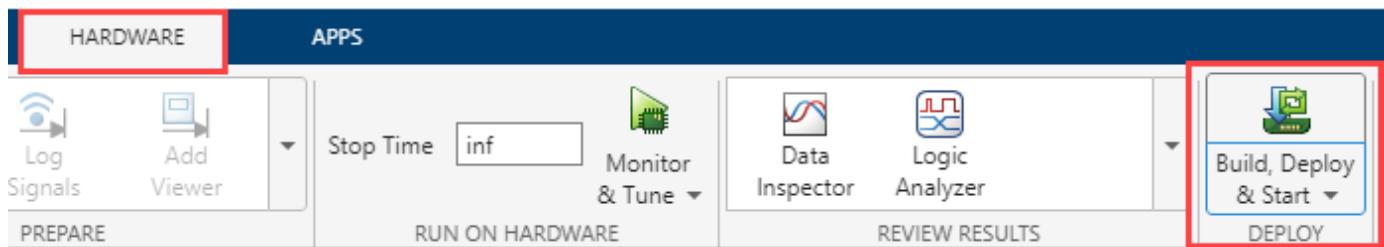


Step 3: Configure Simulink Controller Model for HITL Mode

1. Follow the instructions mentioned in "Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation".

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

2. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*.



The code will be generated for the Controller model and the same will be automatically deployed to the Pixhawk board (Pixhawk 6x in this example).

After the deployment is complete, QGroundControl will be automatically launched.

Note: If you are using Ubuntu®, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

`./QGroundControl.AppImage`

Step 4: Launch Second Session of MATLAB and Open the Flight Visualization Model

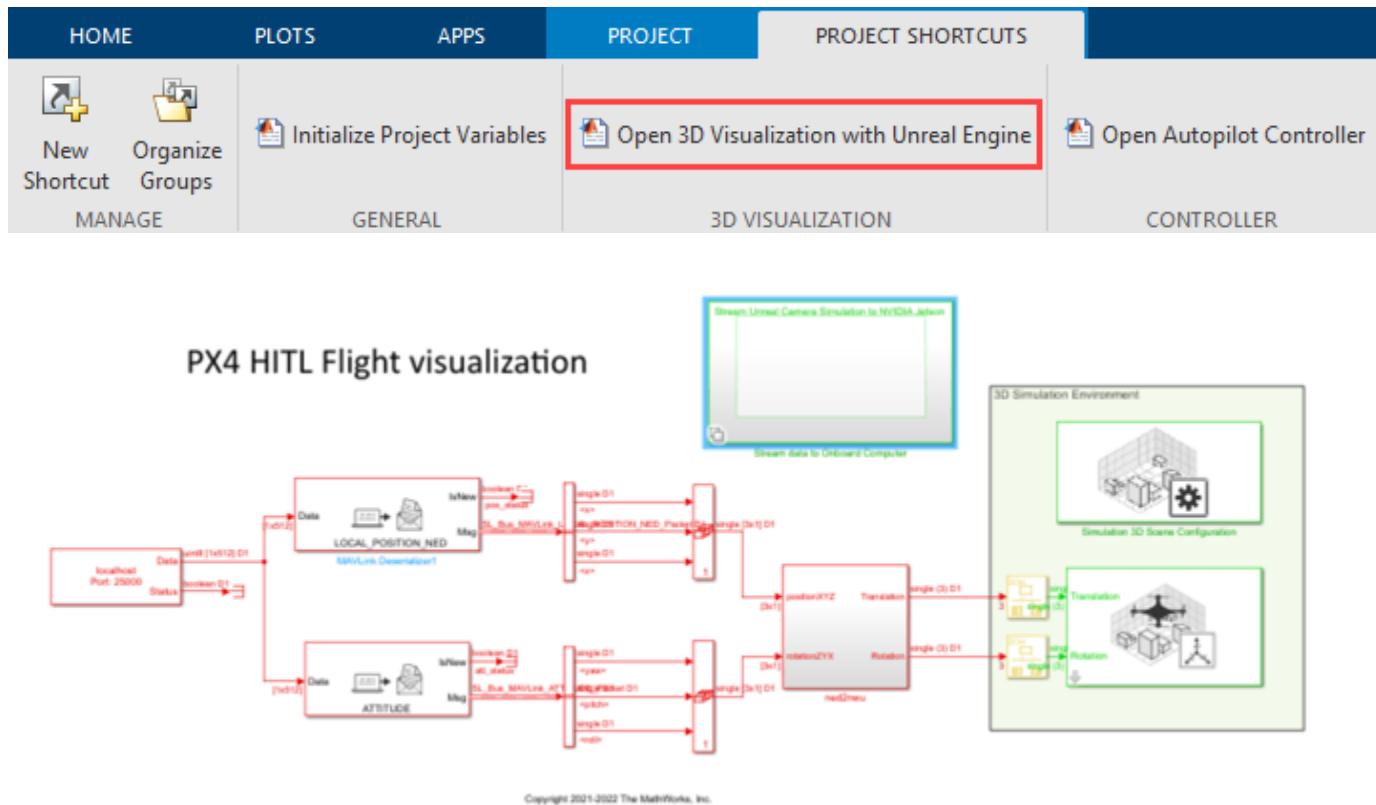
Note: Skip this step if you do not opt for flight visualization with PX4 HITL.

1. Open the second instance of the same MATLAB version. In this MATLAB session, The Simulink model for scenario simulation and flight visualization using unreal environment runs.

Ensure that your Host PC is Configured with MATLAB supported GPU (GPU with compute capability of more than 5).

2. Open the px4demo_HardwareInLoopWithSimulinkPlant MATLAB project.

3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open 3D Visualization with Unreal Engine** to launch the onboard model named *Unreal_3DVisualization*.

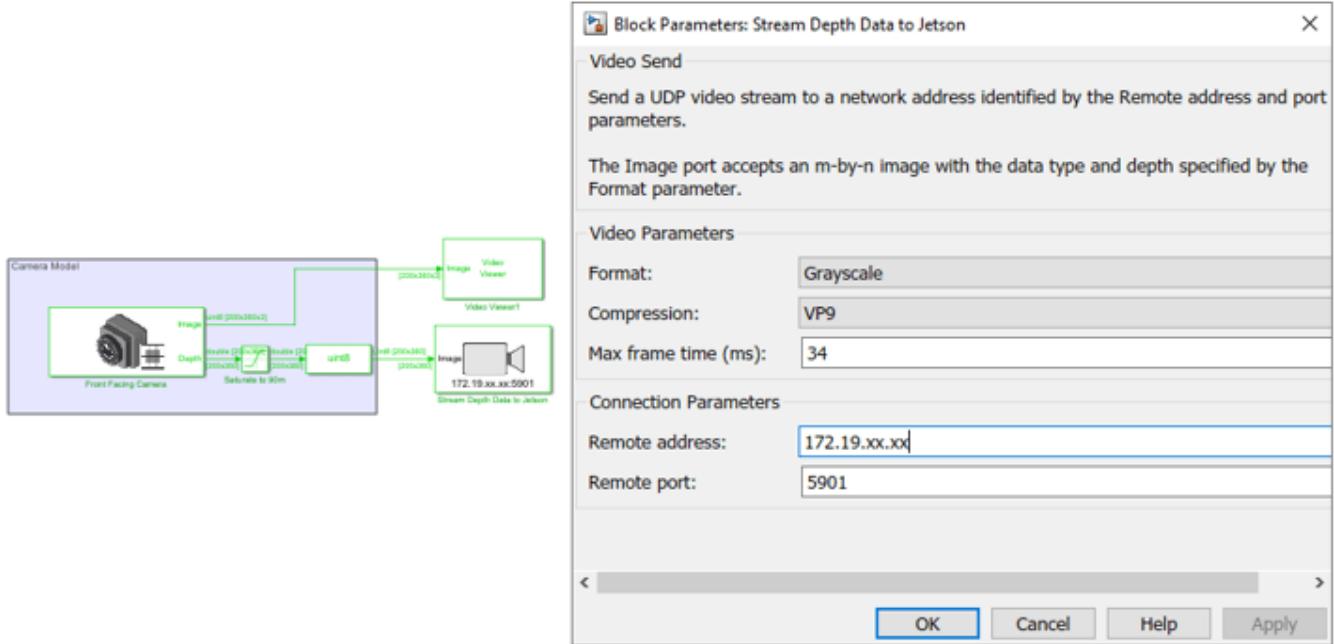


In this model, The MAVLink data from the PX4 Autopilot is received over UDP (port : 25000) and is used to decode the position and attitude data of the UAV. After coordinate conversion, it is then passed to the Simulation 3D UAV Vehicle block for flight visualization.

4. Enable the streaming of simulated depth sensor data in NVIDIA Jetson by updating the variable `enableOnboardStreaming` to 1.

5. The Simulation 3D Camera block provides the Camera image from the Unreal environment. In this example you stream the depth images from the camera block to NVIDIA Jetson using Video Send

block. Double-click block to open the block Parameters dialog box. Add the IP address of onboard computer (NVIDIA Jetson) in the dialog box and click **OK**.



6. On the **Simulation** tab, click **Run** to simulate the model. Once the model starts running, you will see the Unreal simulation environment getting launched. A sample screen is shown below.

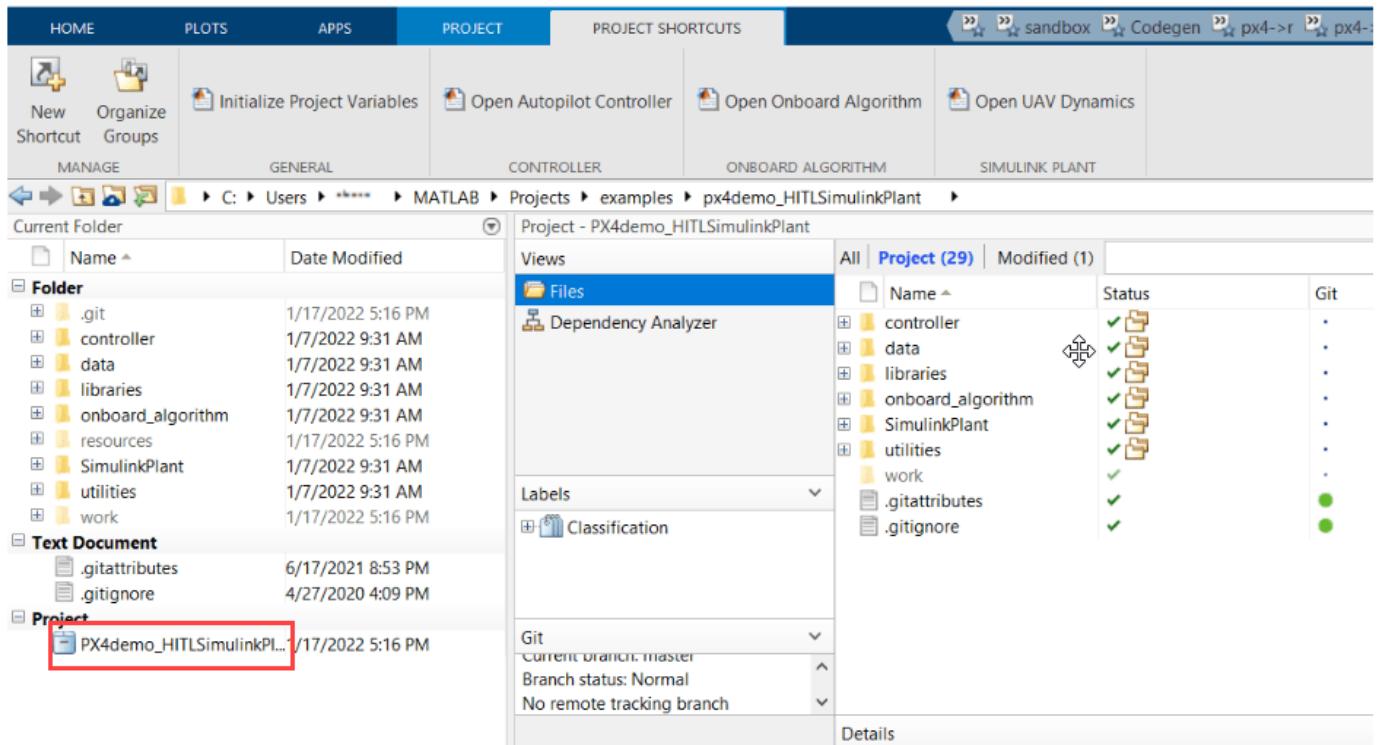


Step 5: Launch Third Session of MATLAB and the Onboard Model

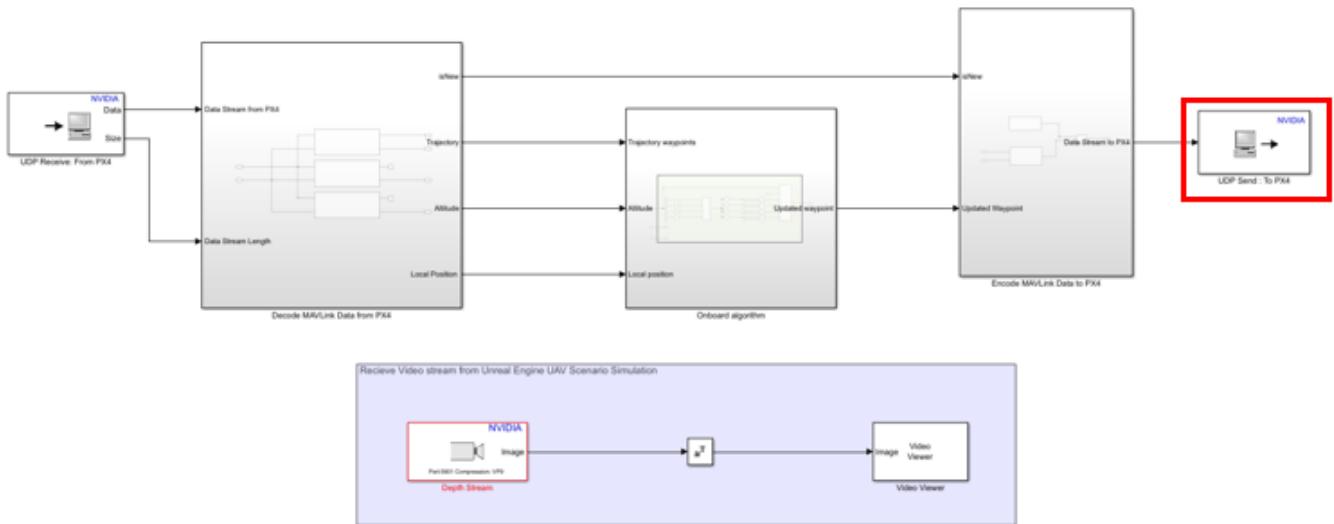
1. Open the third instance of the same MATLAB version.
2. Navigate to the project path previously copied in Step 2 in current MATLAB.

```
fx >> cd C:\Users\██████\MATLAB\Projects\examples\px4demo_HITLSimulinkPlant
```

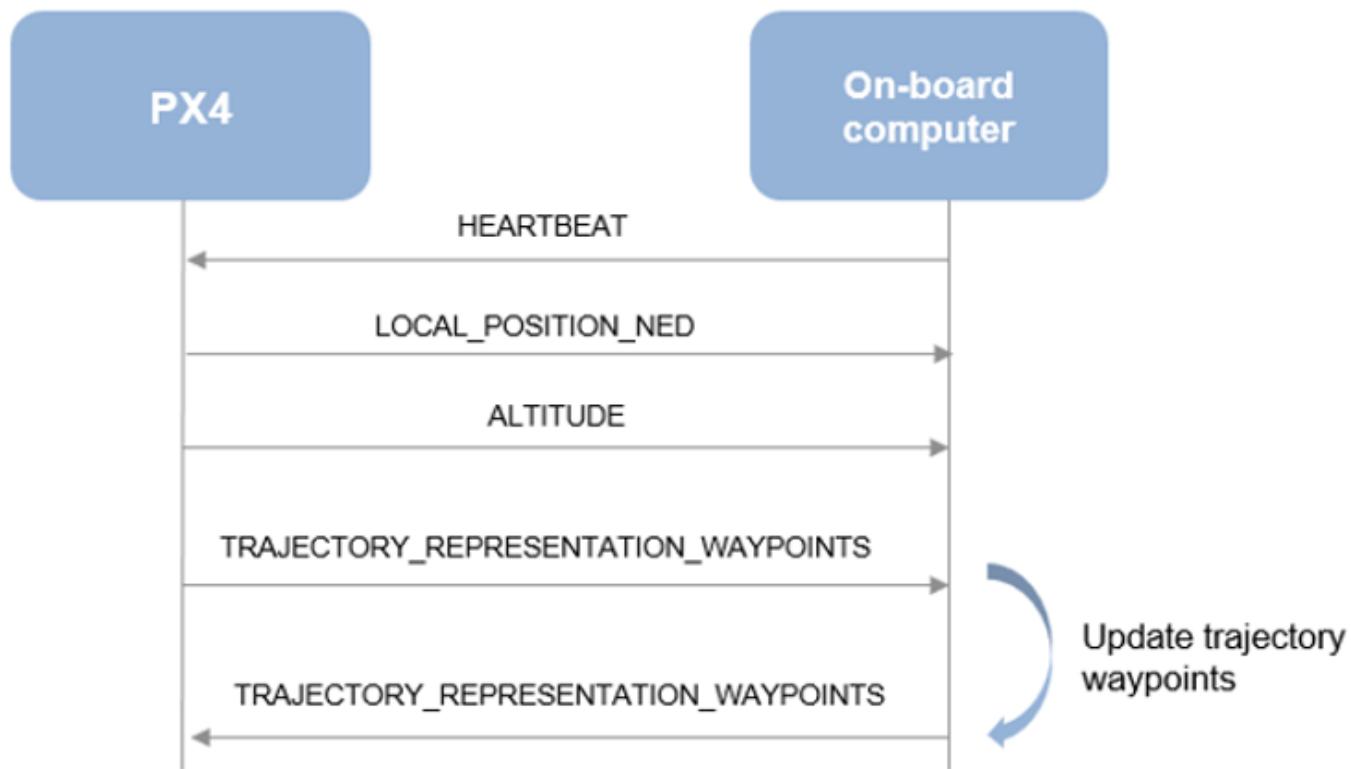
3. Click on the .prj file to launch the same Project in current MATLAB.



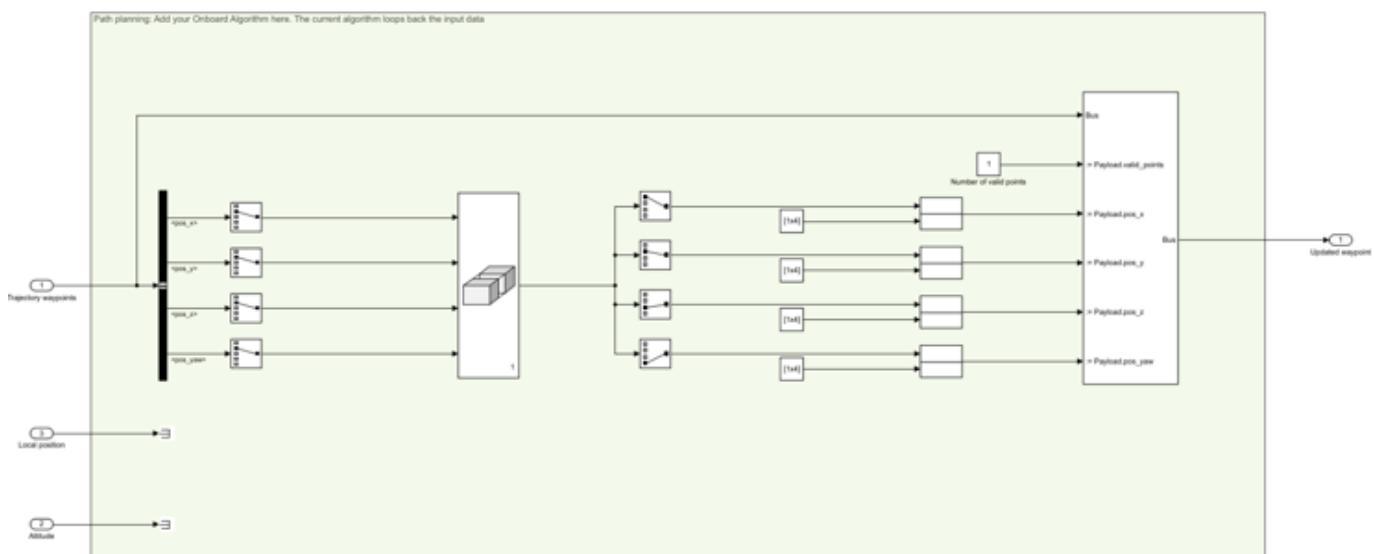
4. In the **Project Shortcuts** tab, click **Open Onboard Algorithm** to launch the onboard model named *Onboard_Autopilot_Communication*.



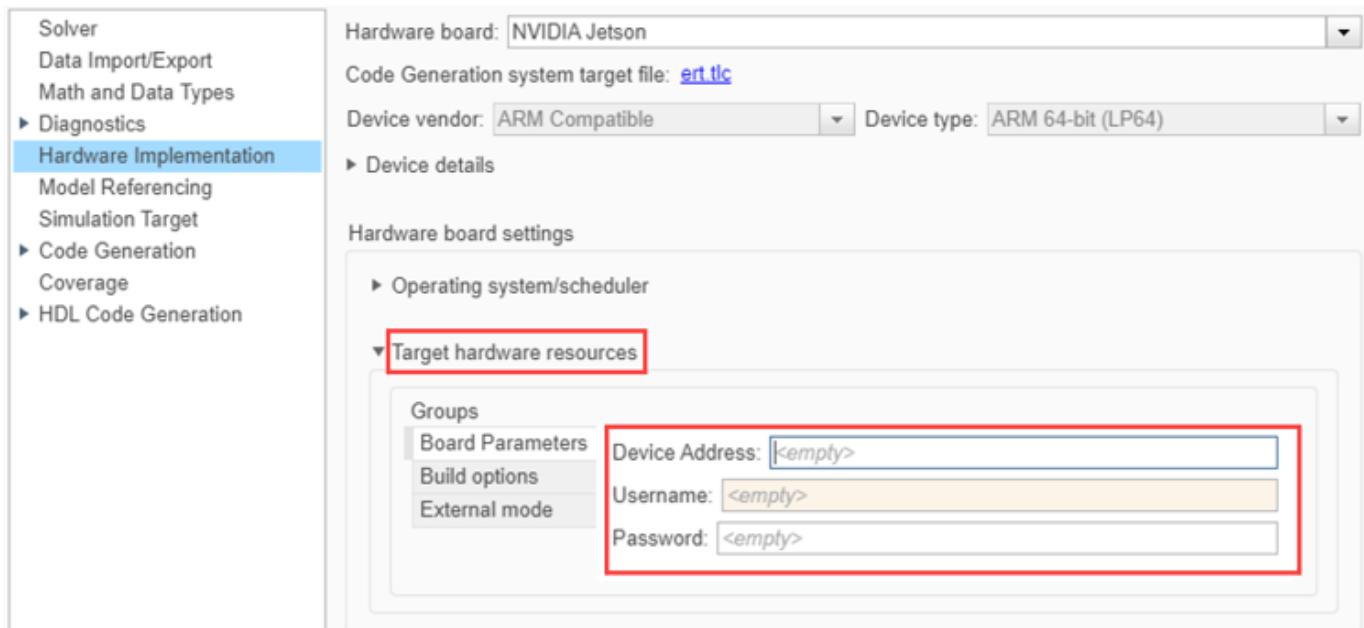
This model implements the PX4 path planning interface using MAVLink Serializer and MAVLink Deserializer blocks. For more information, see [PX4 Path Planning Interface](#). The MAVLink messages that are exchanged as part of this interface are shown in the following diagram.



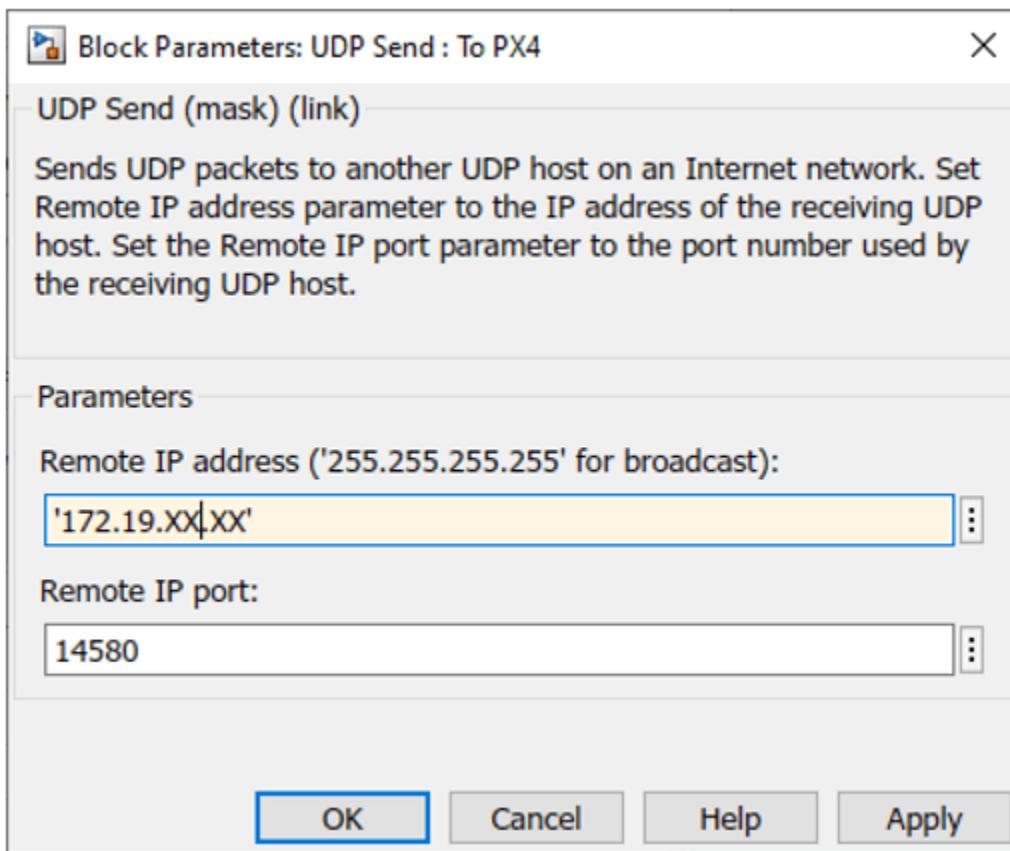
5. This example provides a basic onboard logic, which is sending back the received waypoint as the updated waypoint. You can consider designing your own onboard logic after completing this basic example.



6. Navigate to Target hardware resource > Board Parameters, enter the IP address of the NVIDIA Jetson and your login credentials.



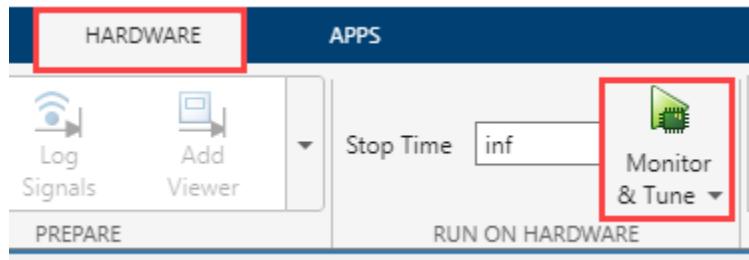
7. Double-click the UDP Send block to open the block Parameters dialog box. Enter the IP address of the host PC on which you are running UAV_Dynamics_Autopilot_Communication as Remote IP address. Enter the value for Port as 14580, as this is the port setting used in MAVLink Bridge block for onboard connection. Ensure that the host computer and the onboard computer are connected to same network.



8. Configure the Network Video Receive block to receive the depth data from Unreal environment. Note that the port number and compression parameters in the Network Video Receive block are same as those of the corresponding Video Send block streaming camera image from Unreal Engine.

Note: If you do not opt for flight visualization with PX4 HITL, comment the Network Video Receive block.

9. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolbar.



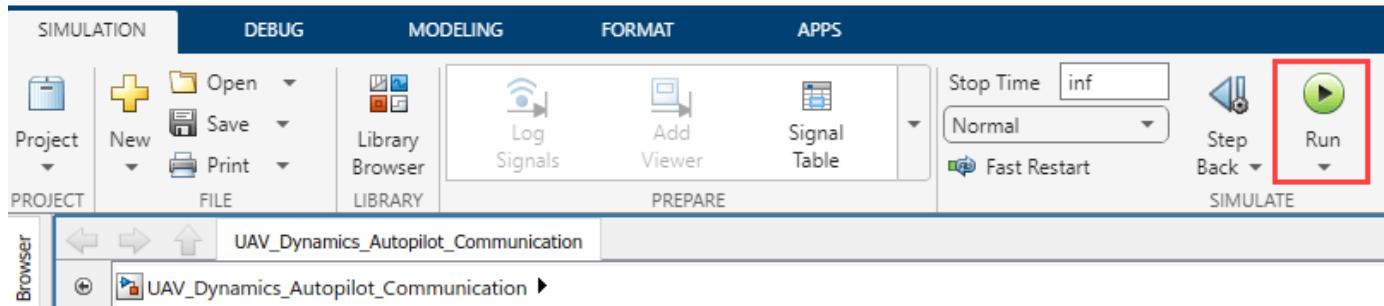
The code will be generated for the Controller model and the same will be automatically deployed to NVIDIA Jetson. NVIDIA Jetson should start communicating with host over Monitor & Tune Simulation.

Note: Ensure that there are no other deployed models from Simulink are running in NVIDIA Jetson. If you are unable to verify, reboot the Pixhawk hardware board before starting the deployment.

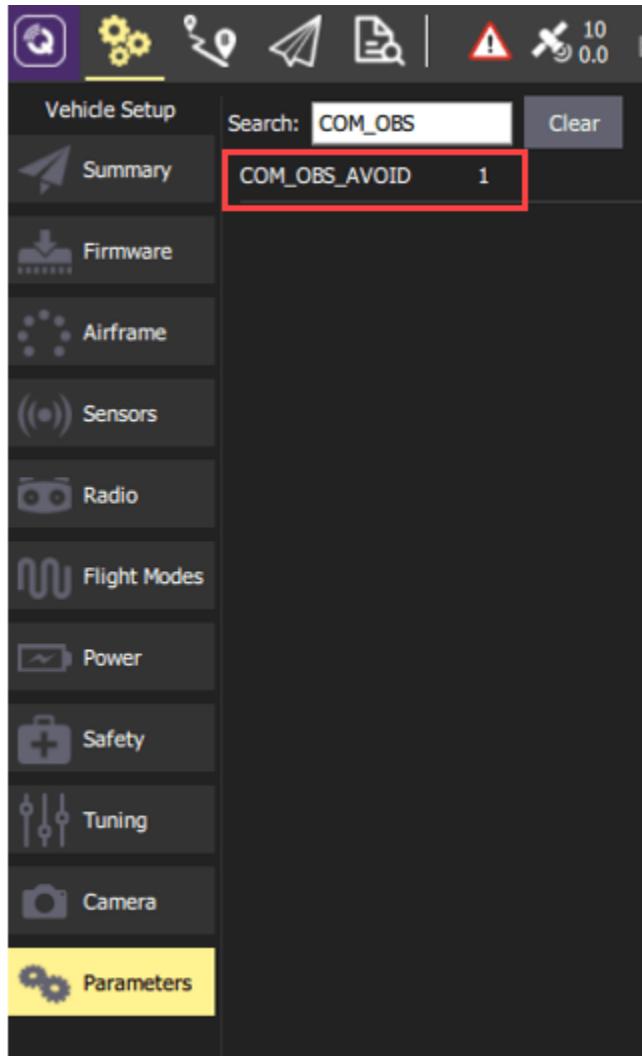
The algorithm in NVIDIA Jetson also communicates with the Plant model *UAV_Dynamics_Autopilot_Communication* over UDP.

Step 6: Run the UAV Dynamics Model, Upload Mission from QGroundControl and Fly the UAV

1. In the Simulink toolbar of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model. Once the plant model starts running, in QGroundControl connection will be established.

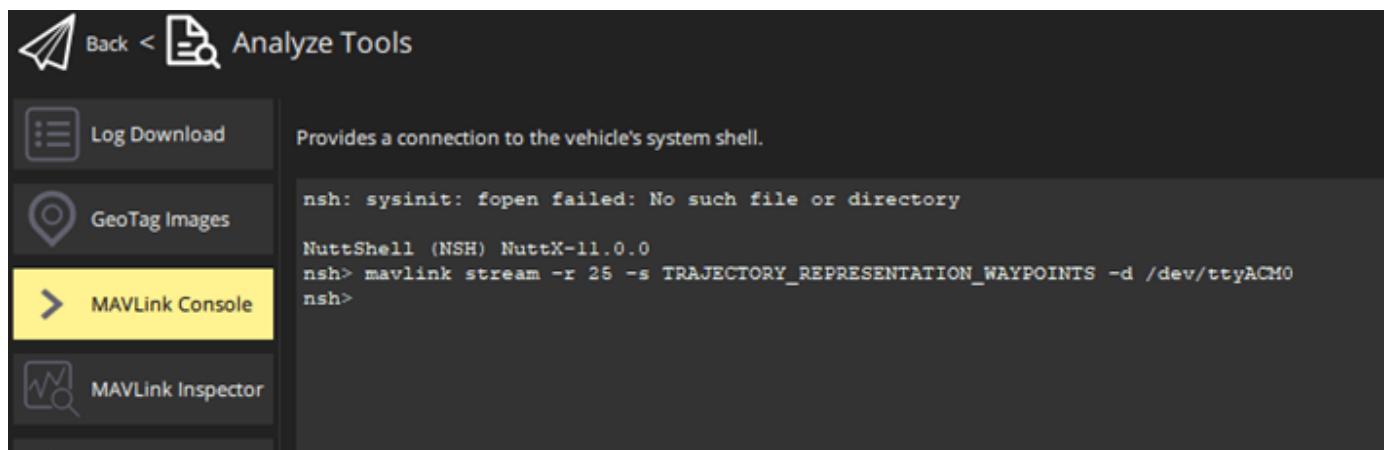


2. Set the PX4 parameter **COM_OBS_AVOID** enabling the PX4 path planning interface. Navigate to **Parameters** from the main menu and set the **COM_OBS_AVOID** parameter value to 1.

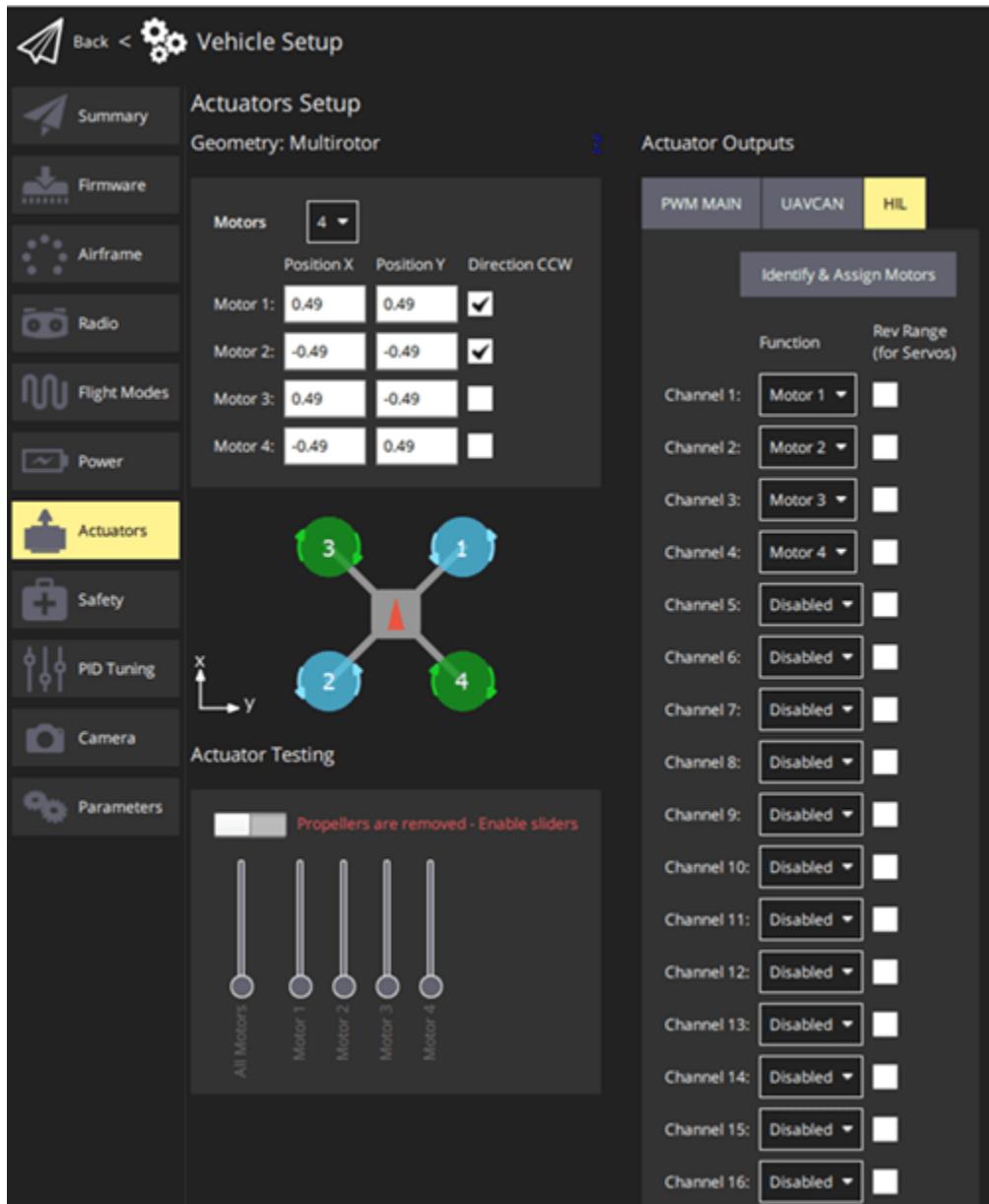


3. In the **Parameters** tab, set the **COM_DISARM_PRFLT** parameter value to -1.
4. Start MAVlink Stream for **TRAJECTORY_REPRESENTATION_WAYPOINTS**. This MAVLink message stream is required for onboard connectivity and is not started by default on USB (/dev/ttyACM0). To start the stream, run the following Command in the QGC MAVLink shell.

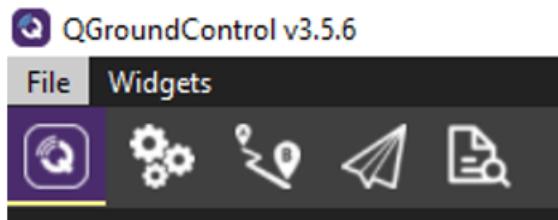
```
mavlink stream -r 25 -s TRAJECTORY_REPRESENTATION_WAYPOINTS -d /dev/ttyACM0
```



5. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



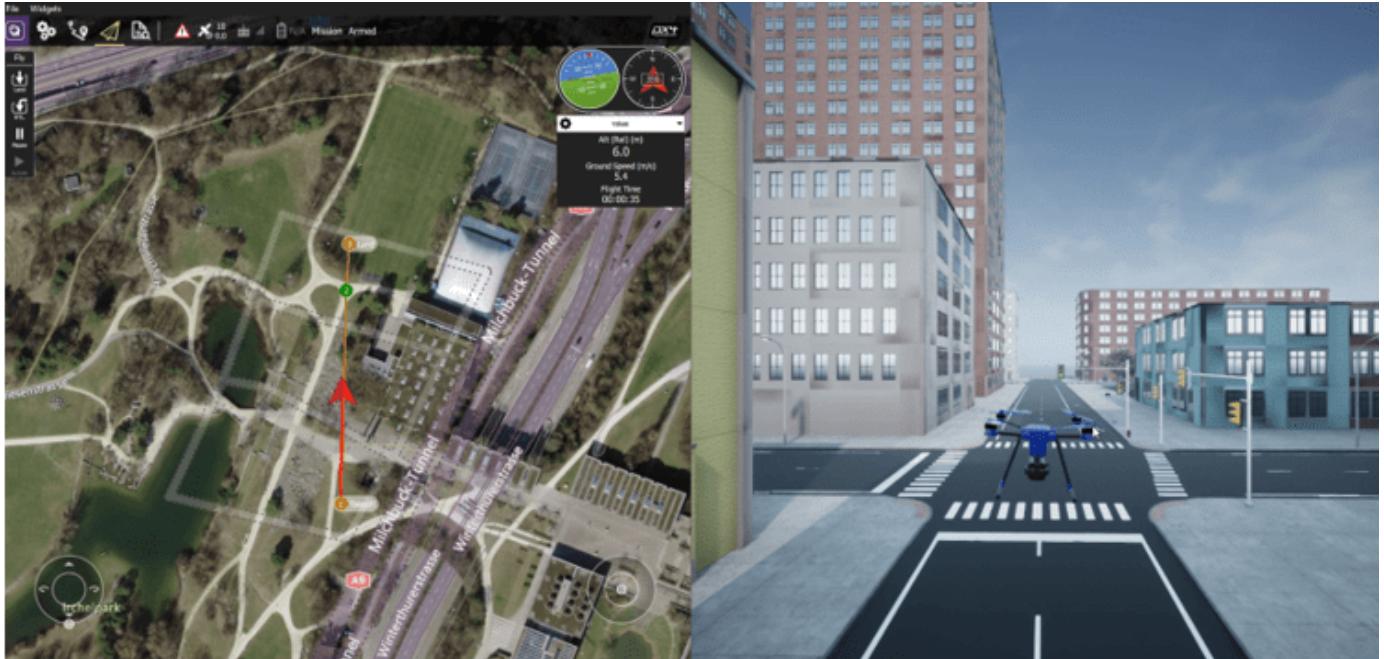
6. In the QGC, navigate to the *Plan View*.



7. Create a mission. For information on creating a mission, see Plan View.

After you create a mission, it is visible in QGC.

8. Click Upload button in the QGC interface to upload the mission from QGroundControl.
9. Navigate to *Fly View* to view the uploaded mission.
10. Start the Mission in QGC. The UAV should follow the mission path.
11. Observe the flight visualization in the Unreal Engine.



12. Go to the *Onboard Autopilot Communication* model and validate the depth image streamed to Jetson during the flight in the Video Viewer (from Computer Vision Toolbox™).



Troubleshooting

- While Simulating the visualization model in Step 4, you might get any STD exception errors such as, "some module could not be found".

Change the compiler to Microsoft Visual C++ 2019 using `mex -setup c++` command to fix the issue.

- "Avoidance system not available" warning when starting a mission. This warning occurs because the communication between NVIDIA Jetson and PX4 HITL is not established.

Ensure that host PC and NVIDIA Jetson are connected to same network. Try pinging NVIDIA Jetson from the host PC and vice versa. Also, double-check the NVIDIA Jetson IP address entered in the MAVLink Bridge blocks as mentioned in 6th point of Step 2 and Host PC IP address entered in the UDP send block as mentioned in 7th point of Step 5.

- When you start the Mission the vehicle is not taking off.

Firstly, verify if you are receiving the 'TRAJECTORY_REPRESENTATION_WAYPOINTS' message in the Onboard model. If the 'isNew' output for 'TRAJECTORY_REPRESENTATION_WAYPOINTS' in the MAVLink Deserializer Block related to is always false, it indicates that the message is not being received.

In this case, start MAVLink Stream for TRAJECTORY_REPRESENTATION_WAYPOINTS. This MAVLink message stream is required for onboard connectivity and is not started by default on USB (/dev/ttyACM0). To start the stream, run the following Command in the QGC MAVLink shell.

```
mavlink stream -r 25 -s TRAJECTORY_REPRESENTATION_WAYPOINTS -d /dev/ttyACM0
```

CAN Communication with Pixhawk Using Raspberry Pi

This example shows how to establish a CAN communication between Pixhawk® hardware and Raspberry Pi® hardware.

Introduction

The UAV Toolbox Support Package for PX4® Autopilots provides CAN Receive and CAN Transmit blocks that you can use to receive and send data to the CAN port on the Pixhawk Series flight controller. Similarly, Simulink® Support Package for Raspberry Pi Hardware provides CAN Receive and CAN Transmit blocks that you can use to receive and send CAN data using a MCP2515-based CAN shield. This example uses CAN blocks to establish a CAN communication between Pixhawk hardware and Raspberry Pi hardware.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- For more information on how to configure the Raspberry Pi board for CAN communication, see Transmit and Receive Data Using Raspberry Pi CAN Blocks.

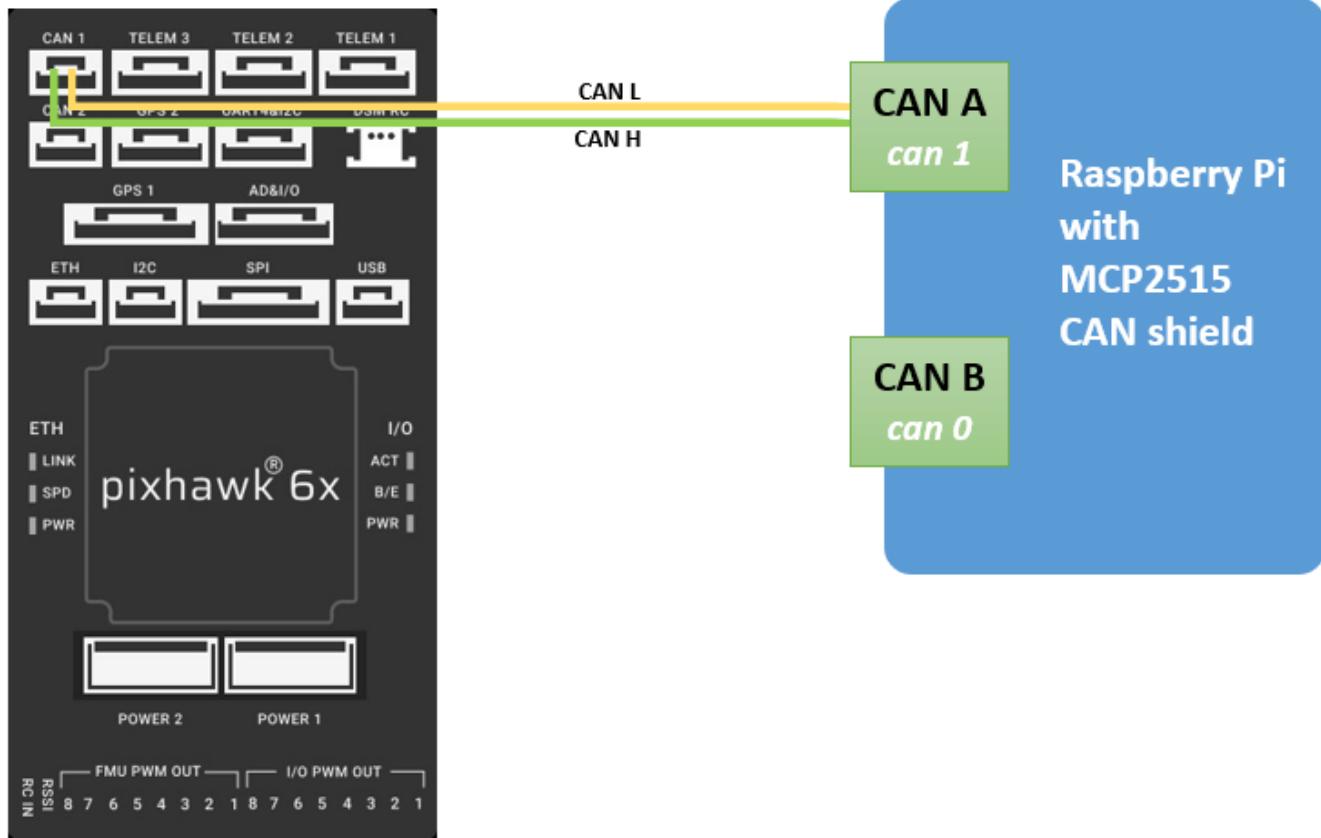
Required Hardware

To run this example, you will need the following hardware:

- Supported PX4 Autopilot
- Raspberry Pi Hardware
- MCP2515-based CAN shield (for example, PiCAN 2 Duo)
- Pixhawk connectors
- Connecting wires

Hardware Setup

- Mount PiCAN 2 Duo MCP2515 based CAN-Bus Shield on Raspberry Pi board.
- Power up and Connect Raspberry Pi to the same Network. You must be able to ping the Raspberry Pi from the Host PC.
- Connect Pixhawk standard 4 pin connector to the CAN 1 port of Pixhawk 6x.
- Connect The Pixhawk 6x to Host PC via USB.
- Use a properly terminated custom cable to link the CANH and CANL pins of the Pixhawk 6x to the CANH and CANL pins of Raspberry Pi shield.



Example Workflow

This example uses two different Simulink models.

- Simulink model for CAN Communication to be deployed on PX4 Autopilot.
- Simulink model for CAN Communication to be deployed on Raspberry Pi.

For better visualization of the logged data in Simulink Data Inspector, consider launching two separate sessions of same MATLAB®.

In first session of MATLAB, run the Pixhawk model in Monitor and tune Simulation (External mode).

In second session of MATLAB, run the Raspberry Pi model in Monitor and tune Simulation (External mode).

Step 1: Configure and Launch the Pixhawk CAN Communication Simulink Model

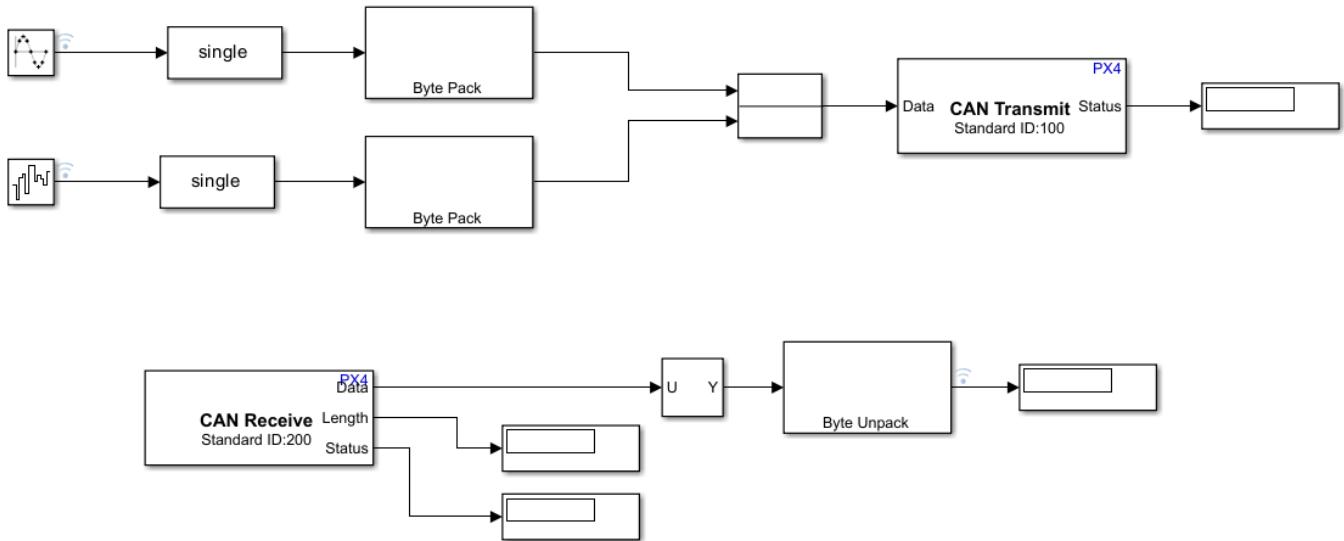
In this model, two signals (Sine wave and random noise) of data type `single` are sent as CAN messages. The Byte Pack block is used to convert signals of data types to a `uint8` vector output. The message ID:100 is transmitted using CAN Transmit block and message ID:200 is received on CAN Receive block. The Byte Unpack block is used to convert a vector of `uint8` data type to signals of data type `single`.

Note: To avoid conflict with PX4 UAVCAN, disable UAVCAN before working with PX4 CAN blocks. To disable, set the PX4 parameter `UAVCAN_ENABLE` to 0. For more information, see [Finding/Updating Parameters](#).

Perform these steps to configure and launch the model.

1. Open MATLAB.
2. Open the example `px4demo_CAN_Pixhawk` model.

CAN Communication with Raspberry Pi using PX4 CAN Blocks

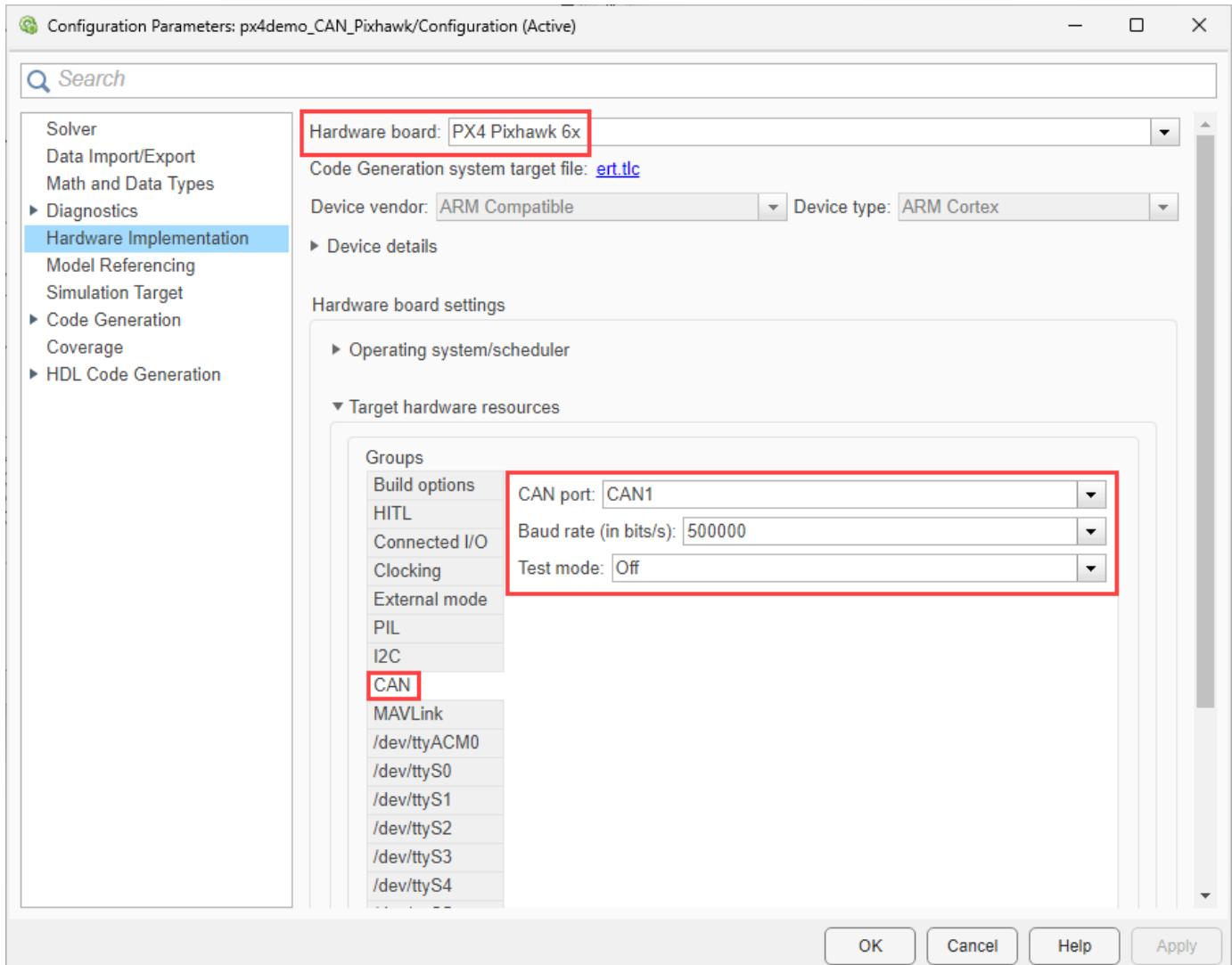


The example model is pre-configured for Pixhawk 6x. The model shows how to send multiple signals (for example, random noise and Sine wave) as a CAN message to the CAN bus.

3. Configure the model.

Note: Steps to configure the model is not required in the pre-configured model. Perform these step if you have changed the hardware or not using the pre-configured model.

- On the **Modeling** tab, click **Model Settings** to open the Configuration Parameters dialog box.
- Click **Hardware Implementation** and select the required PX4 hardware from the list in Hardware board parameter.
- Expand **Target hardware resources** for that board.
- Go to **CAN** tab.
- Set the Baud rate to **500000**.
- Note that CAN port is set to **CAN1** and Test mode is set to **Off**.



4. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolbar. Wait for the Simulation to start.

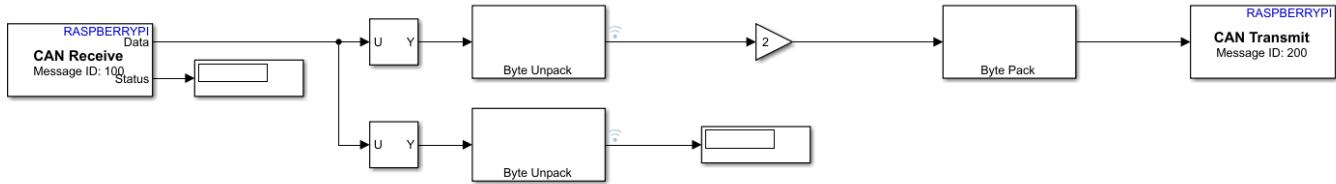
Step 2: Configure and Launch the Raspberry Pi CAN Communication Simulink Model

In this model, two signals (Sine wave and random noise) of data types vector of `uint8` are received as CAN messages. The Byte Unpack block is used to convert a vector of `uint8` data type to signals of data type `single`. In this example, only the Sine wave signal is transmitted back. The Gain block is used to multiply the input by a constant value. The Byte Pack block is used to convert signal of data types to a `uint8` vector output. The message ID:100 is received using CAN Receive block and message ID:200 is transmitted back to the Pixhawk model using the CAN Transmit block.

Perform these steps to configure and launch the model.

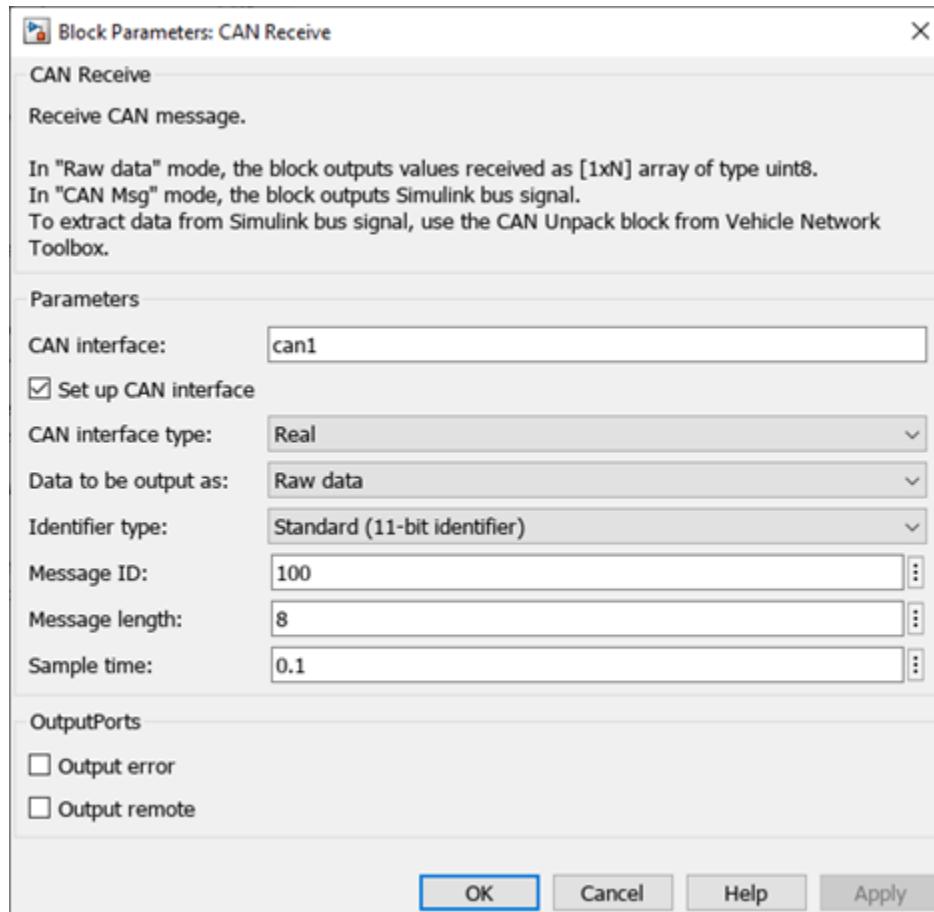
1. Open MATLAB.
2. Open the px4demo_CAN_RasPi model.

CAN Communication with PX4 Autopilot using Raspberry Pi CAN Blocks



3. Configure the Model for the Raspberry Pi Hardware. Enter the IP address of the Raspberry Pi, username, and password.

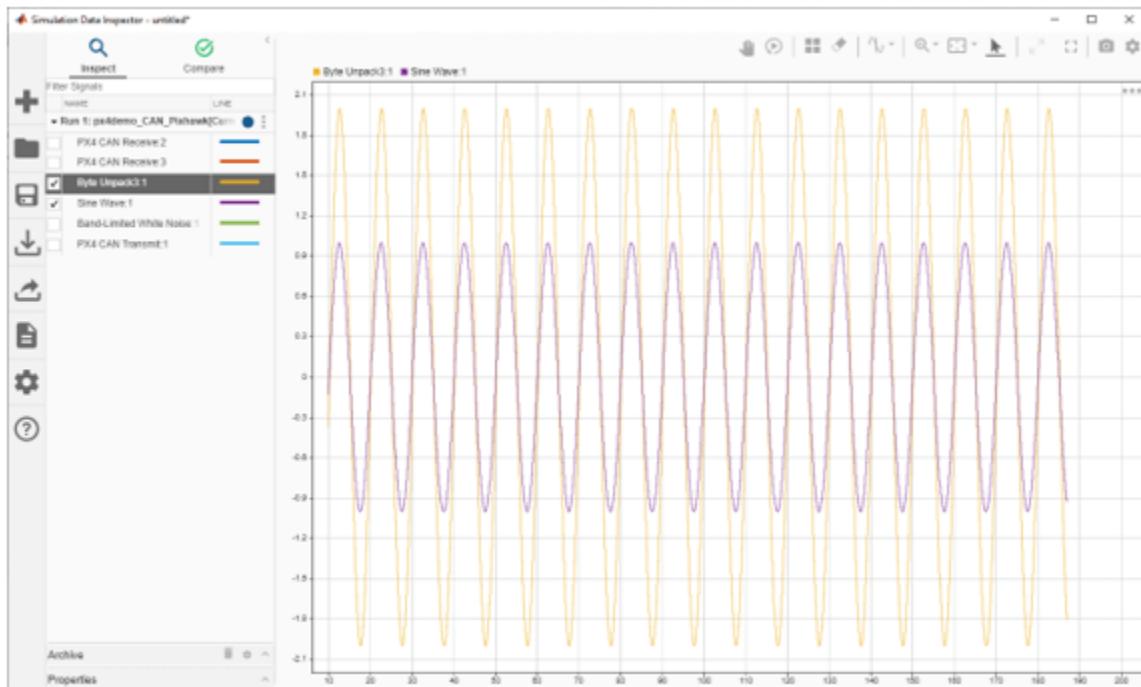
4. This example model uses **can1** as the **CAN interface**. This is the CAN A port in the CAN shield. To use CAN B port, use **can0** as the CAN interface.



5. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolstrip.

Step 3: Analyze the Signals Communicated

1. Observe the transmitted signals (random noise and Sine wave) and compare them with the signals received on the Raspberry Pi board.
2. The Sine wave is amplified with a gain of 2 and sent back to the Pixhawk model. Compare the Sine wave sent to the Raspberry Pi and the Sine wave received on Pixhawk. Change the gain value in the Raspberry Pi model and it gets reflected in the Received signal in Pixhawk.



Other Things to Try

Use NVIDIA® Jetson™ TX2 instead of Raspberry Pi. For information on getting started with CAN communication with NVIDIA Jetson, see CAN Bus Communication on NVIDIA Jetson TX2 in Simulink.

Transmit and Receive Data Using PX4 CAN Blocks

This example shows to use UAV Toolbox Support Package for PX4® Autopilots to transmit and receive data from the CAN network using the specified CAN device.

Introduction

In this example, the Transmitter side of the model uses PX4 CAN Transmit blocks to transmit data. The Receiver side of the model uses PX4 CAN Receive blocks to receive data.

Prerequisites

- If you are new to Simulink®, watch the Simulink Quick Start video.

Required Hardware

To run this example, you will need the following hardware:

- Supported PX4 Autopilot

Configure Model, Transmit Data, and Receive Data

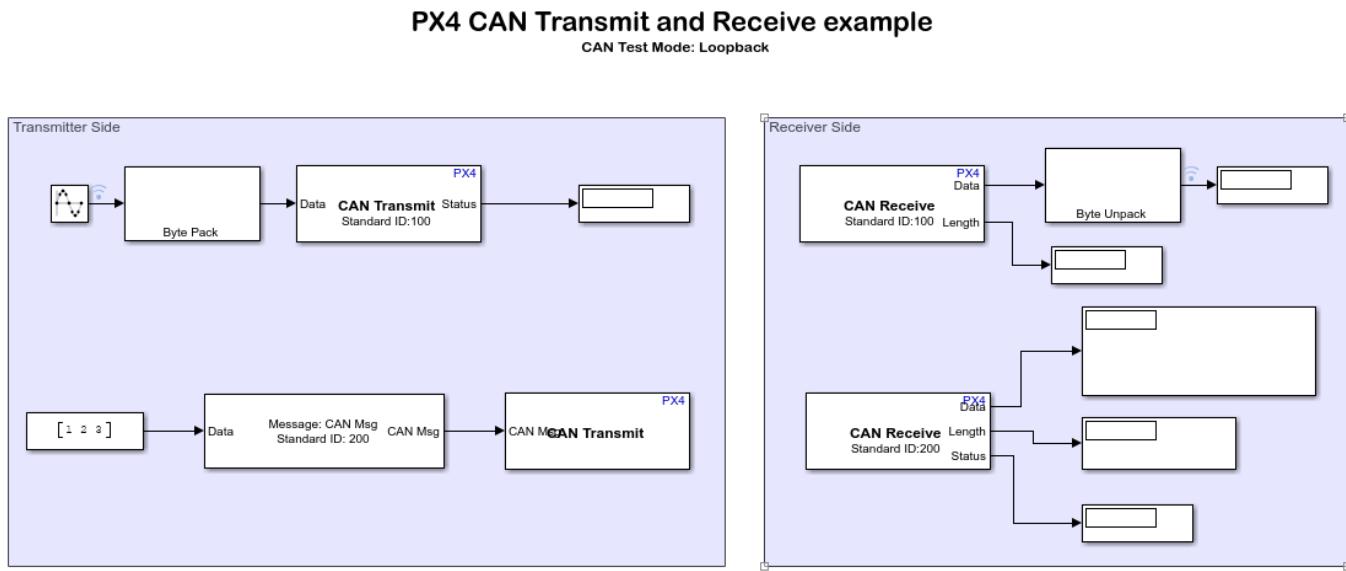
In this example, Sine wave of data type `double` is sent as a CAN message. Byte Pack block is used to convert signals of data types to a `uint8` vector output. Message ID:100 is transmitted to CAN Transmit block in the Transmitter Side of the model. In the Receiver side of the model, Message ID:100 is received by the CAN Receive block. The Byte Unpack block is used to convert a vector of `uint8` data type to signals of data types `double`.

Similarly, a constant value is sent as a CAN message. Message ID:200 is transmitted to CAN Transmit block in the Transmitter Side of the model and in the Receiver side of the model, Message ID:200 is received by the CAN Receive block.

Note: To avoid conflict with PX4 UAVCAN, disable UAVCAN before working with PX4 CAN blocks. To disable, set the PX4 parameter `UAVCAN_ENABLE` to 0. For more information, see [Finding/Updating Parameters](#).

Perform these steps to configure the model, transmit data, and receive data.

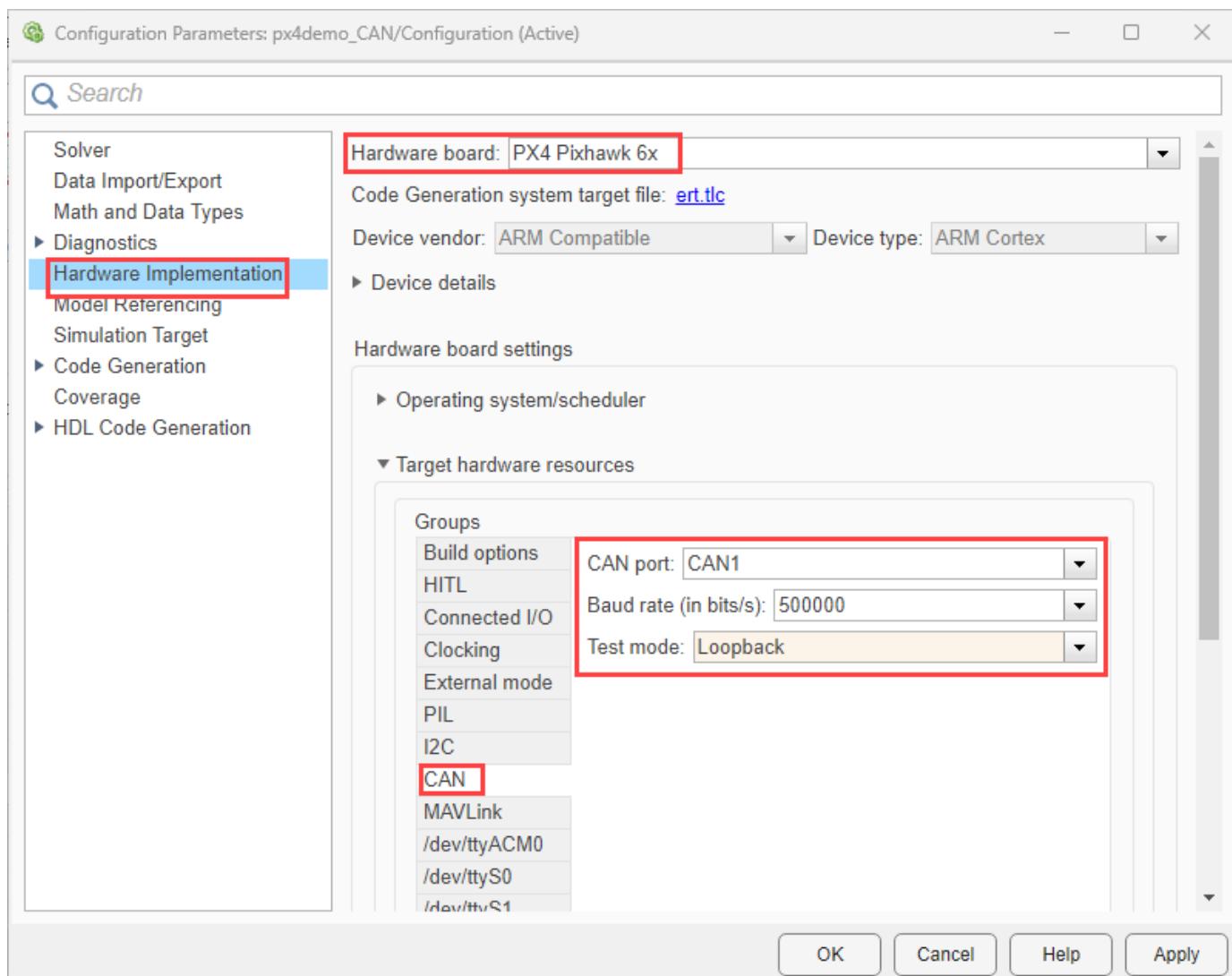
1. Open MATLAB®.
2. Open the example `px4demo_CAN` model.



3. Configure the model.

Note: Steps to configure the model is not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

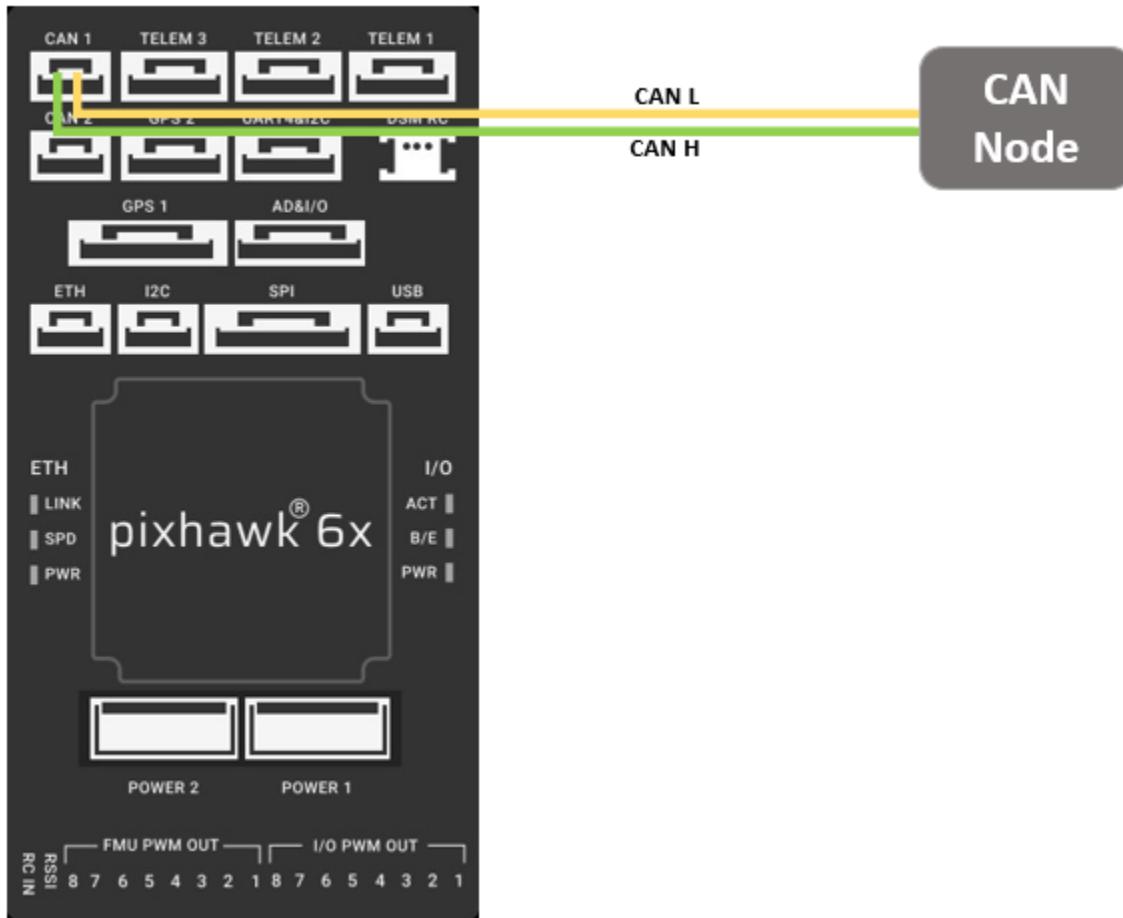
- Go to **Modeling > Model Settings** to open the Configuration Parameters dialog box.
- Click **Hardware Implementation** and select the required PX4 hardware from the list in Hardware board parameter.
- Expand **Target hardware resources** for that board.
- Go to **CAN** tab.
- Set the Baud rate to **500000**.
- Set the Test mode to **Loopback**. This enables loopback testing of CAN messages and allows you to send and receive CAN messages without connecting to the hardware.
- CAN port is set to **CAN1**. Only CAN1 is supported for CAN port.



4. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolbar. Wait for the Simulation to start. The Display block displays the data being received in the target hardware.

Try this Example with STM32H7 based PX4 Autopilots

If you are using a PX4 Autopilot that is based on the STM32H7 processor (such as Cube Orange, Cube Orange+, Pixhawk 6x, and Pixhawk 6c), it is essential to connect a CAN Node to the CAN bus and ensure that it is powered on. Without an external CAN node, both CAN Transmit and Receive operations will fail, even when in Loopback mode.



Other Things to Try

In this example, the identifier types used are Standard ID for both transmit and receive communications. Try using this example with Extended ID as the identifier type for one communication.

PX4 Hardware-in-the-Loop (HITL) Simulation with Fixed-Wing Plant in Simulink

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to verify the controller design by deploying the design on the PX4 Autopilot hardware board. This is done in HITL mode with fixed-wing UAV Dynamics contained in Simulink®.

This example provides a reference model that helps you to design a flight controller algorithm to perform take-off, waypoint-following, loitering, and landing for a fixed-wing UAV plant model. Additionally, the example also provides a plant model which captures aspects such as aerodynamics, propulsion, ground contact, and sensors. This example demonstrates the following three workflows.

- Follow a mission provided from QGC.
- Manual Control with RC Joystick. For more information, see “PX4 Hardware-in-the-Loop (HITL) Simulation with Manual Control for Fixed-Wing Plant in Simulink”.
- Execute a mission hardcoded in Simulink. For more information, see “PX4 Hardware-in-the-Loop (HITL) Simulation with Fixed-Wing Plant and Hardcoded Mission in Simulink”.

For information on fixed-wing plant model architecture, guidance logic, and conventions, see “Fixed-Wing Plant and Controller Architecture”.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Go through the “PX4 Hardware-in-the-Loop System Architecture” document to understand the physical connections required to setup the PX4 Autopilot and Simulink for HITL Simulation.
- Configure and set up PX4 Autopilot in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any PX4 Autopilot as the PX4 Autopilot board and corresponding build target having `_fixedwing` keyword from the drop-down list. For example, if you are using Cube Orange as autopilot board then select `cubepilot_cubeeorange_fixedwing` as the build target.

Required Third-Party Software

- QGroundControl (QGC)

Required Hardware

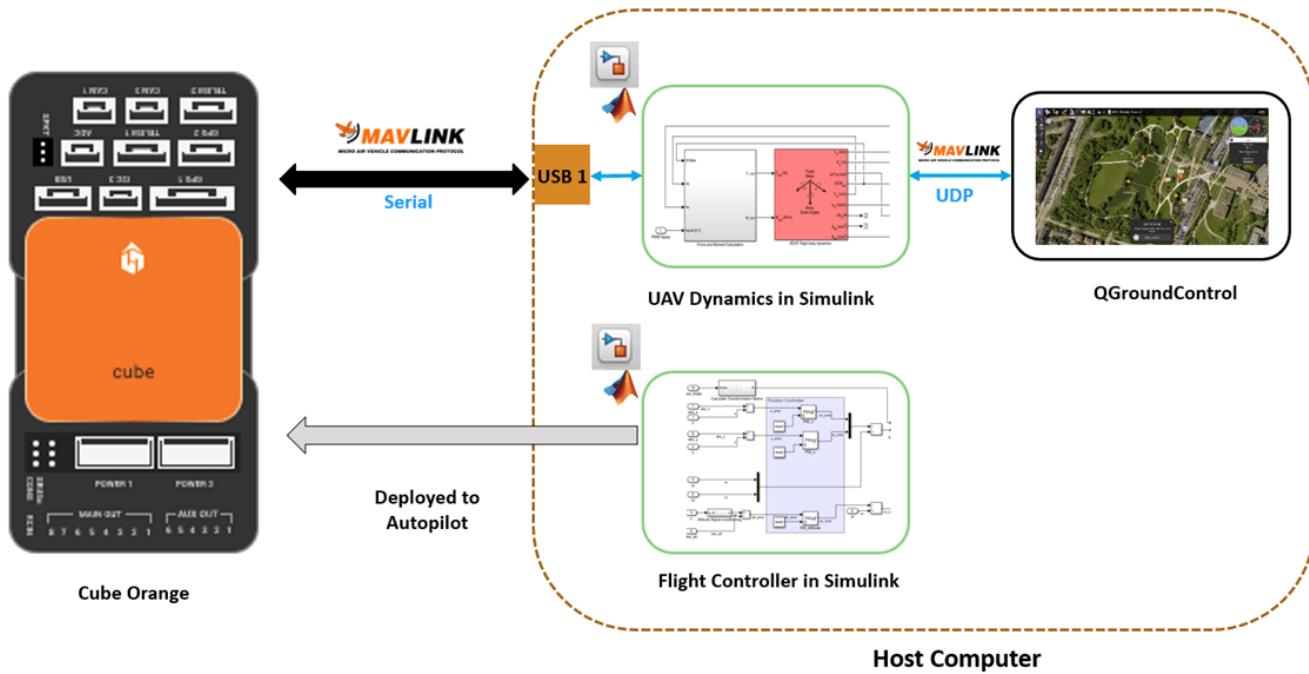
- PX4 Autopilot flight controller
- Micro USB (Universal Serial Bus) type-B cable
- Micro-SD card

Run HITL Simulation

Note: Avoid running this example on a virtual machine. The virtual machine might display an error saying “unable to pace at specified rate” and the drone might crash.

Step 1: Make Hardware Connections and Set Up the PX4 Autopilot in HITL Mode

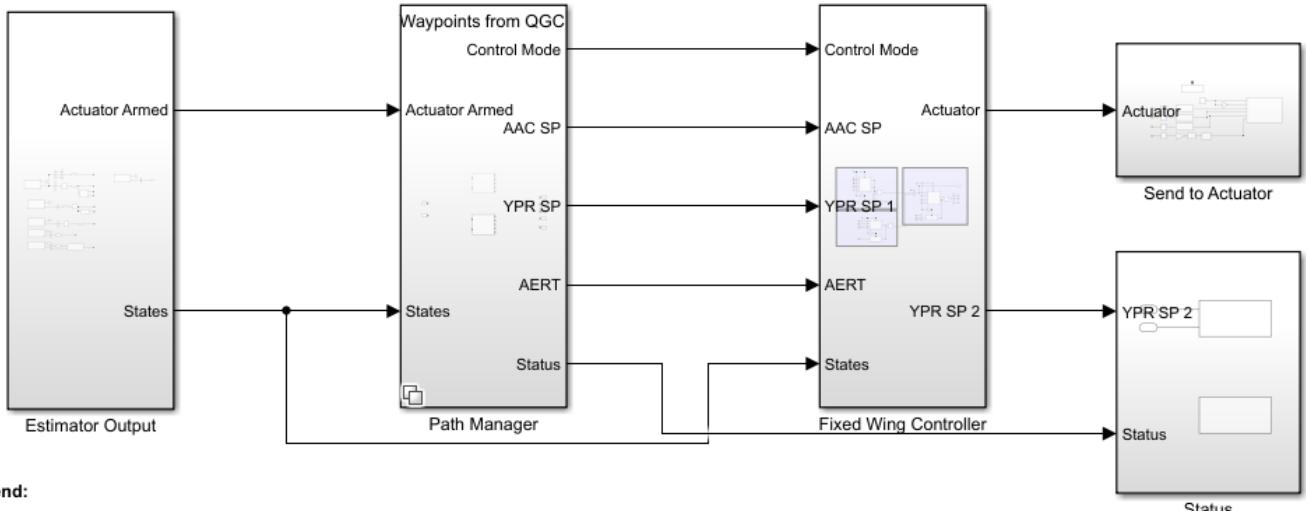
This diagram shows the HITL setup and the physical communication between various modules.



1. Connect your PX4 Autopilot board to the host computer using the USB cable.
2. Configure the PX4 Autopilot board in HITL mode as described in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
3. Set up the PX4 Firmware as described in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.

Step 2: Open MATLAB Project

1. Open MATLAB®.
2. Open the px4demo_HardwareInLoopWithSimulinkFixedWingPlant MATLAB project.
3. Once you open the project, click the **Project Shortcuts** tab on the MATLAB toolbar and click **Launch Autonomous Control Model** to launch the *FixedWingGNC* controller.

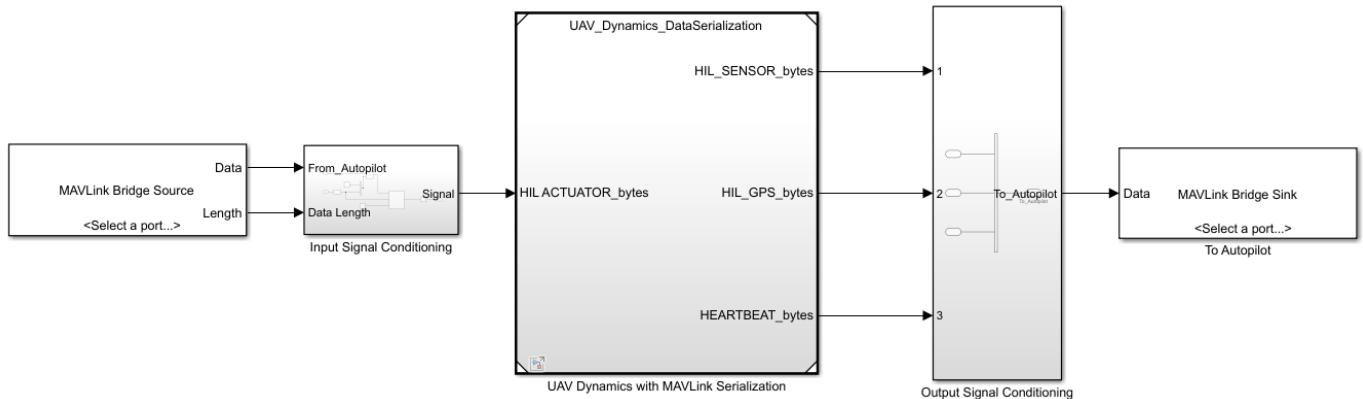


4. In the **FixedWingGNC controller** model, on the **Hardware** tab, in the **Mode** section, select **Run on board (External mode)**.

5. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab.

The code will be generated for the Controller model and the same will be automatically deployed to the PX4 Autopilot board.

6. In the **Project Shortcuts** tab, click **Launch UAV Dynamics** to launch the Simulink plant model named **UAV_Dynamics_Autopilot_Communication**.



7. In the Simulink plant model, set the serial ports in MAVLink Bridge Source and MAVLink Bridge Sink blocks.

Step 3: Configure Simulink Controller Model for HITL Mode

1. The controller model is configured to run in HITL mode. If you are not using this example model, follow the instructions as described in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”.

Note: These steps are not required in the pre-configured model. Complete these steps if you have changed the hardware or not using the pre-configured model.

2. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model.

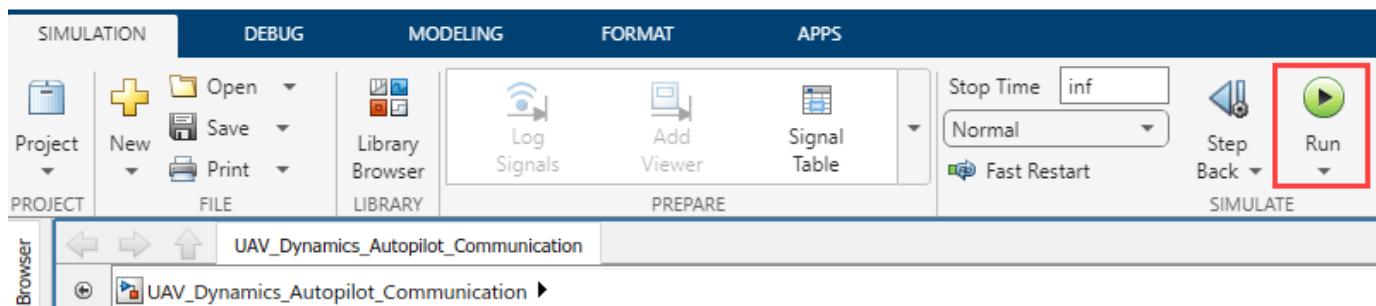
This deploys the controller code on the PX4 Autopilot hardware and launches the QGroundControl (QGC).

Note: If you are using Ubuntu, QGC might not launch automatically. To launch QGC, open Terminal and go to the folder where QGC is downloaded and run the following command:

```
./QGroundControl.AppImage
```

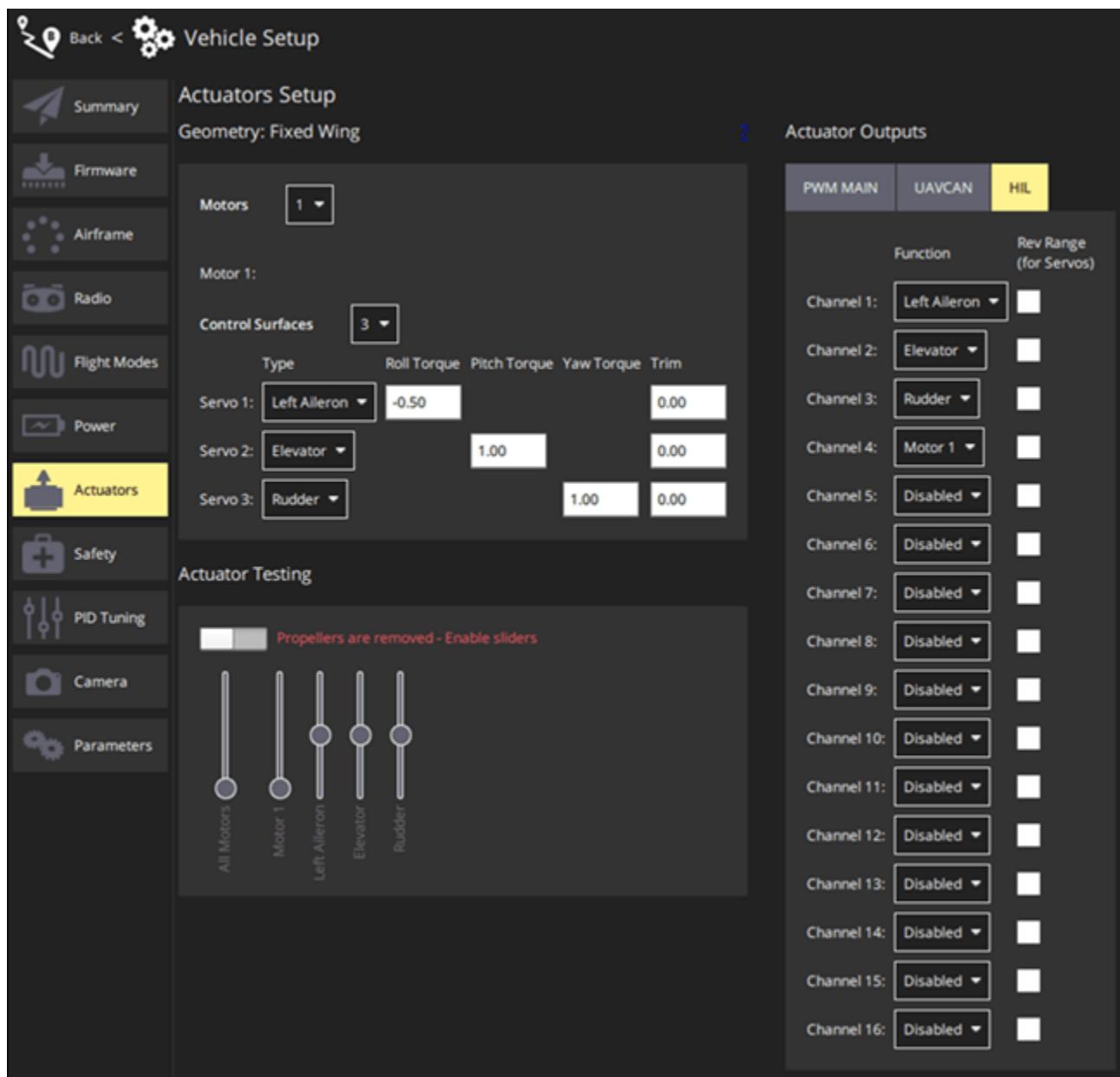
Step 4: Run the UAV Dynamics Model, Upload Mission from QGroundControl and Fly UAV

1. In the Simulink toolstrip of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model.

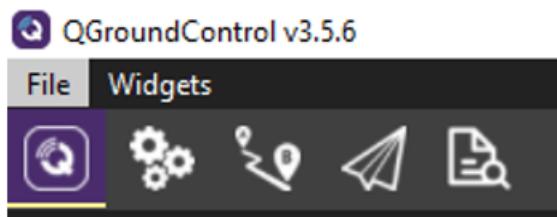


2. Set the parameter **COM_DISARM_PRFLT** value to **-1**, in QGC.

3. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



4. In the QGC, navigate to the *Plan View*.

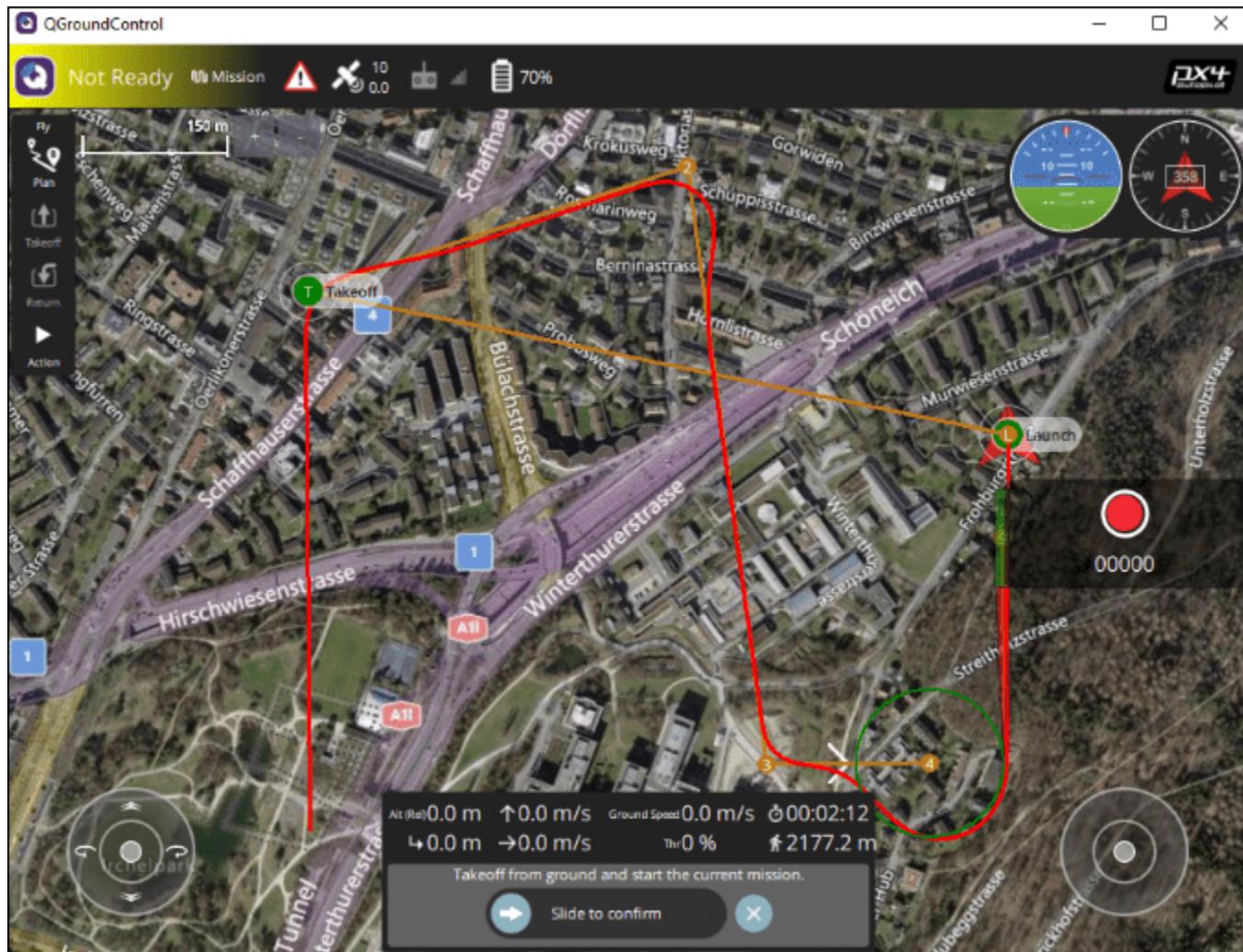


5. This example provides a preplanned mission *SampleMission.plan* that you can upload to QGC. In the QGC, navigate and select *SampleMission.plan* file available in data folder of the MATLAB project.

After you upload the plan, the mission is visible in QGC.

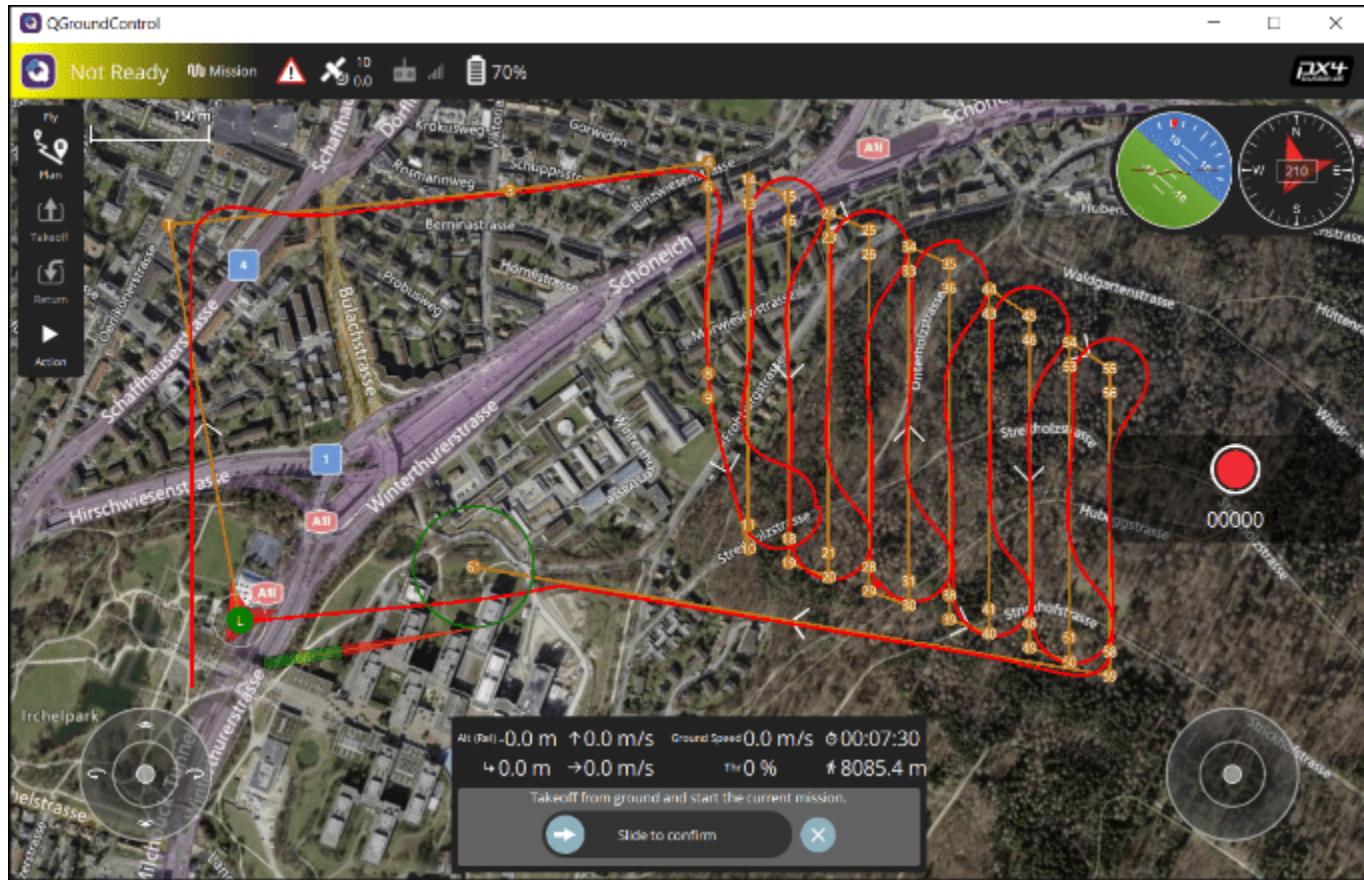
6. Click **Upload** button in the QGC interface to upload the mission from QGroundControl.
7. Navigate to *Fly View* to view the uploaded mission.
8. Start the mission in QGC. The UAV should follow the mission path.

After the mission is started, the drone takes off and follows the set of waypoints from QGC. A sample screen is shown here.



Other Things to Try

- Try running complex missions from QGC. A sample survey mission trajectory is shown here.



- Add wind disturbances to the plant model.

Obstacle Avoidance in NVIDIA Jetson with PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics Modeled in Simulink

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to verify an Obstacle avoidance algorithm deployed on NVIDIA® Jetson™ as Onboard Computer along with Pixhawk® hardware board, in HITL mode and the UAV Dynamics contained in Simulink®.

The Obstacle Avoidance block provided as part of UAV Toolbox is an autonomous algorithm that computes an obstacle-free direction using visual sensor data and destination position. For such computationally intensive algorithm, you can use an Onboard Computer on the drone along with the Autopilot. NVIDIA Jetson is a commonly used Onboard Computer for drones. Using Simulink, you can design a complex Autonomous algorithm and deploy the same on NVIDIA Jetson.

This example also showcases 3D scenario simulation during the flight and streaming the image to the onboard computer using the simulated camera sensor. Unreal Engine® simulation environment is used for the 3D scenario Simulation and visualization. This image can be received in NVIDIA Jetson and can be used by the Obstacle avoidance block.

In summary, in this example you,

- Deploy an Obstacle avoidance algorithm on NVIDIA Jetson that takes vision data from Unreal and provides path correction to PX4.
- Enable flight visualization with PX4 HITL and stream simulated camera image to NVIDIA Jetson.
- Run and complete a UAV mission with onboard computer.

Limitation: The Unreal Engine simulation environment is supported only on Microsoft® Windows® system. If you are using a Linux® system, skip adding the 3D scenario simulation step and still be able to complete this example.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Go through the “PX4 Hardware-in-the-Loop System Architecture” document to understand the physical connections required to set up the Pixhawk and Simulink for HITL Simulation.
- Configure and set up Pixhawk in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Set up the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any Pixhawk Series board as the PX4 Autopilot board and corresponding build target having `_multicopter` keyword from the drop-down list. This example uses Pixhawk 6x as autopilot board and `px4_fmu-v6x_multicopter` build target.
- Familiarize with the co-simulation framework for UAVs, see “Unreal Engine Simulation for Unmanned Aerial Vehicles”

It is recommended to understand the “PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with UAV Dynamics in Simulink” on page 1-214 example.

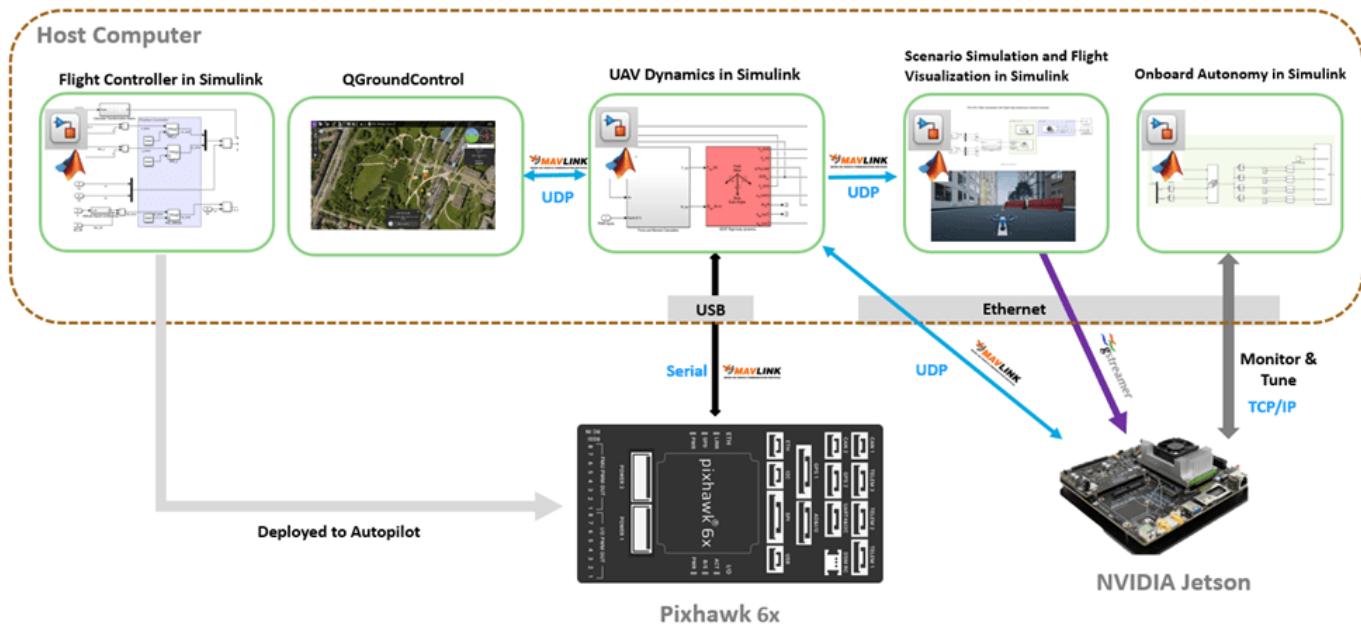
Required Third-Party Software This example requires this third-party software:

- QGroundControl (QGC)

Required Hardware To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card
- Pixhawk serial port connectors
- NVIDIA Jetson & power adaptor
- Host PC Configured with “GPU Computing Requirements” (Parallel Computing Toolbox). It is recommended to use GPU with compute capability of more than 5.

Workflow to Run Model on NVIDIA Jetson Along with Pixhawk in HITL Mode



The above diagram illustrates the PX4 and NVIDIA Jetson HITL setup and the physical communication between various modules.

This example uses four different Simulink models.

- Simulink model for Flight Controller to be deployed on PX4 Autopilot.
- Simulink model for Obstacle avoidance algorithm to be deployed on NVIDIA Jetson.
- Simulink model for Autonomous algorithm to be deployed on NVIDIA Jetson.
- Simulink model for flight visualization with Unreal Engine Simulation for Unmanned Aerial Vehicles.

To avoid performance degradation in MATLAB® due to three different Simulink models running at the same time, launch three separate instances of same MATLAB.

- In the first session of MATLAB, the Flight Controller is deployed on Autopilot and the UAV Dynamics model will run on host computer communicating with Autopilot.
- In the second session of MATLAB, the Simulink model for flight visualization with Unreal Engine Simulation will be running. This can be skipped if you opt to not add the flight visualization.
- In the third session of MATLAB, the Simulink model for NVIDIA Jetson communicates with MATLAB on host computer using Monitor & Tune Simulation.

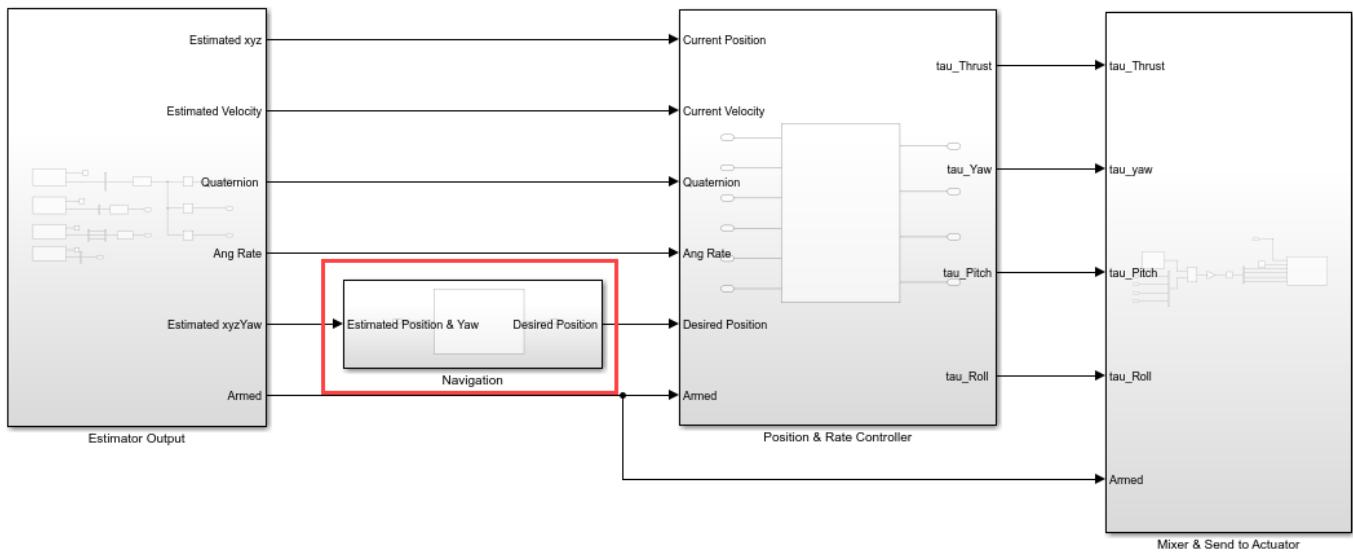
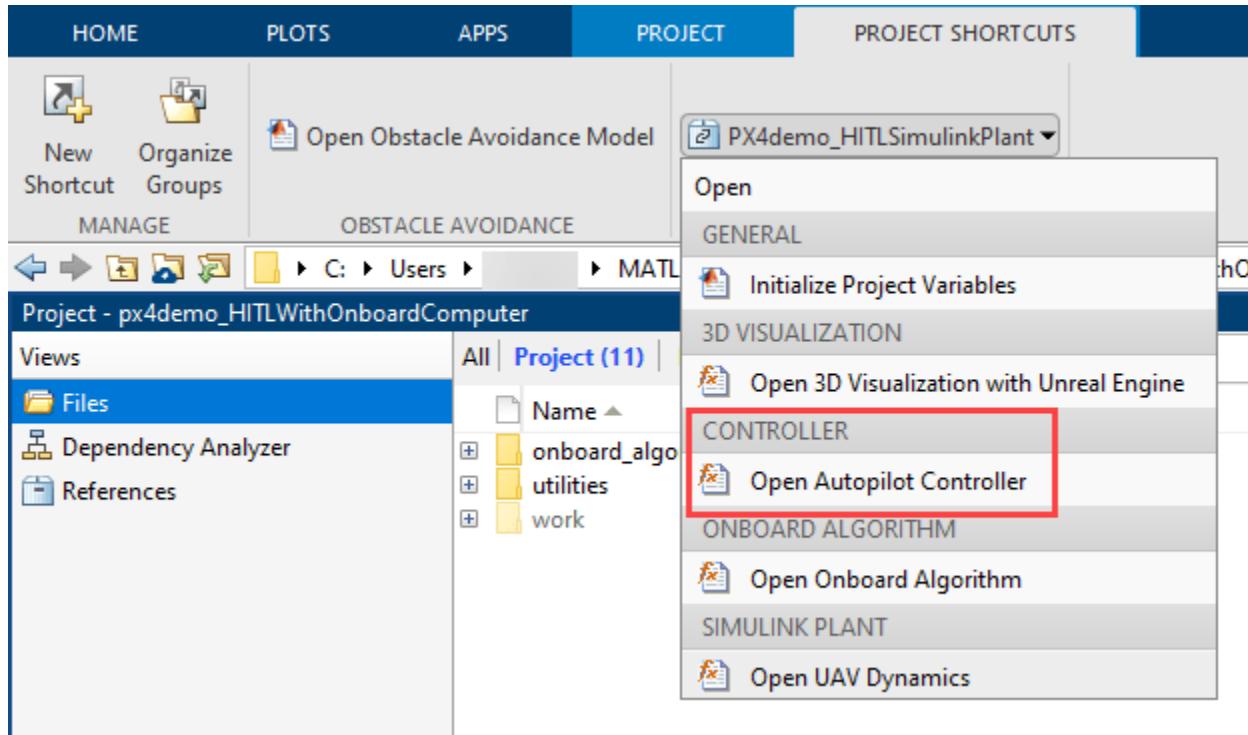
Step 1: Make Hardware Connections and Set Up the Pixhawk in HITL Mode

1. Connect your Pixhawk board to the host computer using the USB cable.
2. Ensure that you have configured the Pixhawk board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
3. Ensure that you have setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.
4. Setup and configure your NVIDIA Jetson on network using “MATLAB Coder Support Package for NVIDIA Jetson and NVIDIA DRIVE Platforms” (MATLAB Coder).

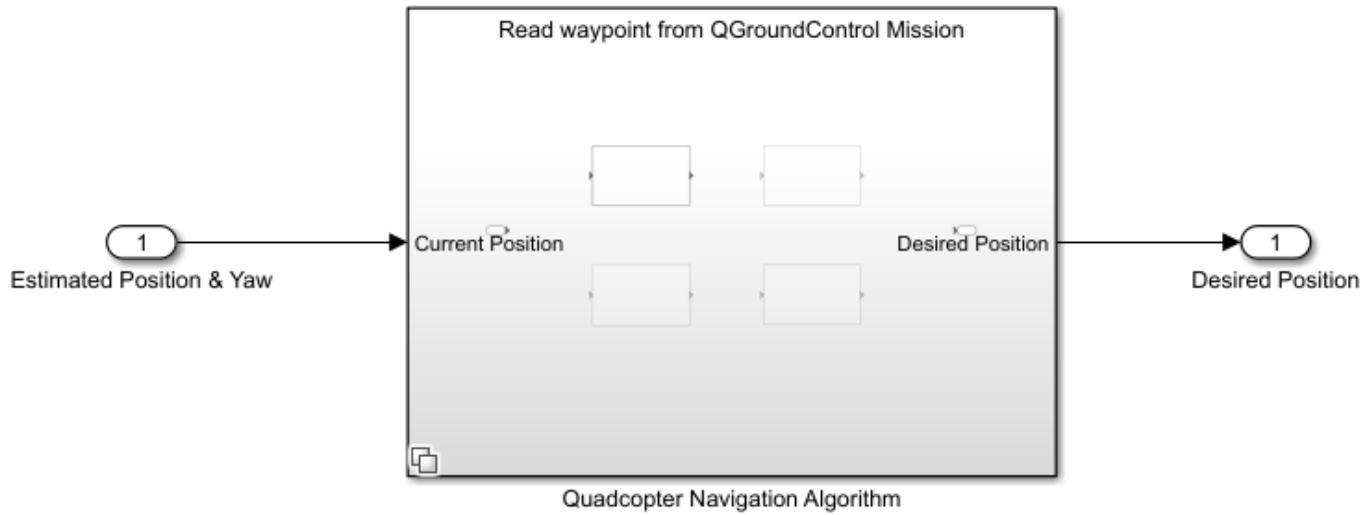
Step 2: Launch First Session of MATLAB and the MATLAB Project

The support package includes an example MATLAB project having the PX4 flight controller and the UAV to follow the mission set in the QGroundControl (QGC).

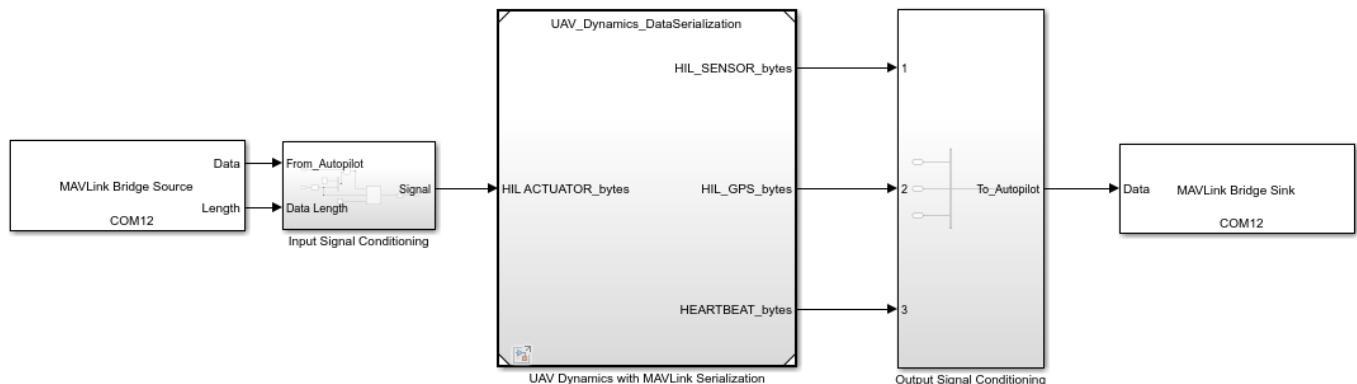
1. Open MATLAB.
2. Open the px4demo_HITLWithOnboardComputer MATLAB project.
3. The Simulink project uses *PX4demo_HITLSimulinkPlant* as reference project. The UAV dynamics and autopilot controller are provided by the *PX4demo_HITLSimulinkPlant* project. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open Autopilot Controller** from the *PX4demo_HITLSimulinkPlant* drop-down list to launch PX4 Controller named *Quadcopter_ControllerWithNavigation*.



4. Navigate to Navigation subsystem. This is a “Implement Variations in Separate Hierarchy Using Variant Subsystems” (Simulink) with `guidanceType` as the variant control variable. Define `guidanceType = 1` in the global workspace to choose the navigation subsystem for this example.

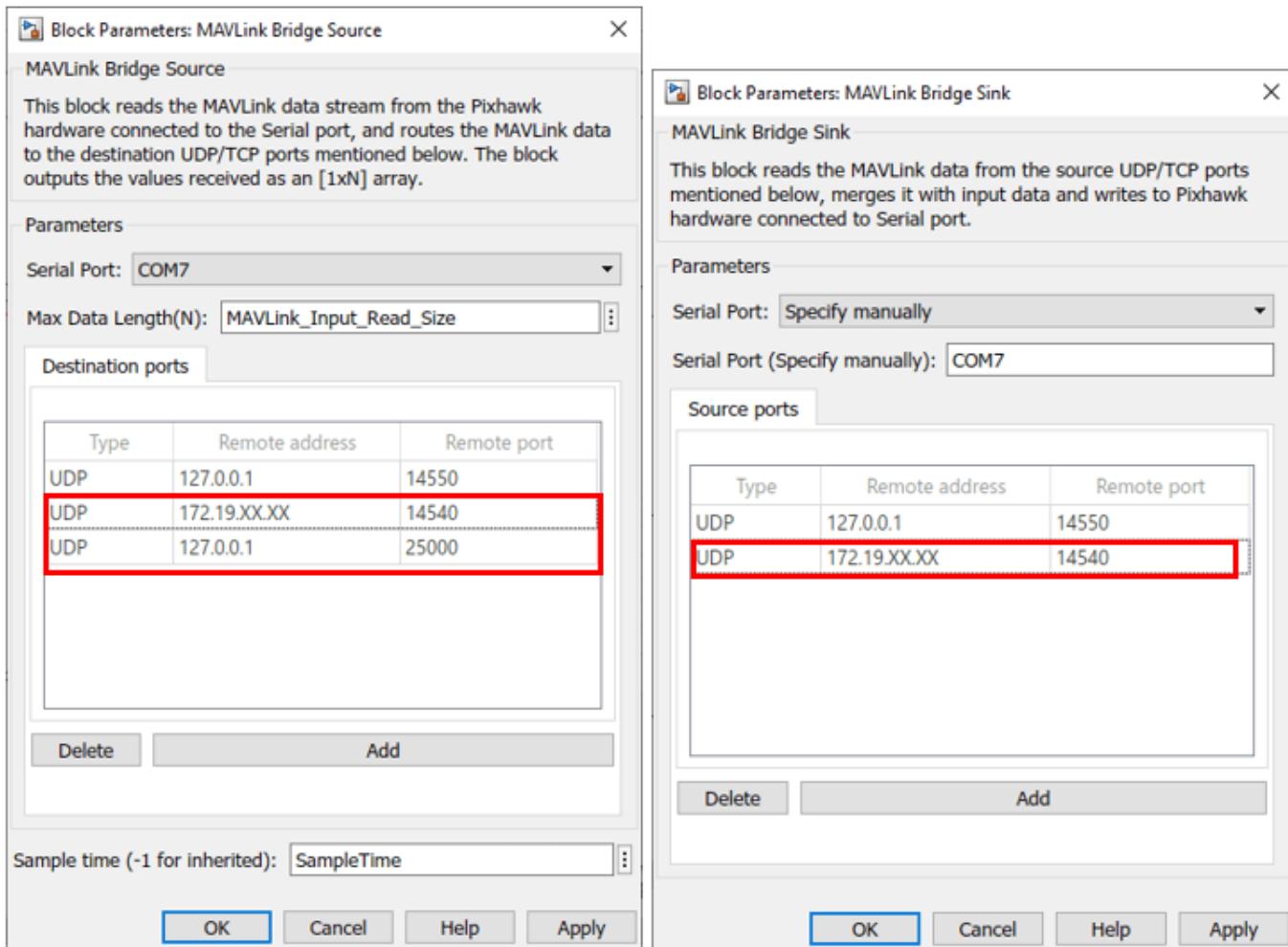


5. In the **Project Shortcuts** tab, from the PX4demo_HILSimulinkPlant drop-down list, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.

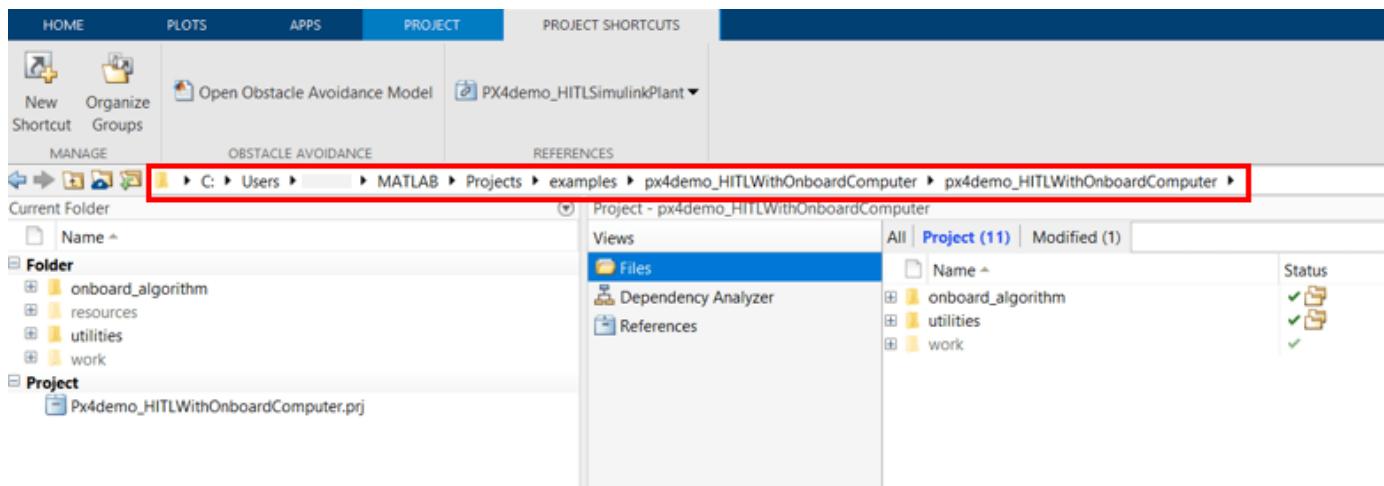


6. Open the Simulink Plant model *UAV_Dynamics_Autopilot_Communication* and configure the serial port. Select the serial port of Pixhawk which is connected to the host computer. Add the following UDP connections of onboard computer in the MAVLink Bridge blocks. For more information, see “MAVLink Connectivity for QGC, On-board Computer and Simulink Plant”. Double-click the MAVLink Bridge blocks to open the block Parameters dialog box.

- Add the IP address of onboard computer (NVIDIA Jetson) in the MAVLink Bridge Source and MAVLink Bridge Sink blocks. Ensure that you can ping the NVIDIA Jetson successfully from the host PC. Enter the port number as 14540.
- Add localhost connection for the flight visualization in the MAVLink Bridge Source block. Enter the port as 25000. Skip this if you are opting to not add the flight visualization.



7. Copy the MATLAB Project Path to clipboard.

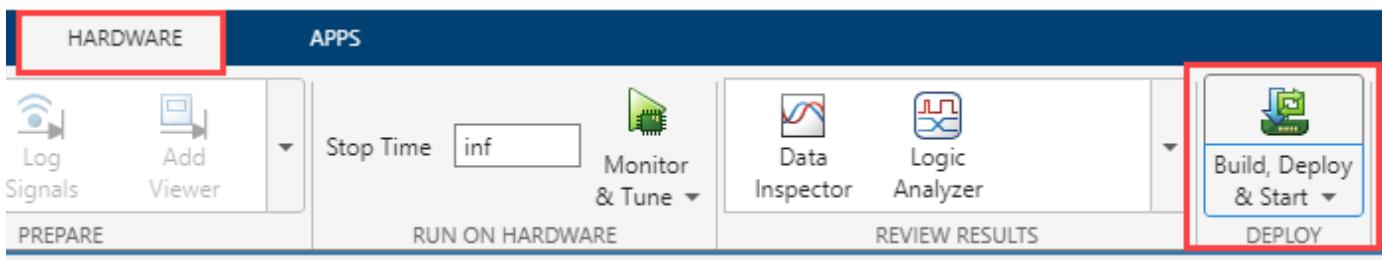


Step 3: Configure Simulink Controller Model for HITL Mode

- Follow the instructions mentioned in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

- Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*.



The code will be generated for the Controller model and the same will be automatically deployed to the Pixhawk board (Pixhawk 6x in this example).

After the deployment is complete, QGroundControl will be automatically launched.

Note: If you are using Ubuntu, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

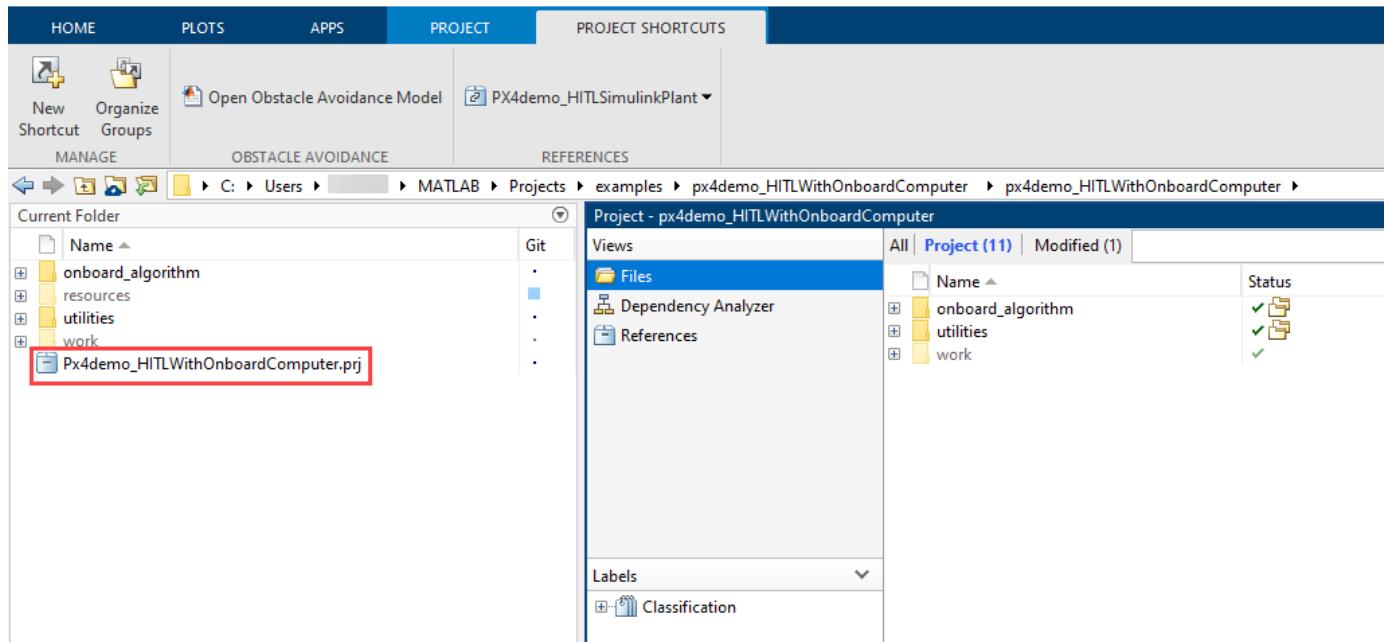
```
./QGroundControl.AppImage
```

Step 4: Launch Second Session of MATLAB and Open the Flight Visualization Model

- Open the second instance of the same MATLAB version. In this MATLAB session, the Simulink model for scenario simulation and flight visualization using unreal environment runs.
- Navigate to the project path previously copied in Step 2 in the current MATLAB.

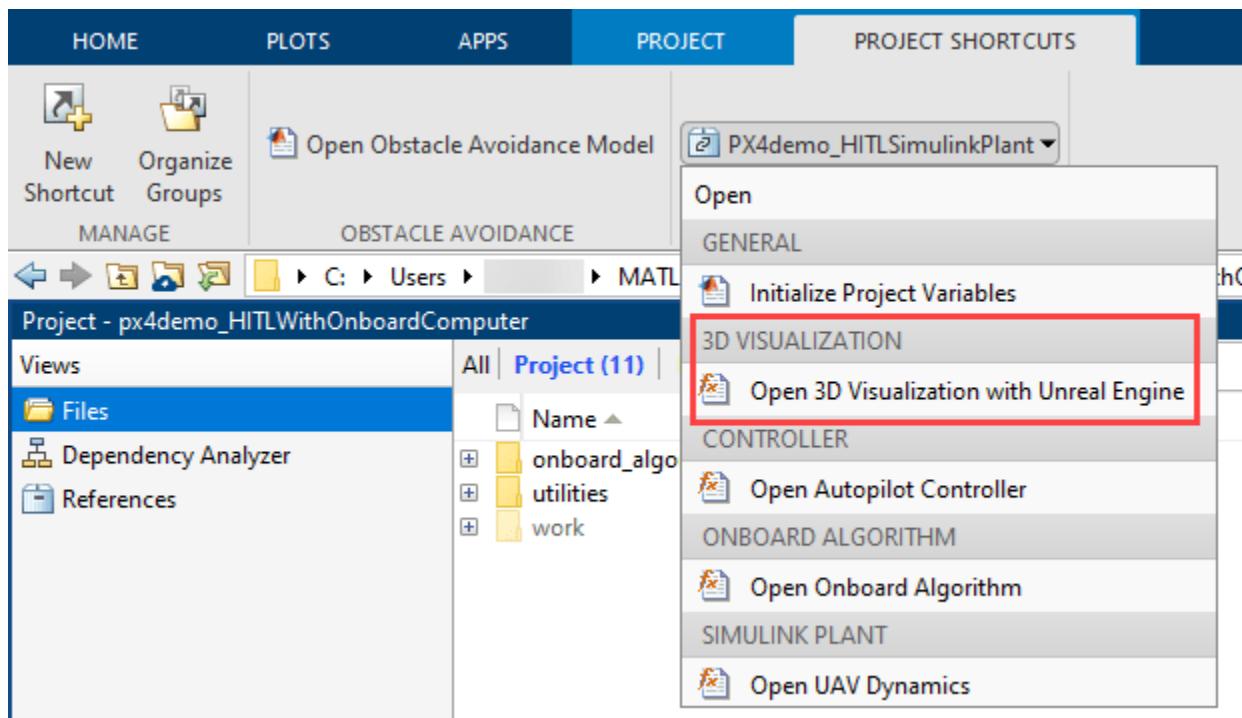
```
fx >> cd C:\Users\...:\MATLAB\Projects\examples\px4demo_HITLWithOnboardComputer\px4demo_HITLWithOnboardComputer
```

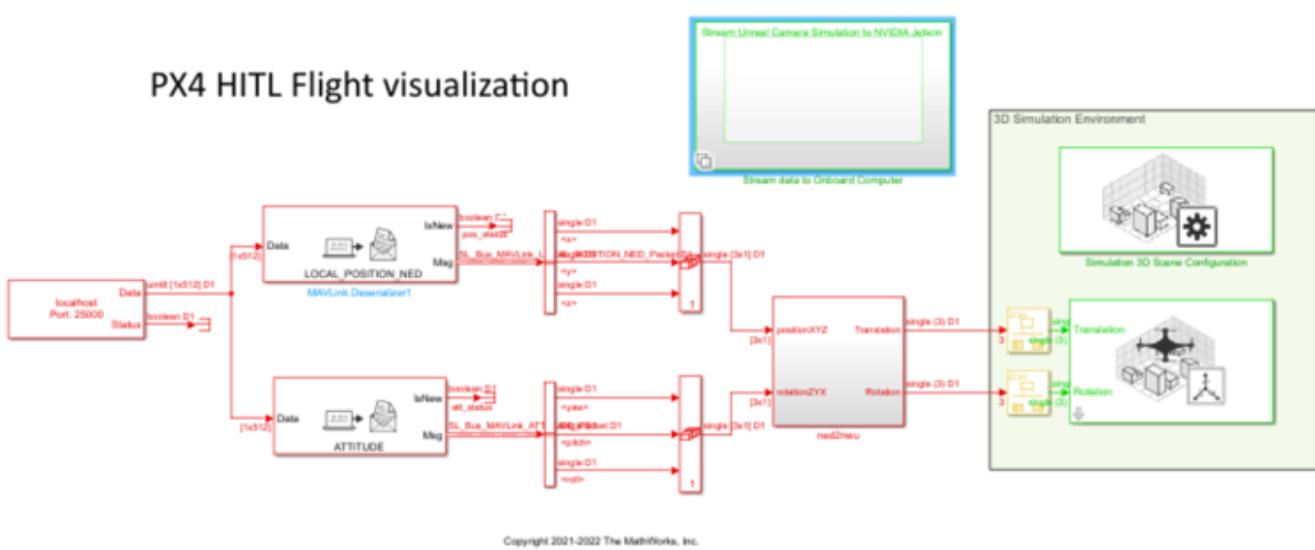
- Click on the .prj file to launch the same Project in current MATLAB.



Ensure that your Host PC is Configured with MATLAB supported GPU (GPU with compute capability of more than 5).

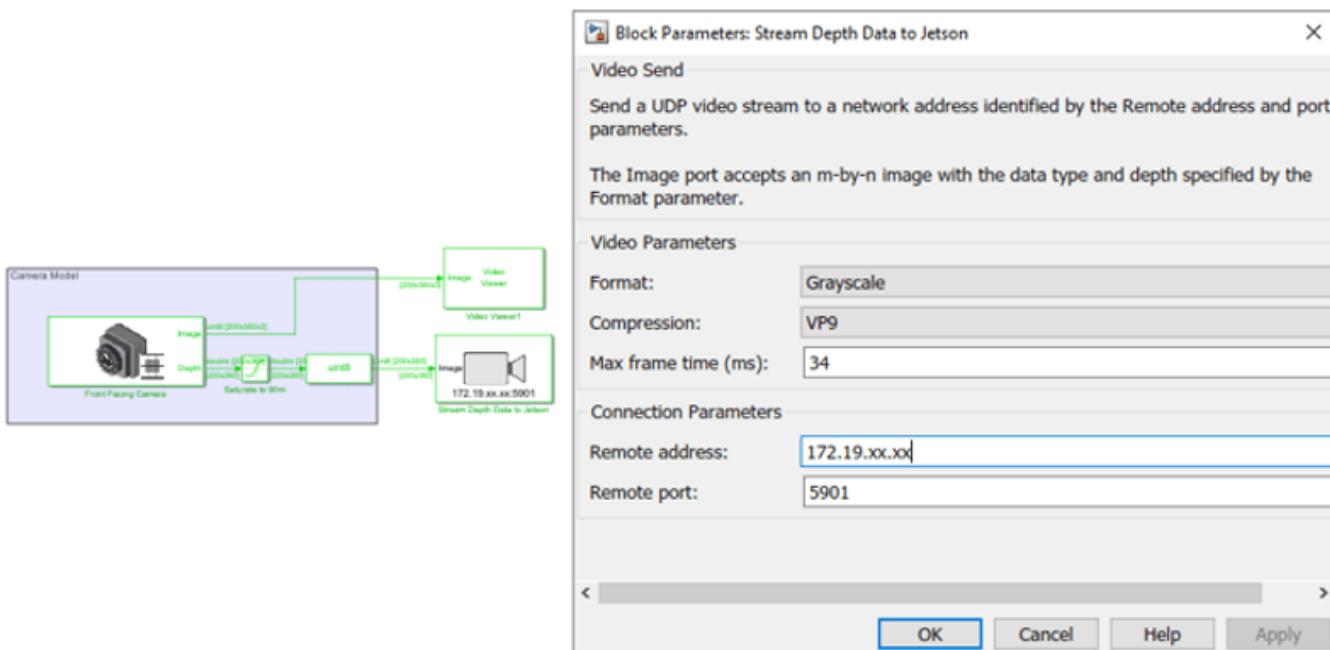
4. In the **Project Shortcuts** tab, under the *PX4demo_HITLSimulinkPlant* dropdown, click **Open 3D Visualization with Unreal Engine** to launch the onboard model named *Unreal_3DVisualization*.





In this model, The MAVLink data from the PX4 Autopilot is received over UDP (port : 25000) and is used to decode the position and attitude data of the UAV. After coordinate conversion, it is then passed to the Simulation 3D UAV Vehicle block for flight visualization.

5. Enable the streaming of simulated depth sensor data NVIDIA Jetson by updating the variable `enableOnboardStreaming` to 1 in MATLAB workspace.
6. The Simulation 3D Camera block provides the Camera image from the Unreal environment. In this example you stream the depth images from the camera block to NVIDIA Jetson using Video Send block. Double-click block to open the block Parameters dialog box. Add the IP address of onboard computer (NVIDIA Jetson) in the dialog box and click **OK**.



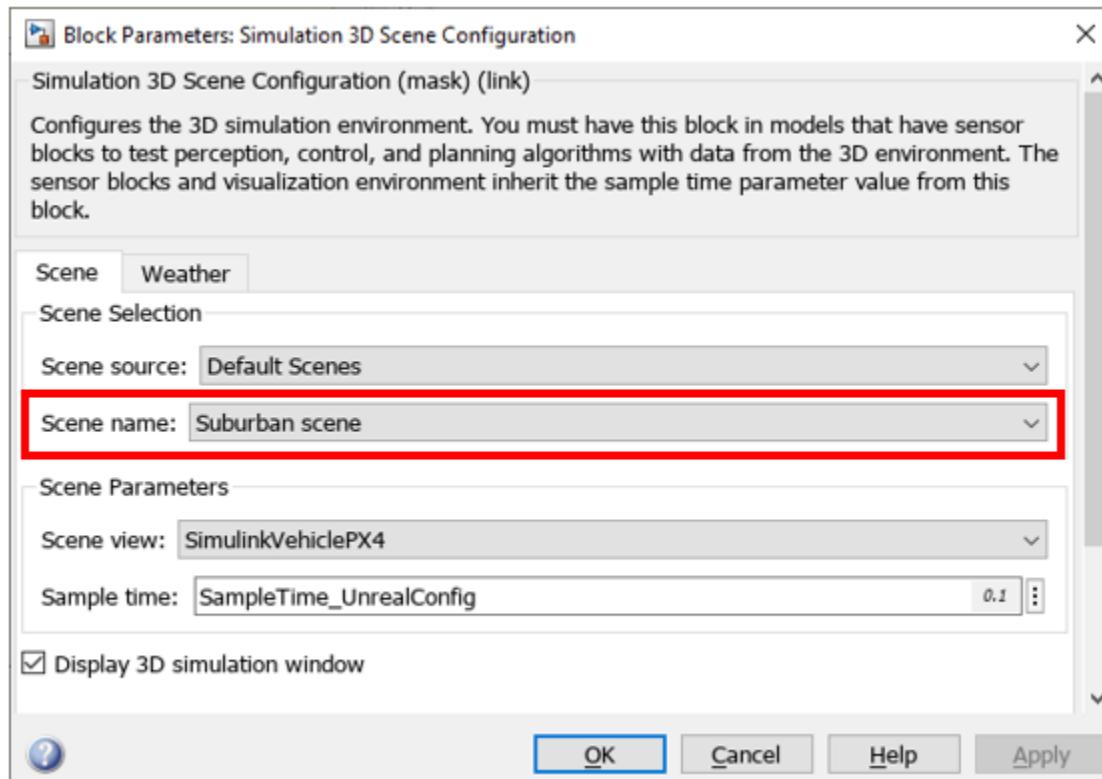
7. In this example, a Suburban scene is used for the Unreal simulation environment. To download this scene, run the following commands.

```
>> sim3d.maps.Map.download("Suburban scene")
```

Verify the map is downloaded on your disk by running the following command:

```
>> sim3d.maps.Map.local
```

8. Double click on the Simulation 3D scene Configuration block and select the Suburban scene from the dropdown for Scene name



9. On the **Simulation** tab, click **Run** to simulate the model. Once the model starts running, you will see the Unreal simulation environment launching. A sample screen is shown below.

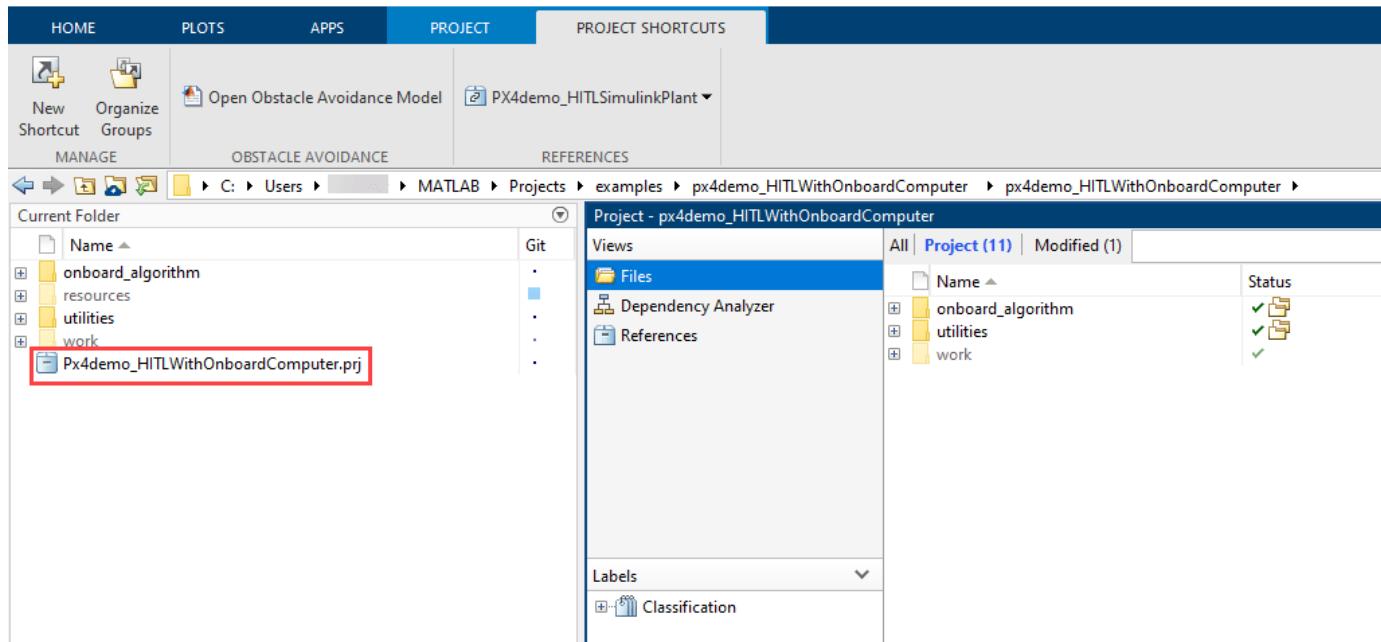


Step 5: Launch Third Session of MATLAB and the Onboard Model

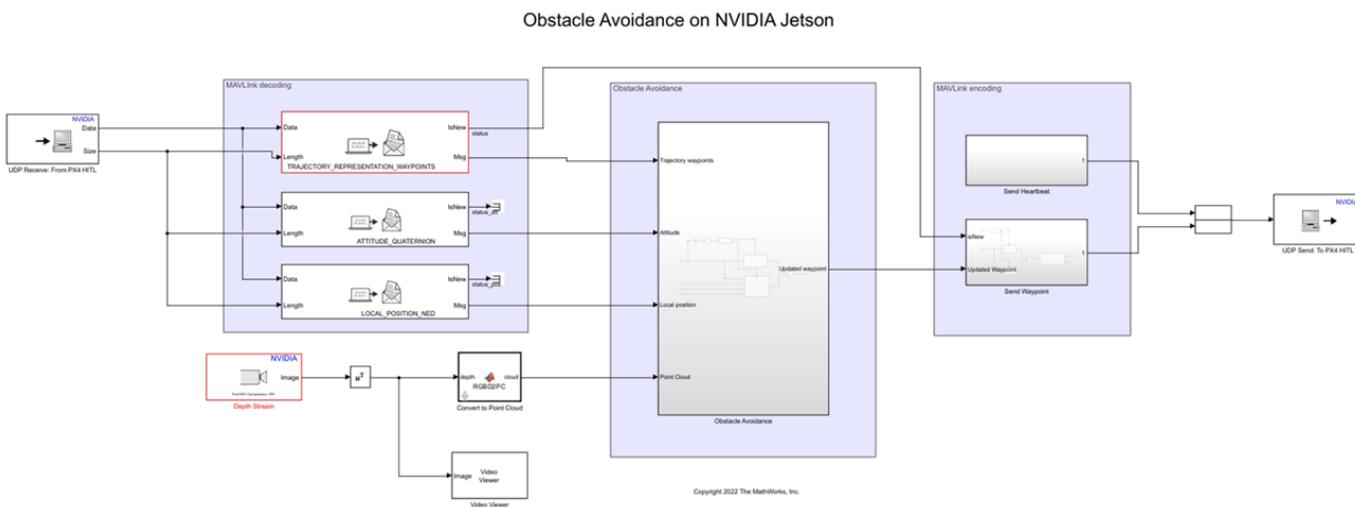
1. Open the third instance of the same MATLAB version.
2. Navigate to the project path previously copied in Step 2 in current MATLAB.

```
fx >> cd C:\Users\...\MATLAB\Projects\examples\px4demo_HITLWithOnboardComputer\px4demo_HITLWithOnboardComputer
```

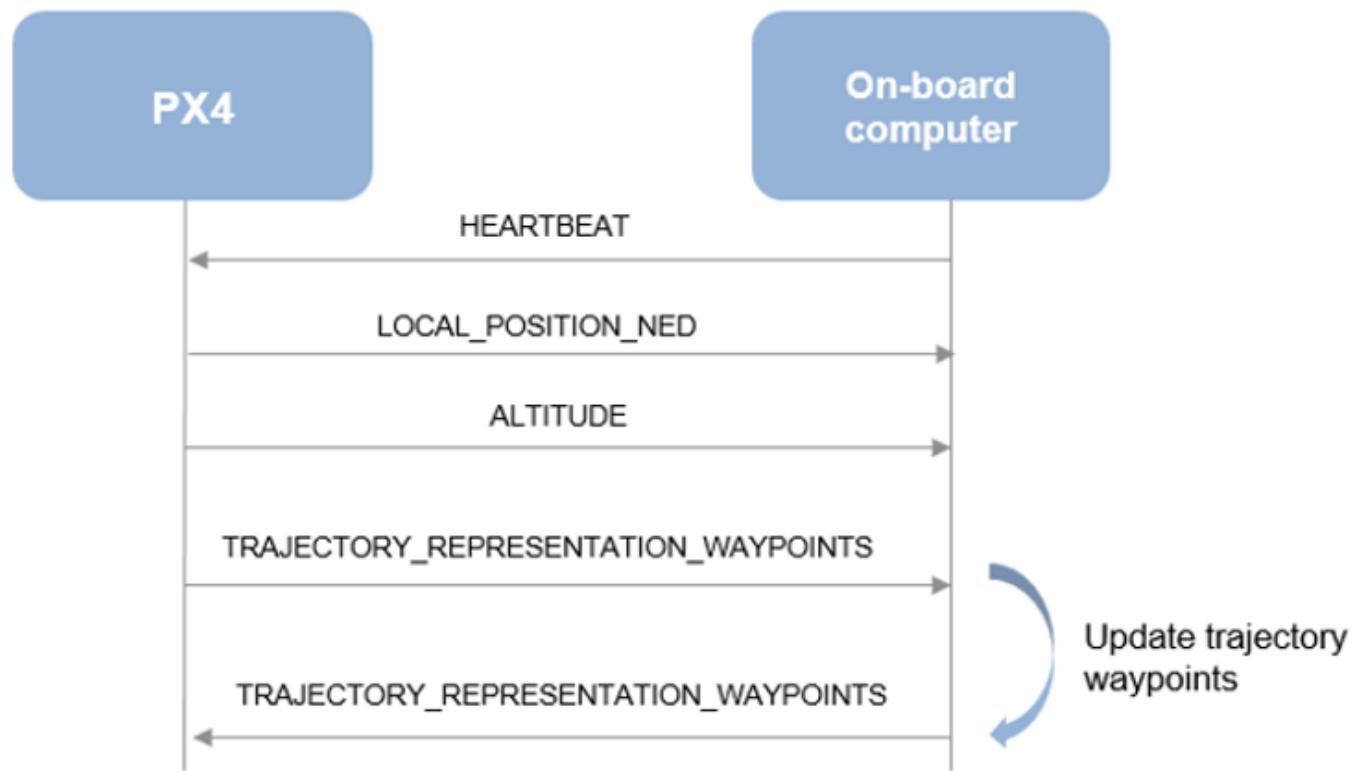
3. Click on the .prj file to launch the same Project in current MATLAB.



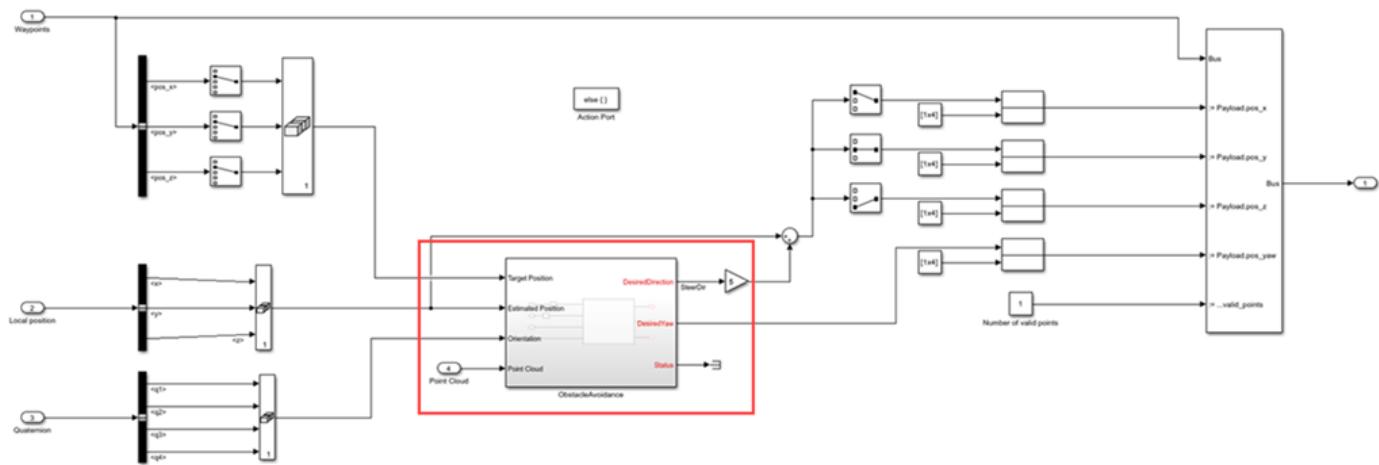
4. In the **Project Shortcuts** tab, click **Open Obstacle Avoidance Model** to launch the onboard model named *Onboard_ObstacleAvoidance*.

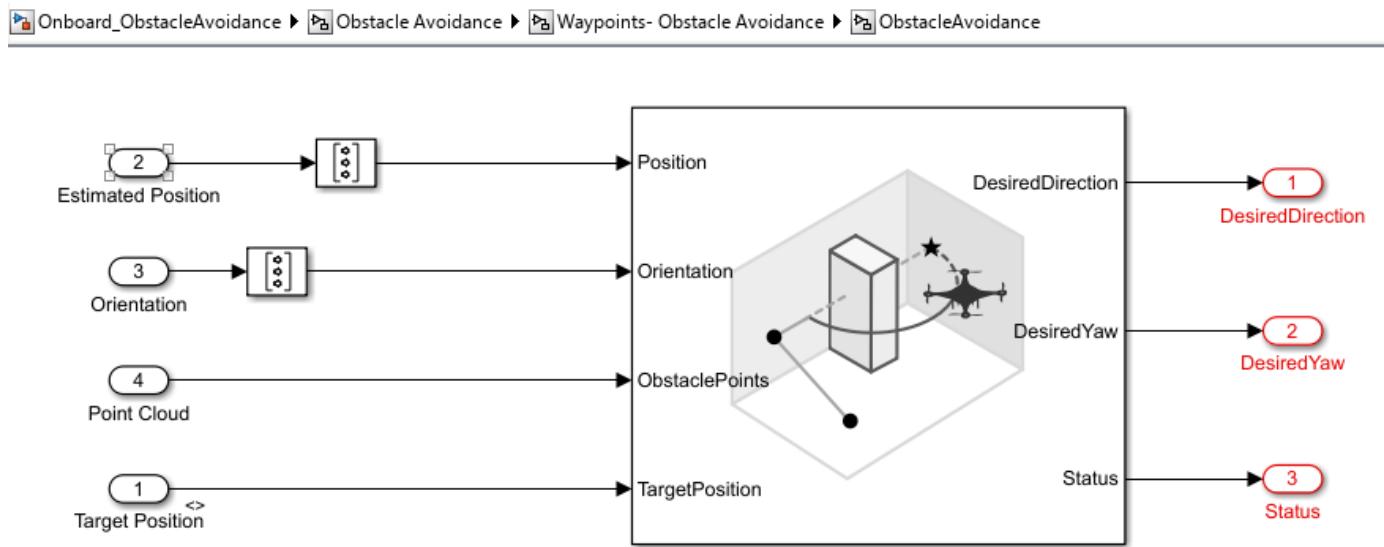


This model implements the PX4 path planning interface using MAVLink Serializer and MAVLink Deserializer blocks. For more information, see PX4 Path Planning Interface. The MAVLink messages that are exchanged as part of this interface is shown in the following diagram.

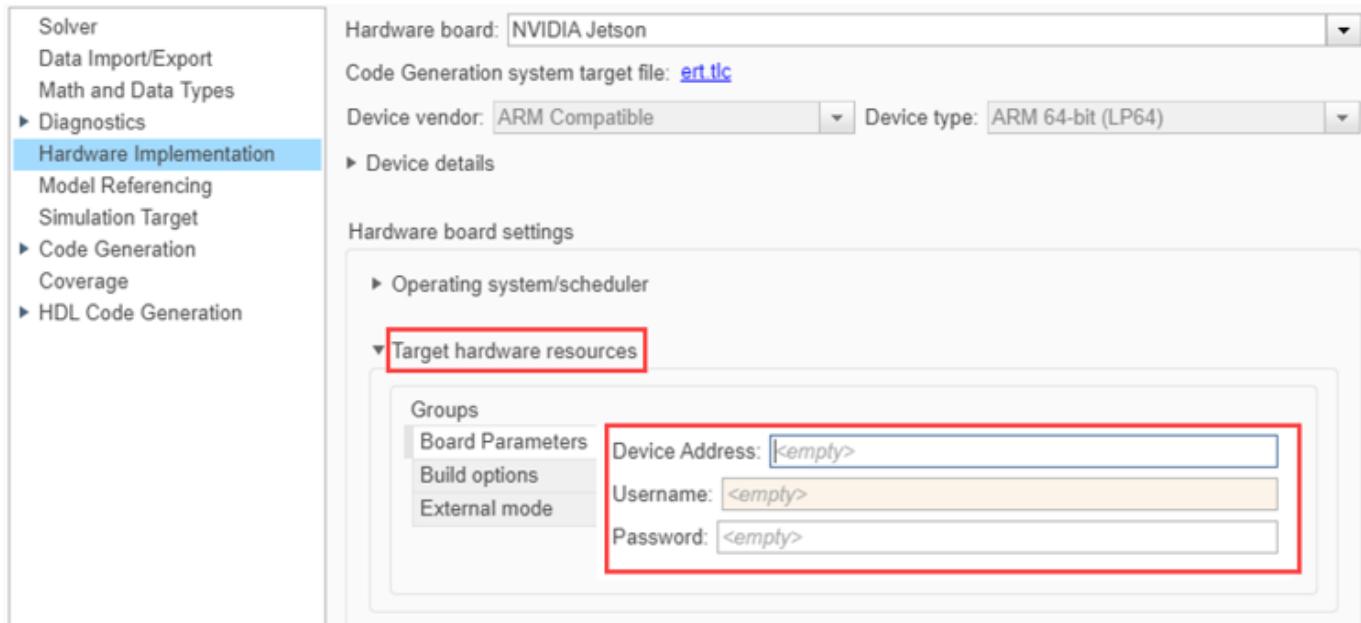


5. This example demonstrates how to avoid obstacles by using the Obstacle Avoidance block from the UAV Toolbox.

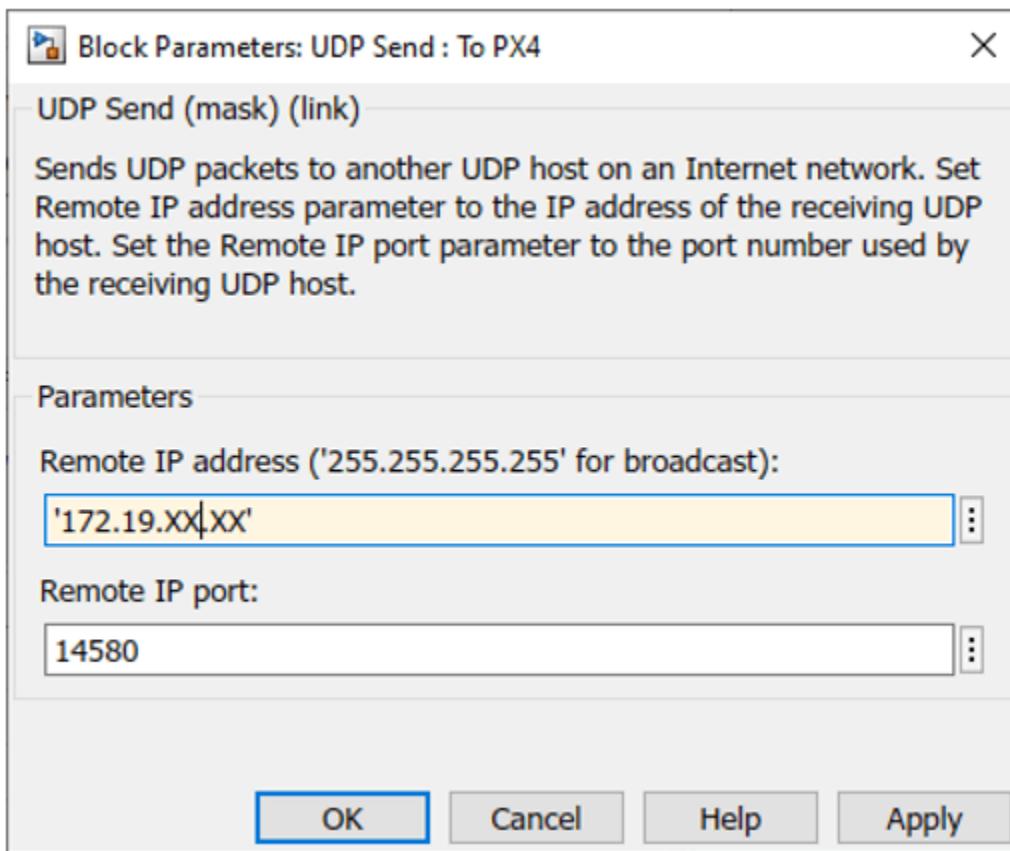




6. Navigate to Target hardware resource > Board Parameters, enter the IP address of the NVIDIA Jetson and your login credentials.

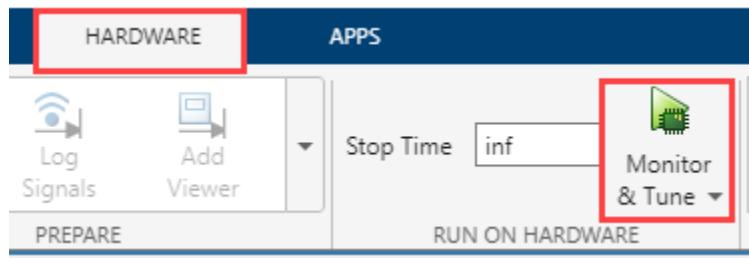


7. Double-click the UDP Send block to open the block Parameters dialog box. Enter the IP address of the host PC on which you are running UAV_Dynamics_Autopilot_Communication as Remote IP address. Enter the value for Port as 14580. Ensure that the host computer and the onboard computer are connected to same network.



8. Configure the Network Video Receive block to receive the depth data from Unreal environment. Note that the port number and compression parameters in the Network Video Receive block are same as those of the corresponding Video Send block streaming camera image from Unreal engine.

9. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolbar.



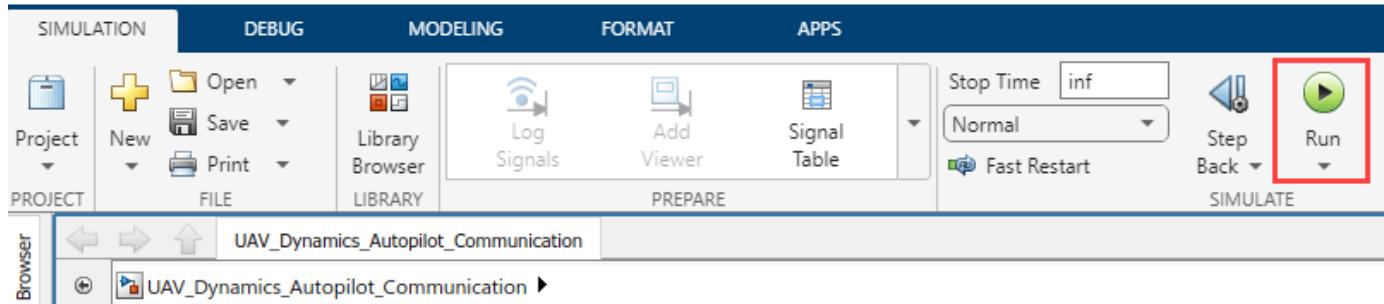
The code will be generated for the Controller model and the same will be automatically deployed to NVIDIA Jetson. NVIDIA Jetson should start communicating with host over Monitor & Tune Simulation.

Note: Ensure that there are no other deployed models from Simulink are running in NVIDIA Jetson. If you are unable to verify, reboot the Pixhawk hardware board before starting the deployment.

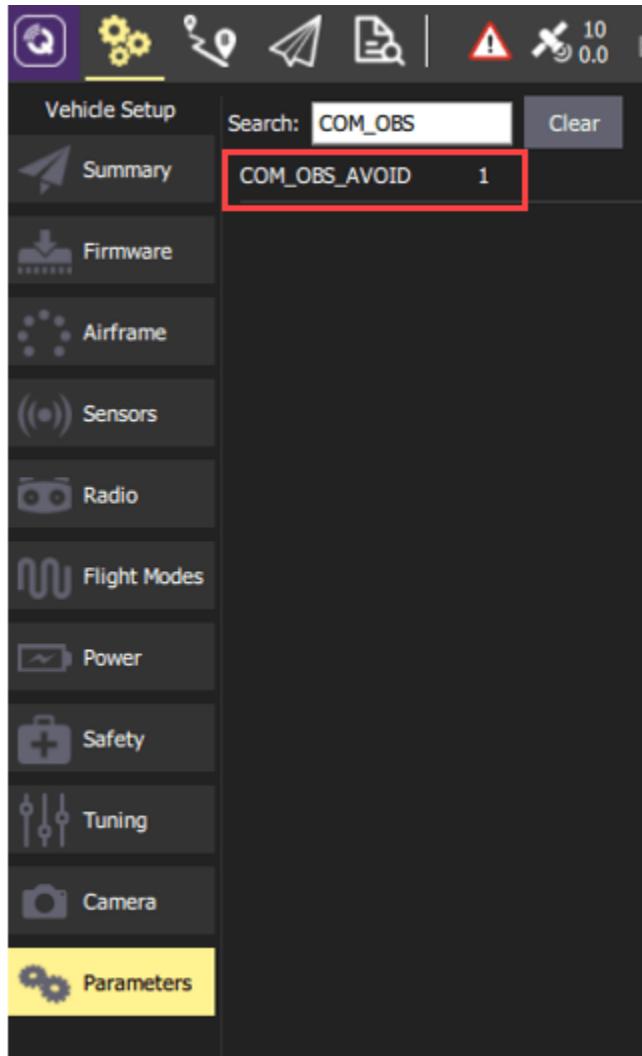
The algorithm in NVIDIA Jetson also communicates with the Plant model *UAV_Dynamics_Autopilot_Communication* over UDP.

Step 6: Run the UAV Dynamics model, Upload Mission from QGroundControl and Fly the UAV

1. In the Simulink toolstrip of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model. Once the plant model starts running, in QGroundControl connection will be established.

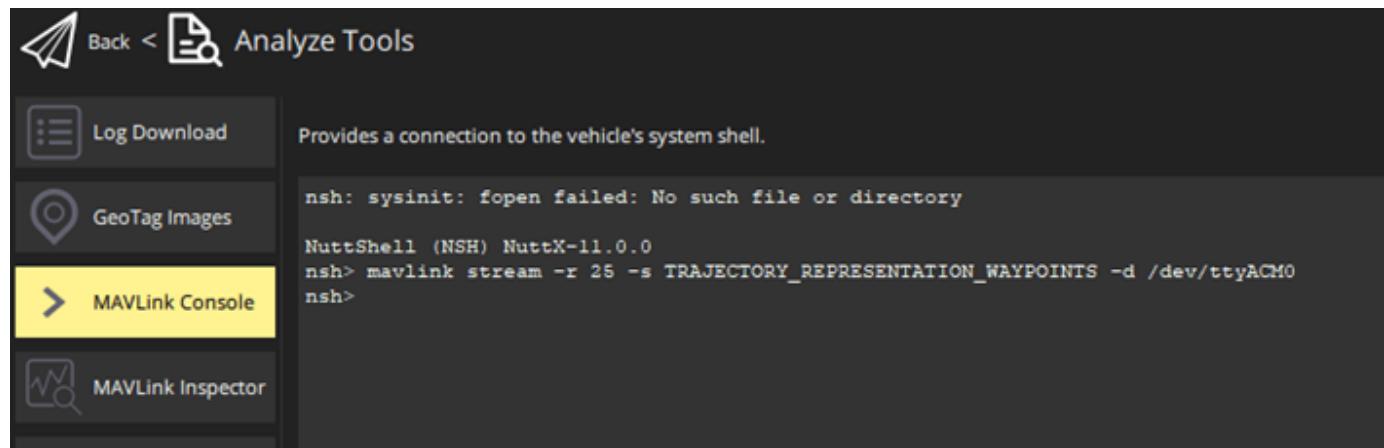


2. Set the PX4 parameter **COM_OBS_AVOID** enabling the PX4 path planning interface. Navigate to Parameters from the main menu and set the **COM_OBS_AVOID** parameter value to 1.

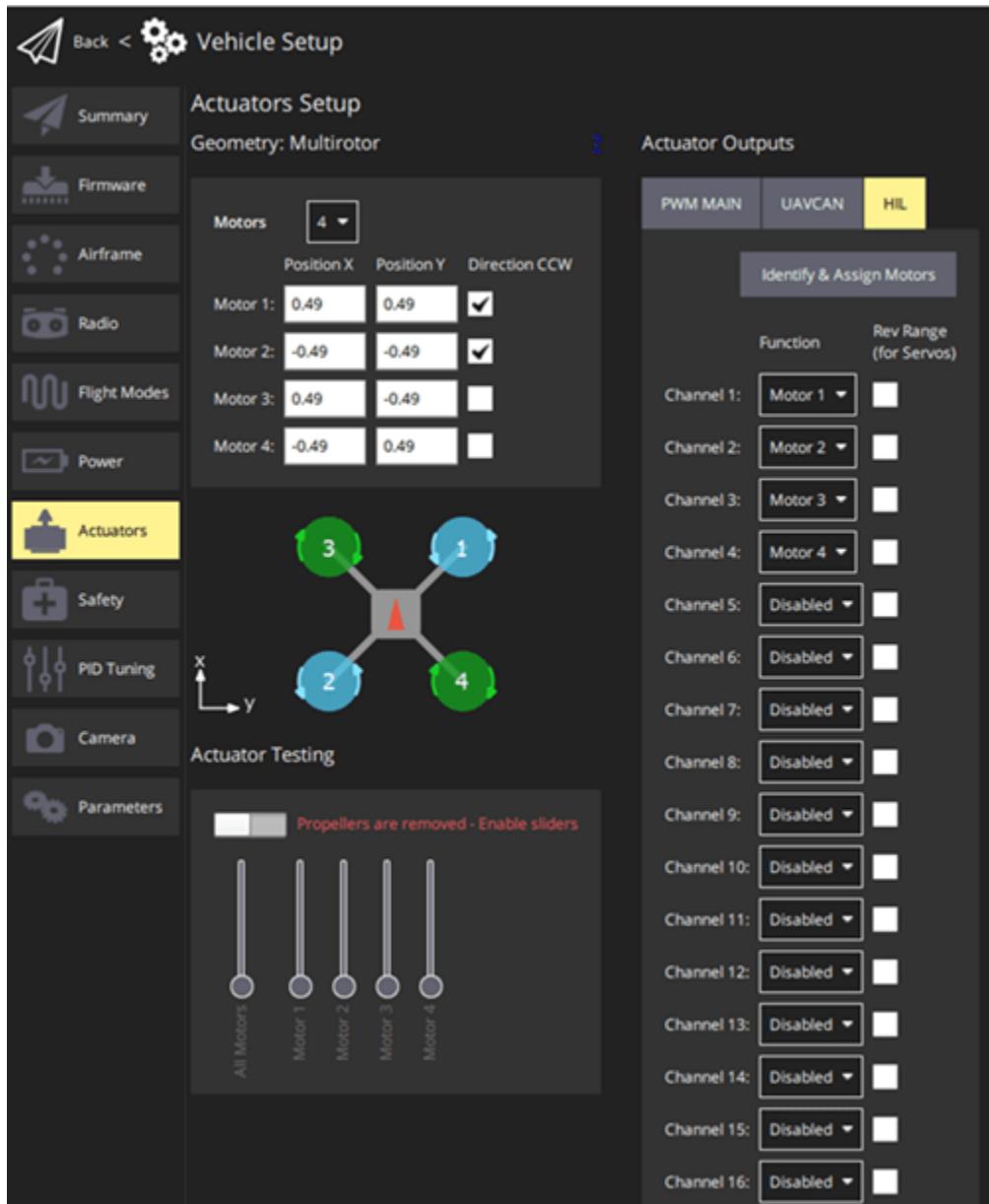


3. Set the parameter COM_DISARM_PRFLT value to -1.
4. Start MAVlink Stream for TRAJECTORY_REPRESENTATION_WAYPOINTS. This MAVLink message stream is required for onboard connectivity and is not started by default on USB (/dev/ttyACM0). To start the stream, run the following Command in the QGC MAVLink shell.

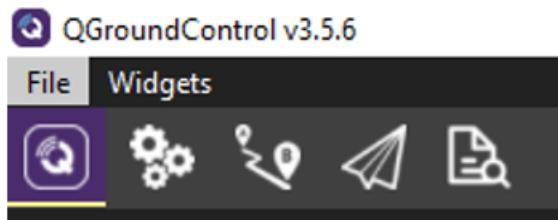
```
mavlink stream -r 25 -s TRAJECTORY_REPRESENTATION_WAYPOINTS -d /dev/ttyACM0
```



5. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



6. In the QGC, navigate to the *Plan View*.



7. This example provides a preplanned mission *OA_simple_mission.plan* that you can upload to QGC. To upload the preplanned mission, click **Open Model** button at the top of this page (MATLAB Help)

browser) to download the plan file (*OA_simple_mission.plan*). In the QGC, navigate and select *OA_simple_mission.plan* file.

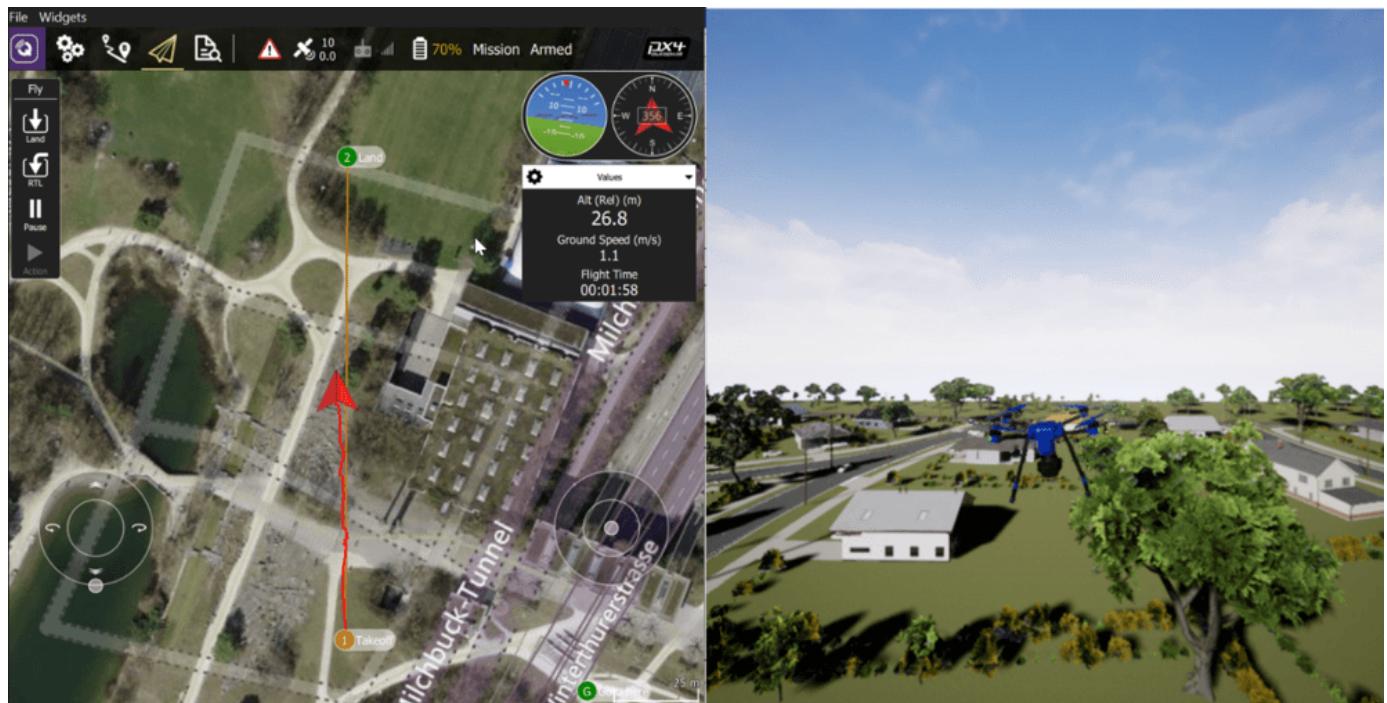
After you upload the plan, the mission is visible in QGC.

8. Click Upload button in the QGC interface to upload the mission from QGroundControl.

9. Navigate to *Fly View* to view the uploaded mission.

10. Start the Mission in QGC. The UAV should follow the mission path.

11. Observe the flight visualization in the Unreal Engine. The drone deviates from the desired path to avoid the tree in the Unreal scene.



12. Go to the *Onboard_ObstacleAvoidance* model and validate the depth image streamed to Jetson during the flight in the Video Viewer (from Computer Vision Toolbox™).



Troubleshooting

- While Simulating the visualization model in Step 4, you might get STD exception errors such as, "some module could not be found".

Change the compiler to Microsoft Visual C++ 2019 using `mex -setup c++` command to fix the issue.

- "Avoidance system not available" warning when starting a mission. This warning occurs because the communication between NVIDIA Jetson and PX4 HITL is not established.

Ensure that host PC and NVIDIA Jetson are connected to same network. Try pinging NVIDIA Jetson from the host PC and vice versa. Also, double-check the NVIDIA Jetson IP address entered in the MAVLink Bridge blocks as mentioned in 6th point of Step 2 and Host PC IP address entered in the UDP send block as mentioned in 7th point of Step 5.

- When you start the Mission the vehicle is not taking off.

Firstly, verify if you are receiving the 'TRAJECTORY_REPRESENTATION WAYPOINTS' message in the Onboard model. If the 'isNew' output for 'TRAJECTORY_REPRESENTATION WAYPOINTS' in the MAVLink Deserializer Block related to is always false, it indicates that the message is not being received.

In this case, start MAVlink Stream for TRAJECTORY_REPRESENTATION WAYPOINTS. This MAVLink message stream is required for onboard connectivity and is not started by default on USB (/dev/ttyACM0). To start the stream, run the following Command in the QGC MAVLink shell.

```
mavlink stream -r 25 -s TRAJECTORY REPRESENTATION WAYPOINTS -d /dev/ttyACM0
```

Reading GPS Data over UAVCAN

This example showcases the connectivity with the UAVCAN peripherals connected to the PX4® Autopilot, using UAV Toolbox Support Package for PX4 Autopilots. In this example, the UAVCAN GPS receiver is used as the UAVCAN peripheral.

UAVCAN driver module of PX4 establishes connectivity with the peripheral and the GPS data is published over uORB message. This example uses a pre-defined Simulink model (*px4demo_readGPS*) that contains the GPS block. The GPS block gets the GPS data by reading the `sensor_gps` uORB message.

Prerequisites

- If you are new to Simulink®, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

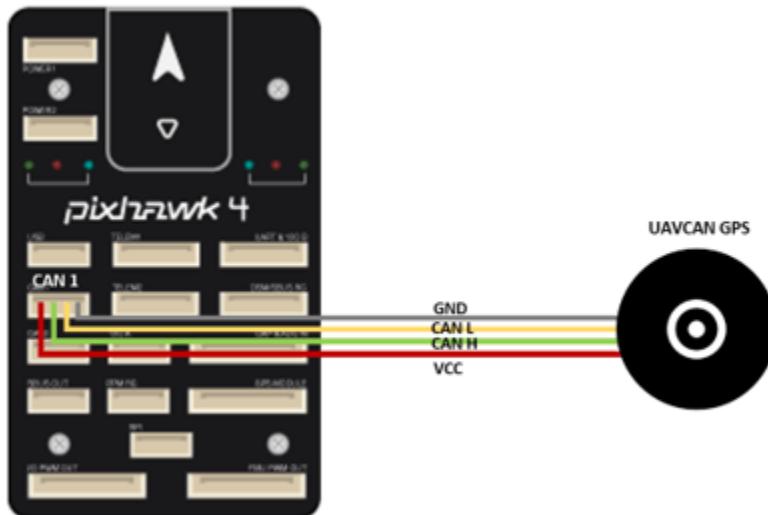
Required Hardware

To run this example, you will need the following hardware:

- Supported PX4 Autopilot
- Micro USB type-B cable
- UAVCAN GPS device

Hardware Connection

Connect the UAVCAN GPS module to the CAN1/CAN port of the PX4 Autopilot and power up the PX4 Autopilot.



Task 1 - Enable UAVCAN Module Using QGroundControl

In this task, you will enable the UAVCAN module using the parameter `UAVCAN_ENABLE`. You can set different values for `UAVCAN_ENABLE` parameter and their behavior is listed below.

0 - UAVCAN disabled.

1 - Enables support for UAVCAN sensors without dynamic node ID allocation and firmware update.

2 - Enables support for UAVCAN sensors with dynamic node ID allocation and firmware update.

3 - Enables support for UAVCAN sensors and actuators with dynamic node ID allocation and firmware update. Also sets the motor control outputs to UAVCAN.

As this example uses UAVCAN sensor, set the `UAVCAN_ENABLE` parameter value to 2. If you are interfacing an UAVCAN actuator such as ESC, consider setting the `UAVCAN_ENABLE` parameter value to 3.

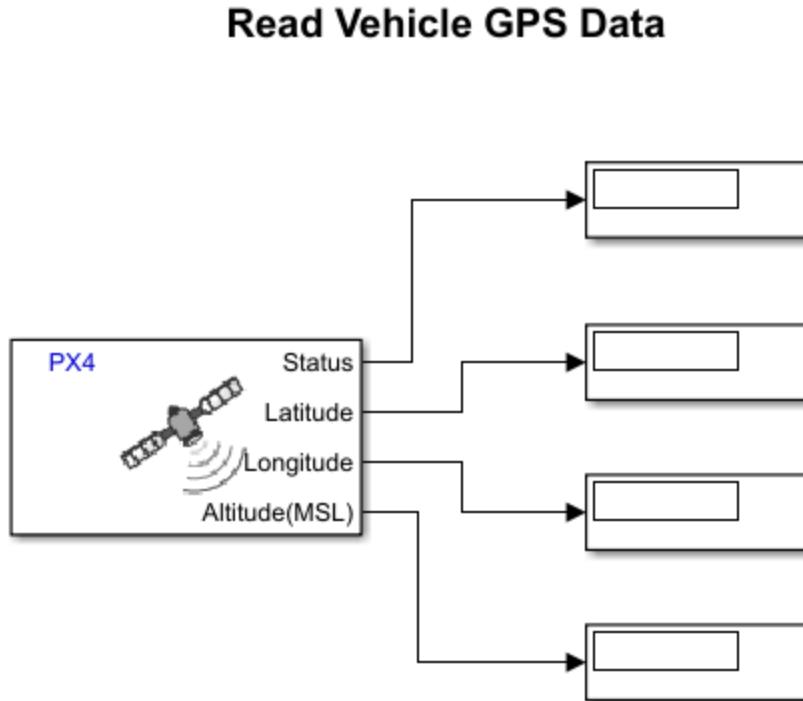
Set the `UAVCAN_ENABLE` parameter value to 2 using QGroundControl as described here.

Note: PX4 CAN Transmit and PX4 CAN Receive blocks cannot be used when UAVCAN is enabled. Disable the UAVCAN before using the PX4 CAN Transmit and Receive blocks.

Task 2 - Signal Monitoring and Parameter Tuning

In this task, you will configure and run the pre-defined Simulink model (`px4demo_readGPS`) model in Monitor and Tune action to verify the GPS data. This model contains the GPS block that reads `sensor_gps` uORB topic. After you verify the GPS data, you deploy the model to the PX4 hardware board.

1. Open the `px4demo_readGPS` model.



In the example model, the GPS block is used to read GPS values. This block accepts signals from the `sensor_gps` uORB message, and outputs the required values.

2. Double-click the GPS block. By default, signals corresponding to **Latitude**, **Longitude**, and **Altitude(MSL)** are selected as outputs. You can add the other signals as output by selecting the corresponding checkboxes.

Note: The **Status** output indicates if the uORB message was received during a previous time step or not. A value of 0 indicates that the uORB data at the output is the latest, and a value of 1 indicates that the uORB data was received during the previous time step. This output can be used to trigger subsystems for processing new messages received in the uORB network.

3. In the **Modeling** tab, click **Model Settings**.

4. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane:

- Set the **Hardware board** to the same PX4 hardware board that you selected during Hardware Setup screens.
- In the **Target Hardware Resources** section, set the **Build options** to **Build, load and run** to automatically download the generated binary file on to the connected Pixhawk® Series flight controller.
- Enter the serial port of the host computer to which the Pixhawk Series flight controller is connected, in the *Serial port for firmware upload* field.
- In the **External mode** pane, select the option **Use the same host serial port for External mode as used for firmware upload**.

5. Navigate to **Solver** pane and select the option **Treat each discrete rate as a separate task**. Click **OK**.

6. In the **Simulation** tab, set the *Stop time* to **inf**.

7. Connect the USB cable from the PX4 flight controller to the host computer.

8. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

9. Verify the output GPS values in the Scope display. The values correspond to the current GPS position of the PX4 flight controller.

10. Stop the Monitor and Tune operation by clicking **Stop** in the **Hardware** tab.

Deploy Simulink Model

To deploy the Simulink model, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. The model is deployed to the PX4 hardware board.

Log Simulink Signals Using PX4 ULog

This example shows how to log signals in ULog format from a Simulink® model running on a Pixhawk® hardware and retrieve the log files from the SD card for further analysis.

This example uses a PX4® Pixhawk 6x as the hardware board. However, you can run this example using any PX4 Pixhawk hardware board.

Introduction

The UAV Toolbox Support Package for PX4 Autopilots supports logging of signals from your Simulink model on Pixhawk hardware in the ULog format. Signal logging enables you to monitor the behavior of the signal and perform analysis of historical data.

Prerequisites

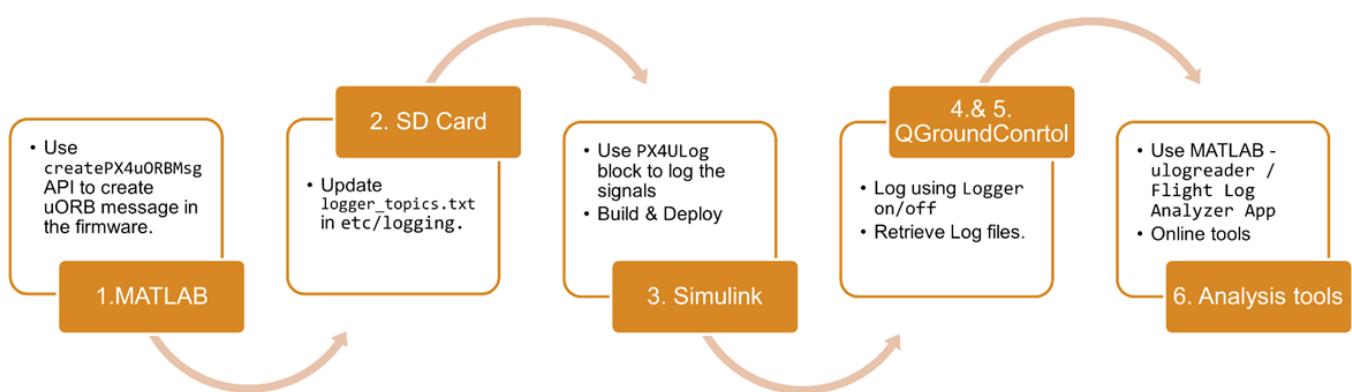
- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

To run this example, you will need the following hardware:

- Pixhawk Series flight controller
- Micro USB type-B cable
- Micro-SD card (already used during the “Performing PX4 System Startup from SD Card”)

To log signals of your choice, follow the instructions provided below. The following image helps you in understanding the complete workflow.



Task 1: Getting Started with Logging Prebuild Custom uORB Message

In this task, you will utilize the pre-configured Simulink model to log few signals into the custom uORB message that builds during the Hardware Setup screens.

Step 1 - Include Custom uORB Message in Firmware

The first step requires you to create a new custom uORB message and include this in the firmware. To create a new message, use the `createPX4uORBMessage` MATLAB API. A custom message `SimulinkCustomMessage` is generated when performing the setup screens. This message is used in the current example to log signals.

This topic is configured to log 4 inputs with double and single data types. To log the desired signals, input the necessary information with appropriate data types for logging.

Step 2 - Update SD Card Configuration

You can customize the list of uORB topics to log by using a text file on the SD card. This needs to be updated with the new `SL_CustomPX4uORBVerbose` topic. Each time a new uORB message is created, this file includes the new topic to log. This step adheres with the current PX4 Logging workflow.

For more information, see [PX4 Logging Workflow](#).

Note: There is a difference between the naming conventions used in MATLAB® and those used for PX4 logging. To use the generated uORB topic `SimulinkCustomMessage` you must change this before updating in the `logger_topics.txt`.

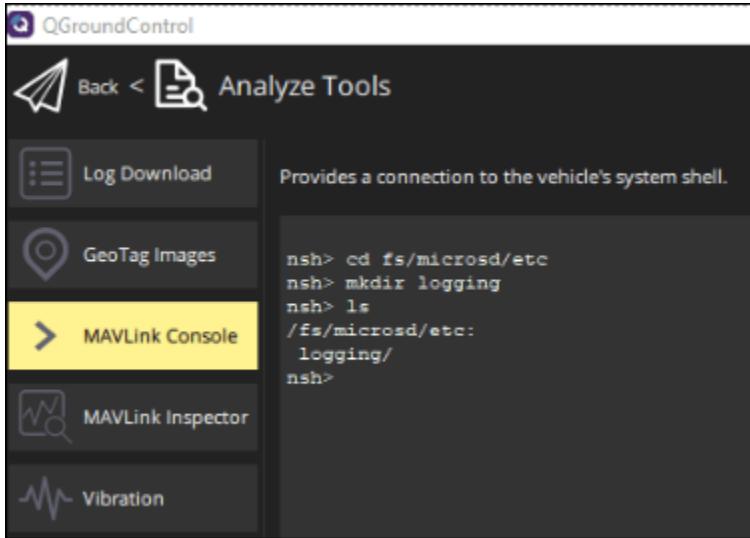
This step involves converting the names to lower case letters with a trailing underscore. Below is an example that illustrates this conversion process.

<u>MATLAB / Simulink</u>	<u>logger_topics.txt (SD Card)</u>
SimulinkCustomMessage	simulink_custom_message
NewSignalA	new_signal_a
DoubleTypeSignal	double_type_signal

You can update the SD card configuration in one of two ways.

Use Terminal with Connected SD Card (Recommended method)

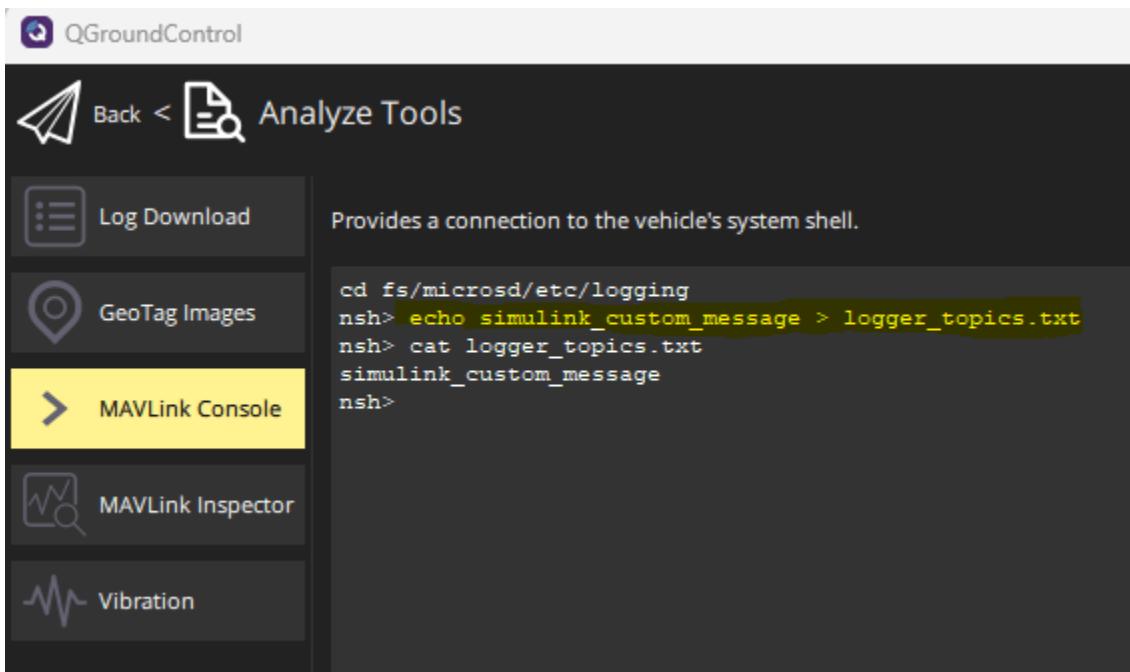
1. This method requires QGC to access the nsh console. In the QGC, navigate to **Analyze Tools > MAVLink Console**. If the console is not accessible, ensure to turn on the Enable MAVLink option as mentioned in Step 3, point 4.
2. Go to the `etc/` folder on the SD Card and create a new logging folder. Use the following commands to create the folder, if it is not already created.
 - a. Access the location using the `cd` command.
`nsh > cd fs/microsd/etc`
 - b. Create the logging folder using the `mkdir` command.
`nsh > mkdir logging`



3. Once the logging folder is created, include a `logger_topics.txt` file is in this location.

- Use the following command to include the text file `logger_topics` with the uORB topic `SimulinkCustomMessage` mentioned in the file.

```
nsh > echo simulink_custom_message >> logger_topics.txt
```



Manual Method - with SD Card disconnected

The above mentioned steps can be manually done by removing the SD card from Pixhawk board and accessing the contents.

Note: Along with the `topic_name` the instance and interval for the topic can be mentioned as additional information.

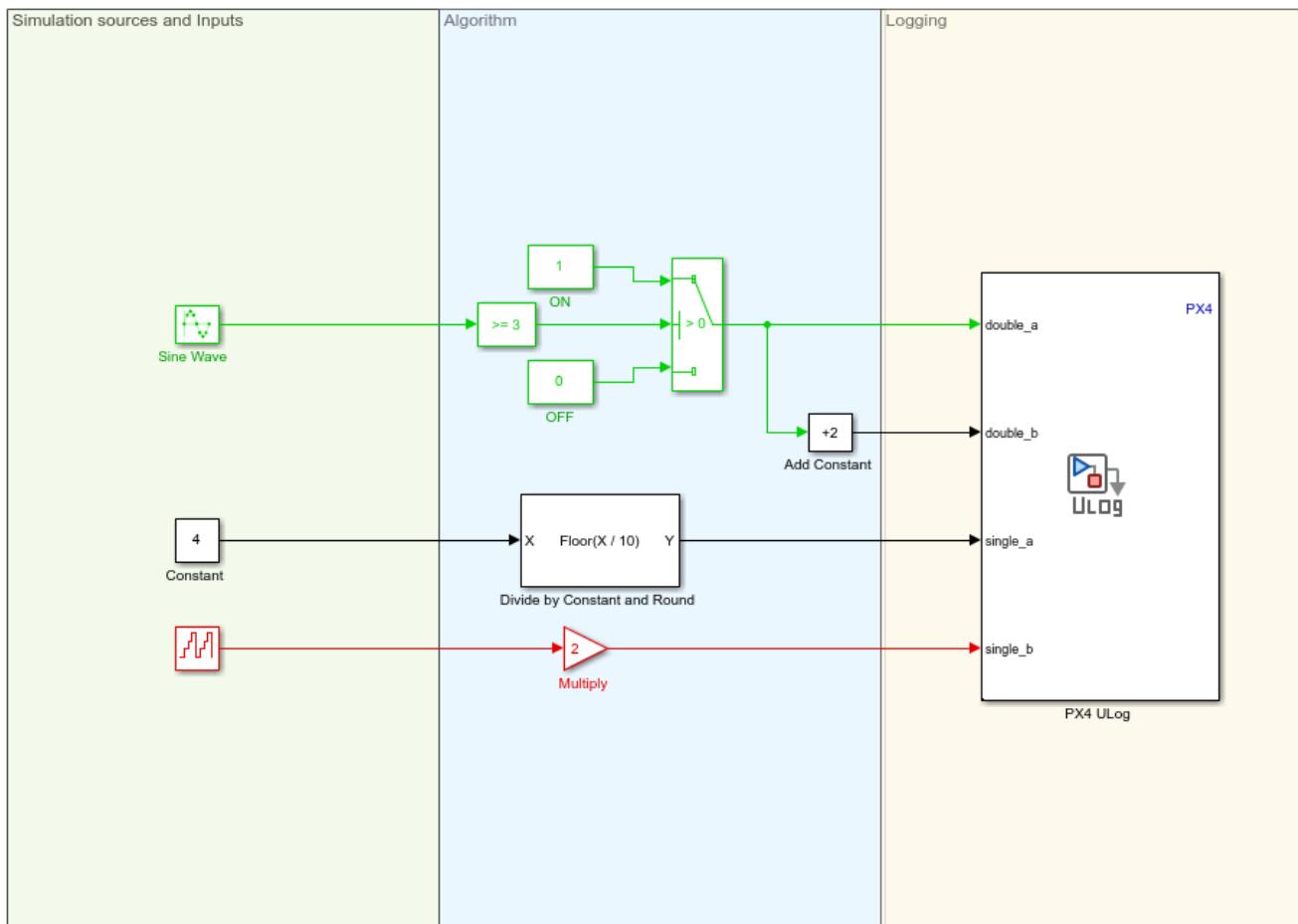
Example:

```
<topic_name><interval><instance>
simulink_custom_message 100 1
```

Step 3 - Open Model

1. Open the px4demo_uglog model.

Logging Simulink signals in ULog format



In the example model, the Simulink signals are connected to the PX4 ULog, which is responsible for logging the selected topic (`SimulinkCustomMessage`) on the block.

This block logs the input signals as a ULog file in the SD Card. The input signals are logged as a custom uORB topic as mentioned in the block parameters.

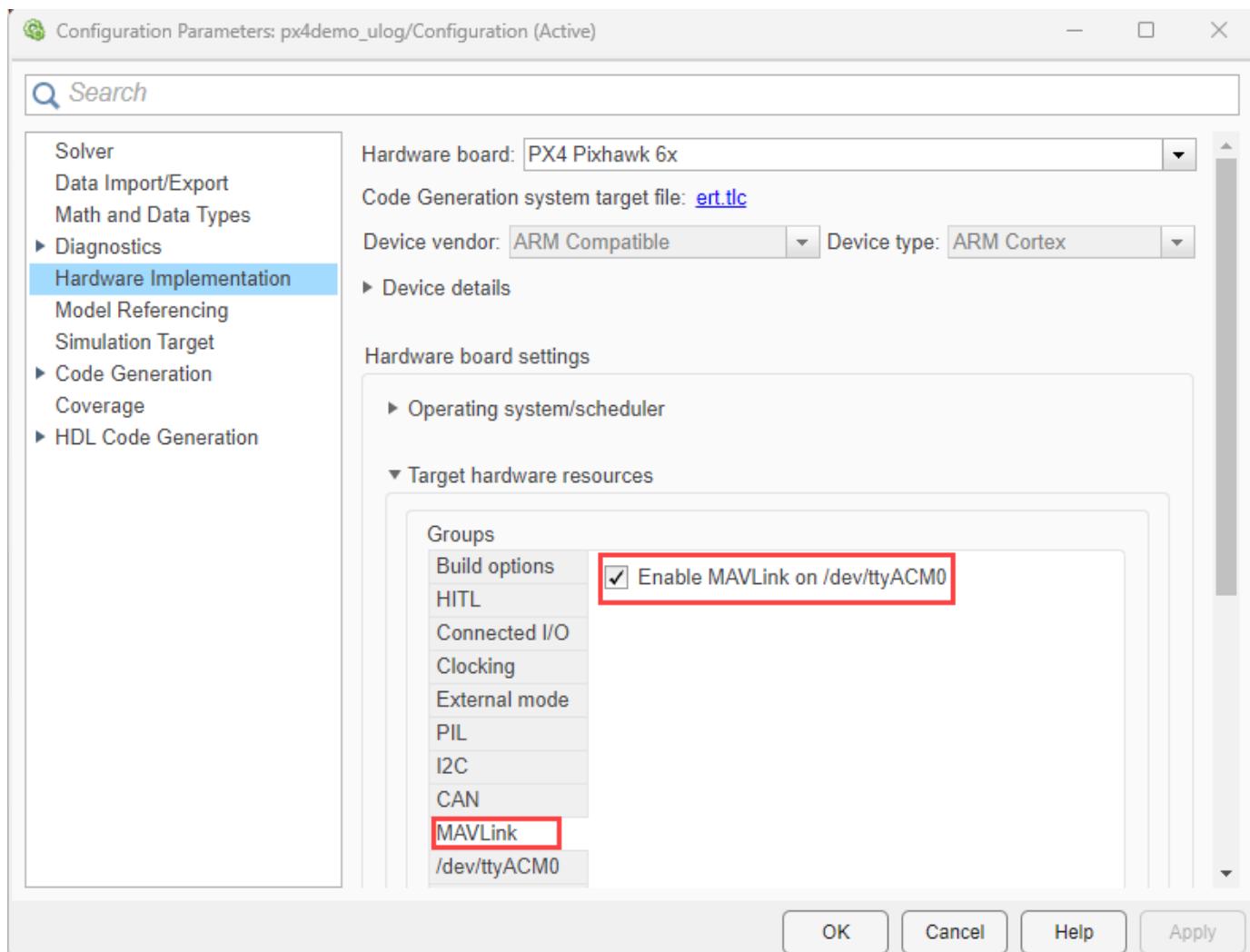
The name and number of imports are mentioned in the custom uORB topic.

This block internally maps the connected signals to the uORB Write block and is connected to a PX4 Timestamp block that writes the time from Hardware. Here the ports are updated as per message description for the selected topic **SimulinkCustomMessage**.

Note: In this example, sample time is set to 30ms, as there might be issues related to logging while using Pixracer board and sample time set to 10ms. However, there is no issue on other boards with sample time set to 10ms. For troubleshooting, see Dropouts and SD Cards.

The timestamp block **PX4 Absolute Time** in this model is required to write the timestamp from Pixhawk to Simulink. This timestamp information is useful to analyze the ULog data and can be used from the PX4 Utils library in the Simulink model. This Simulink model allows you to log the desired Simulink signals in a custom uORB topic, which can be analyzed later to gain insights. You can copy and paste this block to any other Simulink model using **Ctrl + C** and **Ctrl + V** keys on the keyboard.

2. Open the Model Configuration Parameters dialog box by selecting **Modeling > Model Settings** on the Simulink toolbar.
3. Go to the **Hardware Implementation** pane and select the name of the target hardware from the **Hardware board** list. If you are using a hardware board other than the Pixhawk 6x, change the **Hardware board** selection.
4. In the Configuration Parameters dialog box, go to **Hardware Implementation > Target hardware resources > MAVLink**, and select **Enable MAVLink on /dev/ttyACM0**.



5. Click **OK** to close the dialog box.

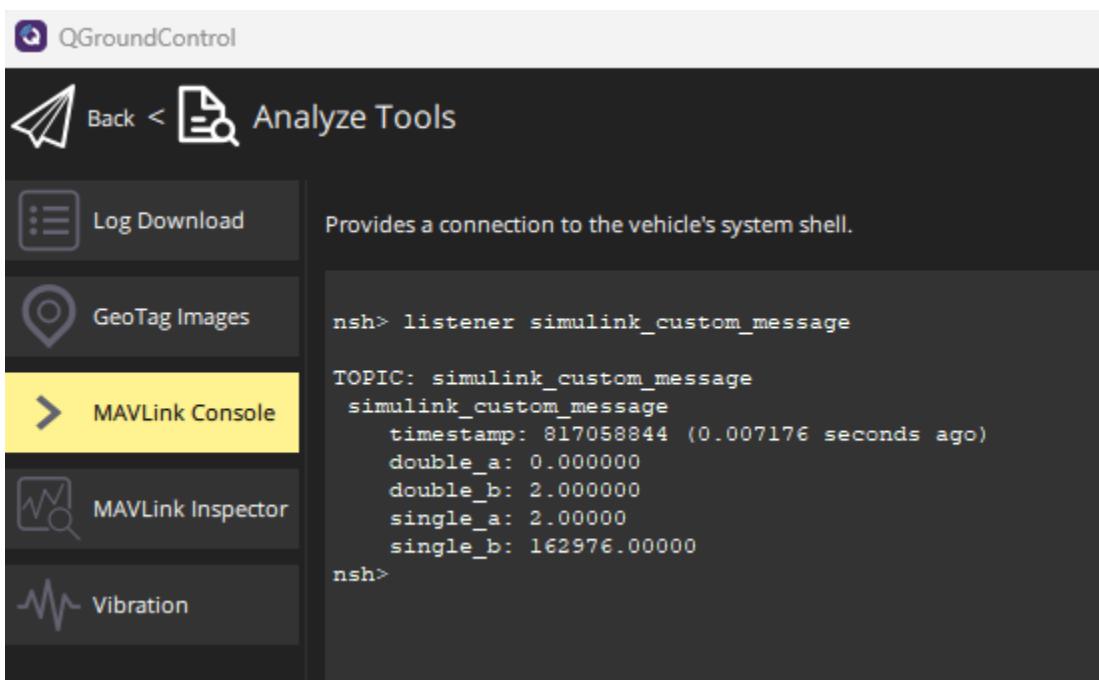
Deploy the Simulink model to Pixhawk Hardware

In the Simulink model, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. The build process for the model starts and it is deployed on the selected Hardware board.

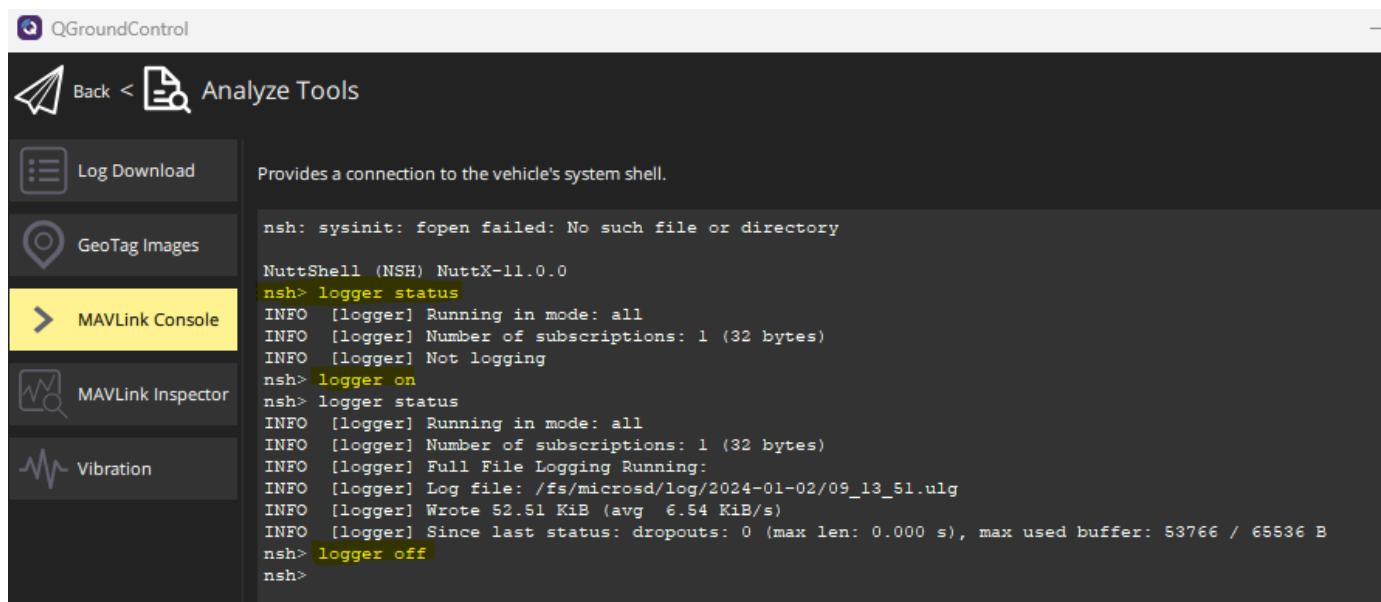
Step 4 - Start PX4 Logger

The system logger in PX4 is responsible for logging information. Since this example is not a flying demo, you need to manually switch the logger on and off. By default, the PX4 logger is configured to log when the system is armed.

1. Access the PX4 console by opening QGC and selecting **Analyze Tools > MAVLink Console** for logging the topics.



2. As an additional checkpoint before logging, ensure that the uORB message is being published by using `listener topic_name` command. For example, `listener SimulinkCustomMessage`.
3. In the PX4 console, use the **logger on** and **logger off** command to log the data. To verify the status of the logging process, use the **logger status** command.

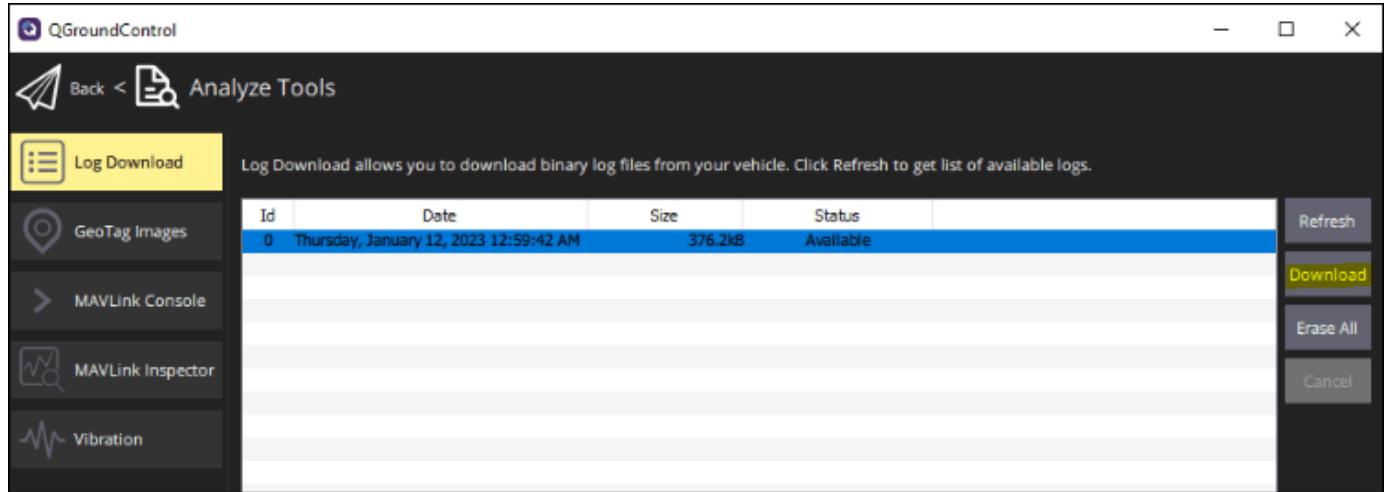


Note: If the logger is not started, use the **logger start** command to start it.

Step 5 - Retrieve the Data

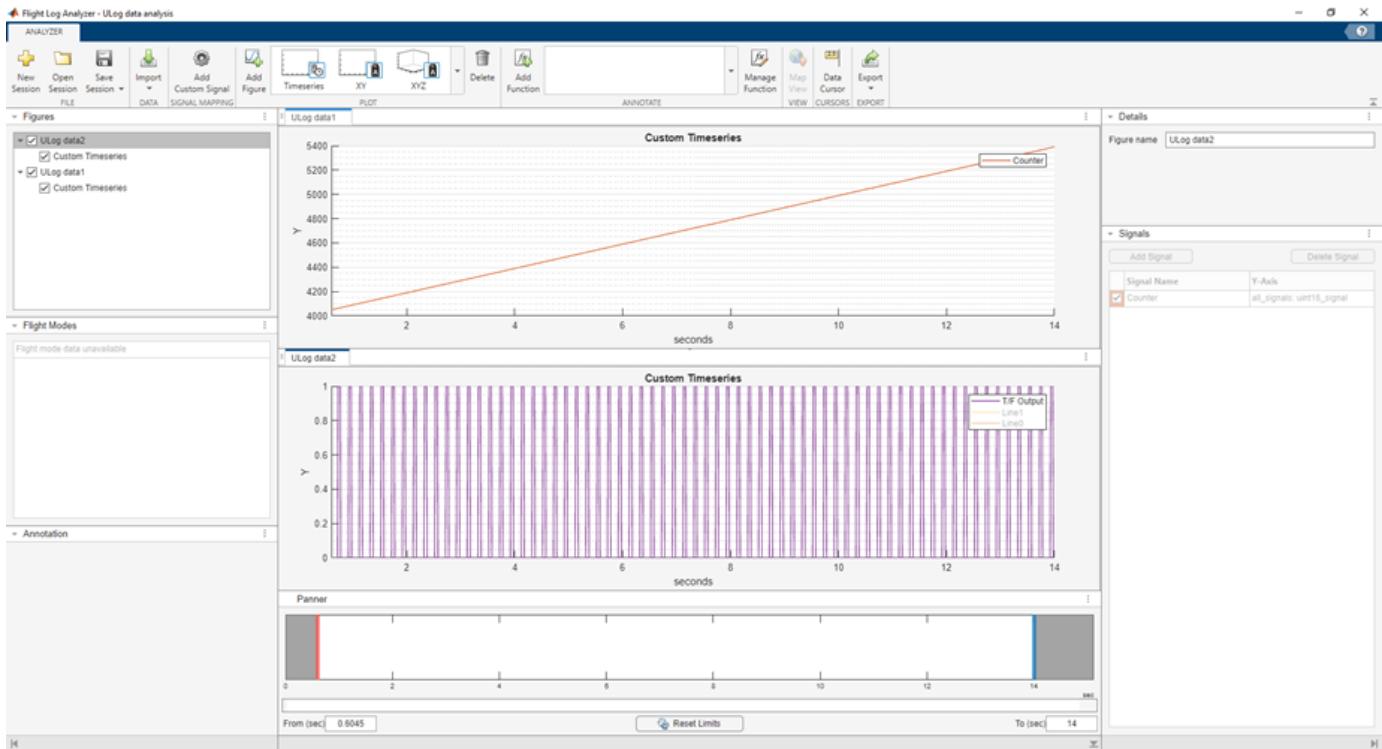
Download the logged flight data to your system using the **Log Download** screen in QgroundControl (QGC).

1. In the QGC, navigate to **Analyze Tools > Log Download**.
2. Click **Refresh** to get the latest logged flight data and then click **Download** to download the files.



Step 6 - Analyze ULog Data

You can analyze the downloaded signals in MATLAB by using `ulogreader`, which enables you to import the data into workspace for further analysis. You can also use the Flight Log Analyzer to plot the data directly from ULog file as shown below.



For more online tools, see Flight Log Analysis.

Task 2: Create Custom uORB Message and Log in ULog Format

For an advanced workflow, you can create a message your choice as per the requirements and log them in uLog format. The uORB message can be created using the `createPX4uORBMessage` MATLAB API.

Assuming you have a Simulink model that requires a set of signals to be logged in uLog format, you can create the necessary custom message using the API, as illustrated below.

Use the following command in MATLAB terminal to create the custom uORB message.

```
>> createPX4uORBMessage ('NewExampleMessage','int16 int16_signal', 'int32
int32_signal', ...

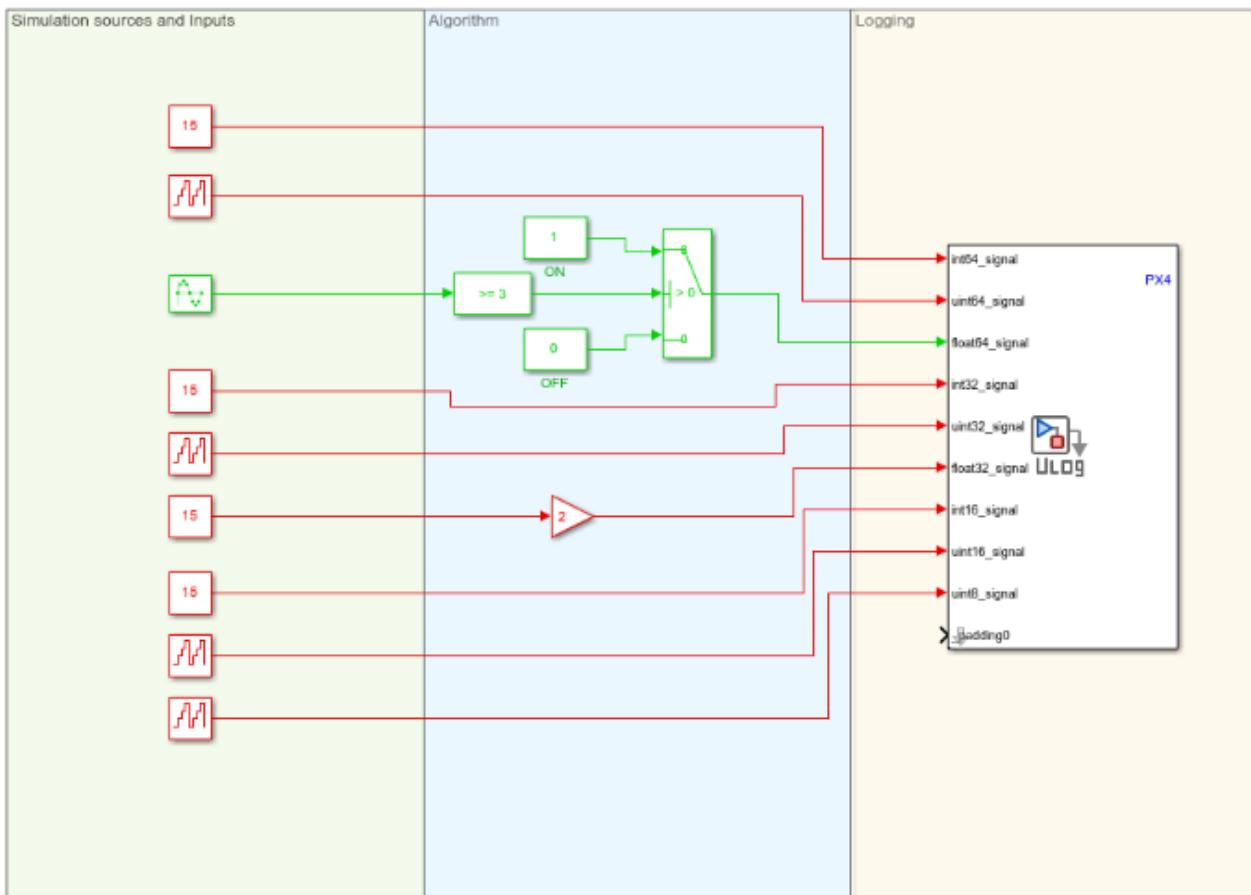
'int64 int64_signal','uint8 uint8_signal','uint16 uint16_signal','uint32
uint32_signal', ...

'uint64 uint64_signal','float32 float32_signal','float64 float64_signal')
```

To use this custom uORB message, build the PX4 firmware using the “Install UAV Toolbox Support Package for PX4 Autopilots in Linux”.

A representation of the Simulink model for the generated new custom message is shown below.

Logging Simulink signals in ULog format



To log signals using ULog format from the illustrated Simulink model perform the steps listed from Step 2 to Step 6.

Note: In Step2, use the topic name `new_example_message` to update the `logger_topics.txt`.

```
nsh > echo new_example_message >> logger_topics.txt
```

Replace the example model (`px4demo_ulog`) in Step 3 with the your specified model. Next, proceed with points 2, 3, 4, and 5 to successfully log the desired signals in your model.

For more information on uORB messages, see PX4 User Guide.

See Also

Related Examples

- “Log Signals on an SD Card”
- “Validate Multirotor UAV Model by Playing Back Flight Log in Simulink”

UVify IFO-S Autopilot and Nvidia Jetson Hardware-in-the-Loop (HITL) Simulation in Simulink

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to deploy and verify algorithms on UVify IFO-S Drone, with NVIDIA® Jetson™ as Onboard computer.

Using this example you,

- Deploy a flight control algorithm modeled in Simulink® to UVify IFO-S Autopilot.
- Deploy an autonomous algorithm to NVIDIA Jetson.
- Enable Scenario visualization in Unreal Engine®.
- Perform HITL Simulation of UAV Dynamics and sensors.

Limitation: The Unreal Engine simulation environment is supported only on Microsoft® Windows® system. If you are using a Linux® system, skip adding the 3D scenario simulation step and still be able to complete this example.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Go through the “PX4 Hardware-in-the-Loop System Architecture” document to understand the physical connections required to setup the Pixhawk® and Simulink for HITL Simulation.
- Configure and set up Pixhawk in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any Pixhawk Series board as the PX4 Autopilot board from the drop-down list. This example uses UVify IFO-S.
- Familiarize with the co-simulation framework for UAVs, see “Unreal Engine Simulation for Unmanned Aerial Vehicles”

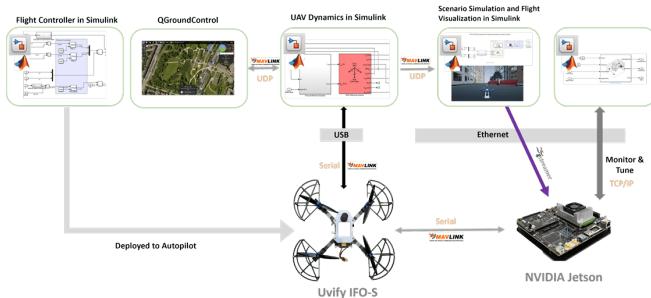
Required Third-Party Software This example requires this third-party software:

- QGroundControl (QGC)

Required Hardware To run this example, you will need the following hardware:

- UVify IFO-S
- Micro USB type-B cable
- Micro-SD card
- NVIDIA Jetson & power adaptor (recommended)
- Host PC Configured with MATLAB supported GPU, as shown in “GPU Computing Requirements” (Parallel Computing Toolbox). It is recommend to use GPU with compute capability of more than 5.

Workflow to Run Model on NVIDIA Jetson Along with Pixhawk in HITL Mode



The above diagram illustrates the PX4 and NVIDIA Jetson HITL setup and the physical communication between various modules.

This example uses four different Simulink models.

- Simulink model for Flight Controller to be deployed on PX4 Autopilot.
- Simulink model for UAV Dynamics and sensor simulation.
- Simulink model for Autonomous algorithm to be deployed on NVIDIA Jetson.
- Simulink model for flight visualization with Unreal Engine Simulation for Unmanned Aerial Vehicles.

To avoid performance degradation in MATLAB® due to three different Simulink models running at the same time, launch three separate sessions of same MATLAB.

- In the first session of MATLAB, the Flight Controller is deployed on Autopilot and the UAV Dynamics model will run on host computer communicating with Autopilot.
- In the second session of MATLAB, the Simulink model for flight visualization with Unreal Engine Simulation will be running. This can be skipped if you opt to not add the flight visualization.
- In the third session of MATLAB, the Simulink model for NVIDIA Jetson communicates with MATLAB on host computer using Monitor & Tune Simulation.

Step 1: Make Hardware Connections and Set Up the UVify IFO-S Drone in HITL Mode

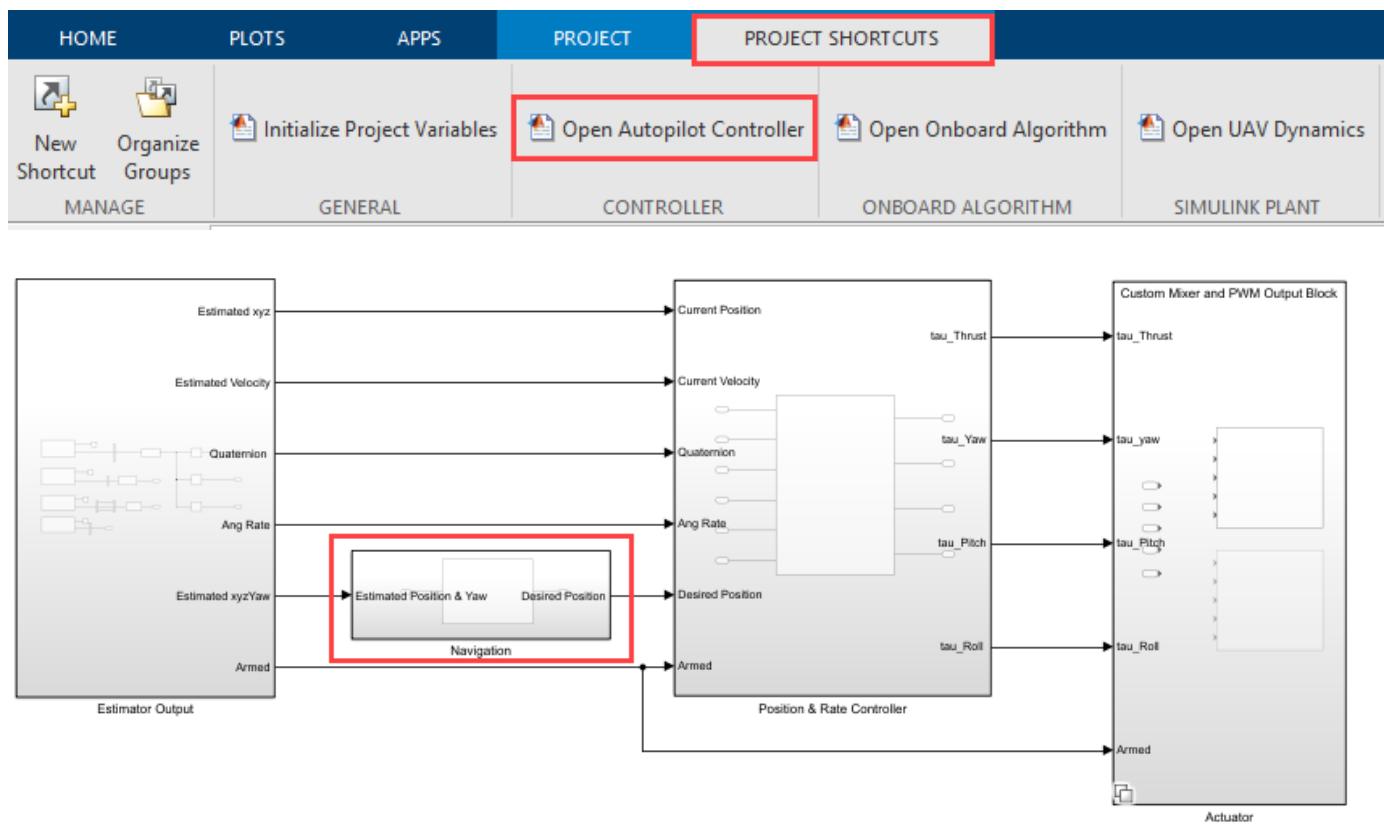
1. Connect your Autopilot board to the host computer using the USB cable.
2. Ensure that you have configured the Pixhawk board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
3. Ensure that you have setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”.
4. Setup and configure your NVIDIA Jetson on network using “MATLAB Coder Support Package for NVIDIA Jetson and NVIDIA DRIVE Platforms” (MATLAB Coder).

Tip: Power your NVIDIA Jetson using a separate A/C adapter so that the Autopilot can be independently powered On/Off without rebooting NVIDIA Jetson.

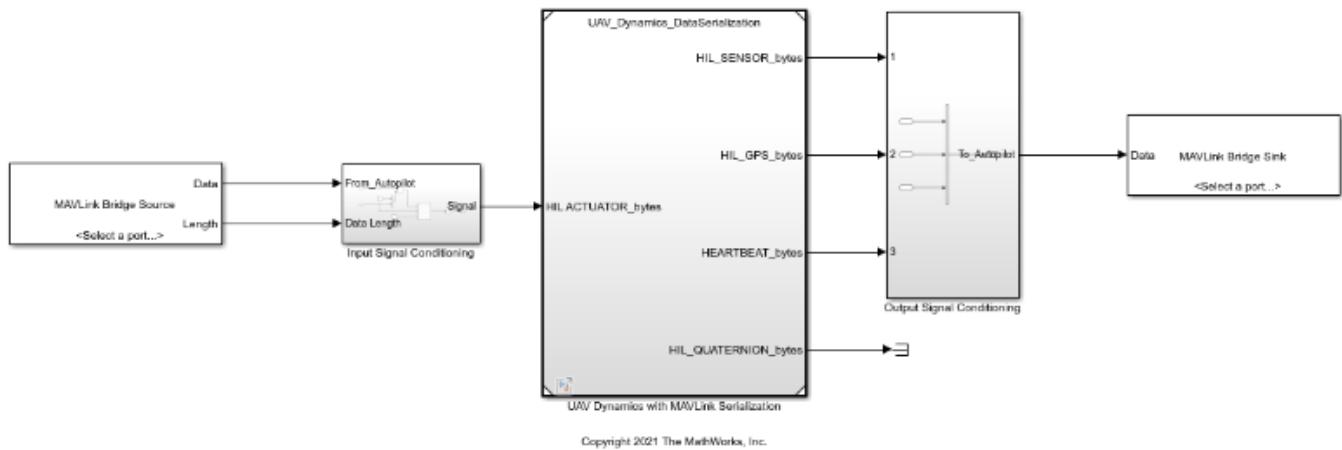
Step 2: Launch First Session of MATLAB and the MATLAB Project

The support package includes an example MATLAB project having the PX4 flight controller and the UAV to follow the mission set in the QGroundControl (QGC).

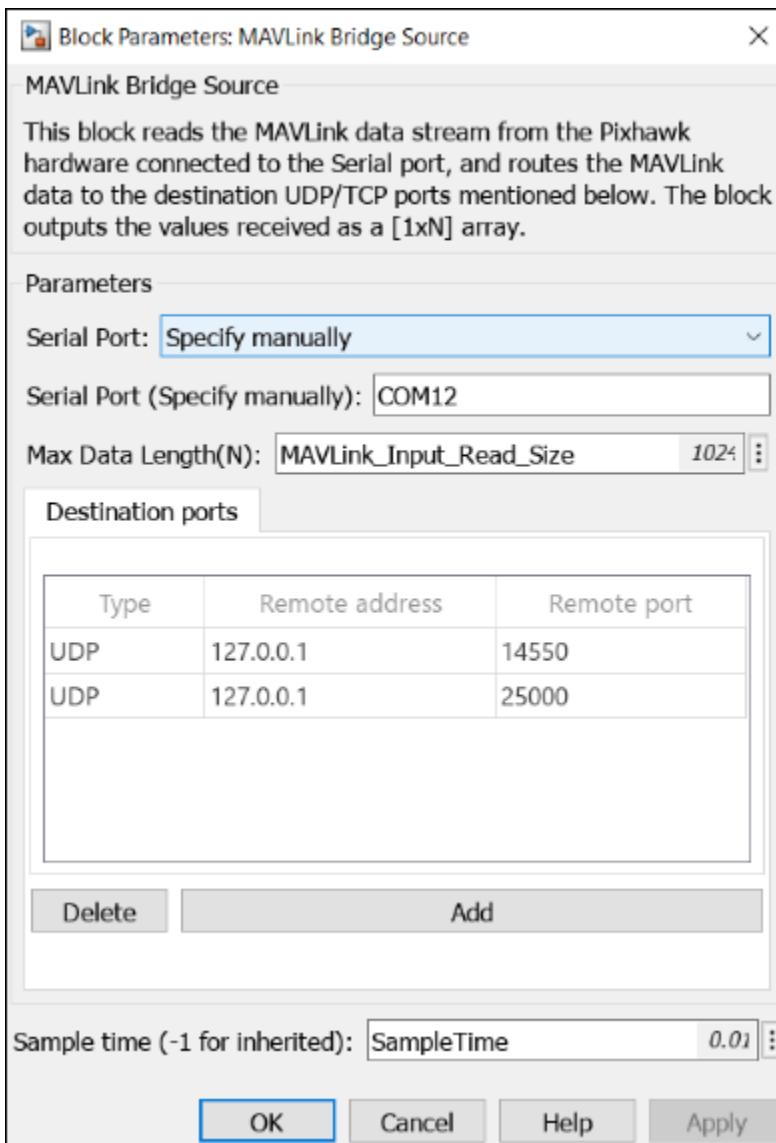
1. Open MATLAB.
2. Open the px4demo_HardwareInLoopWithSimulinkPlant MATLAB project.
3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open Autopilot Controller** to launch PX4 Controller named *Quadcopter_ControllerWithNavigation*.



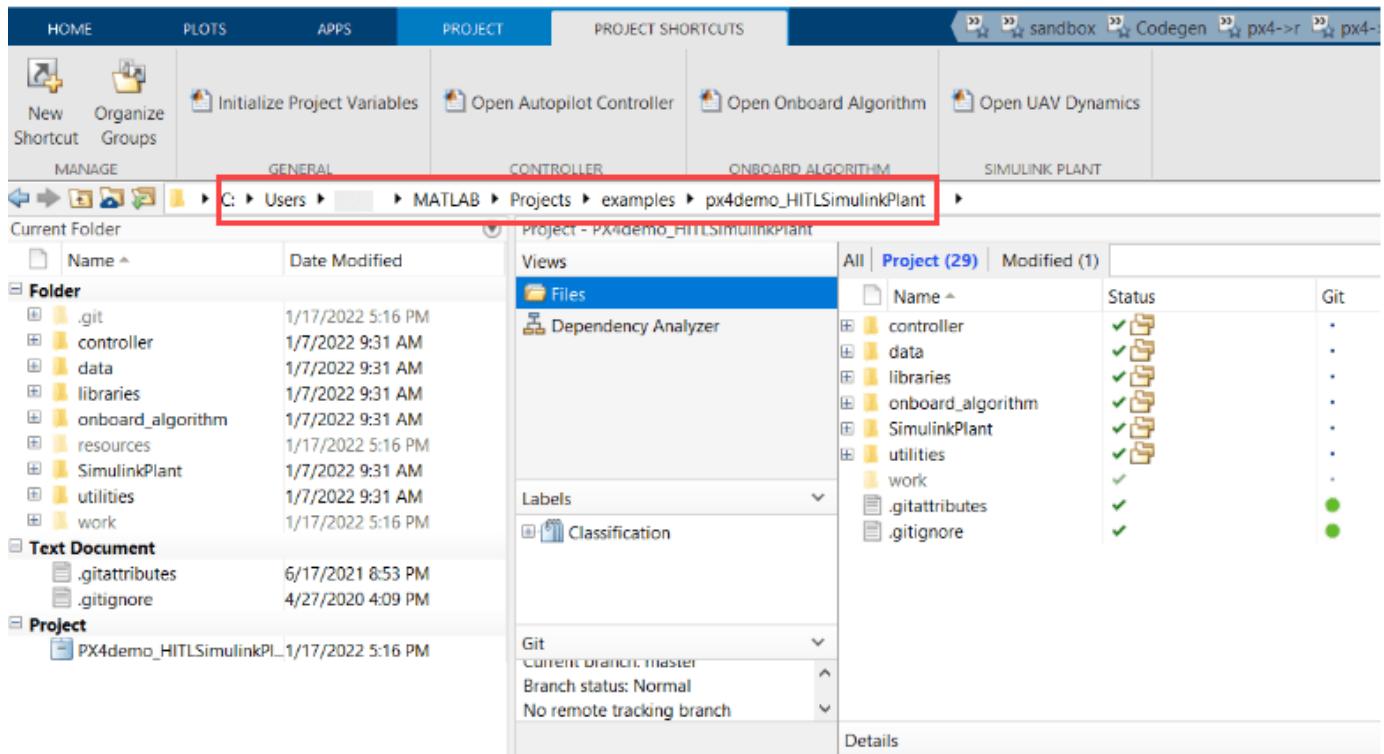
4. Navigate to Navigation subsystem. This is a Variant Subsystem with guidanceType as the variant control variable. Define guidanceType = 1 in the base workspace to choose the navigation subsystem for this example.
5. In the **Project Shortcuts** tab, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.



6. Double click on MAVLink Bridge source and sink blocks and set the serial port of autopilot board.
7. Add a local host connection for flight visualization in MAVLink Bridge Source block. Set the port to 25000.

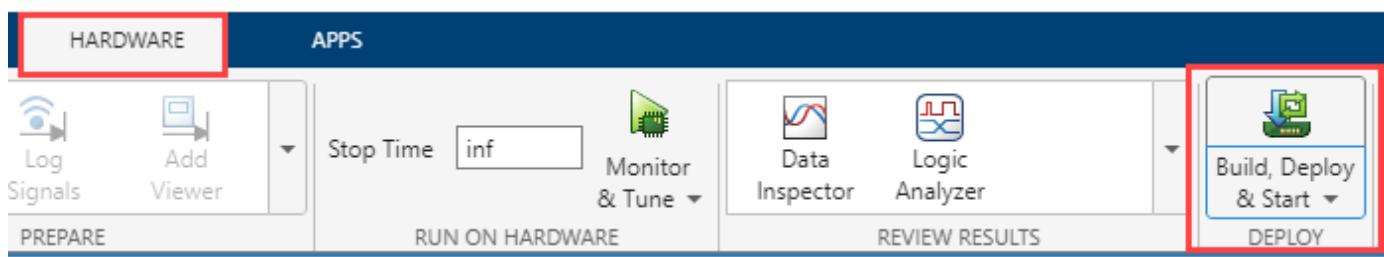


8. Copy the MATLAB Project Path to clipboard.



Step 3: Configure Simulink Controller Model for HITL Mode

1. Follow the instructions mentioned in "Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation". Select UVify IFO-S as the Hardware Board.
2. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*.



The code will be generated for the Controller model and the same will be automatically deployed to the autopilot board.

After the deployment is complete, QGroundControl will be automatically launched.

Note: If you are using Ubuntu®, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

```
./QGroundControl.AppImage
```

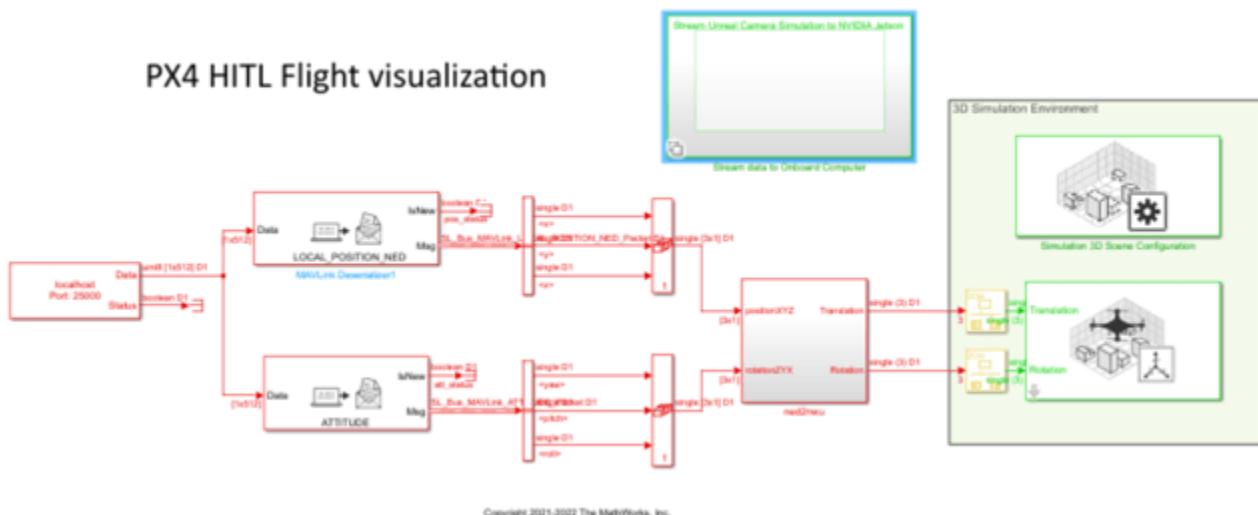
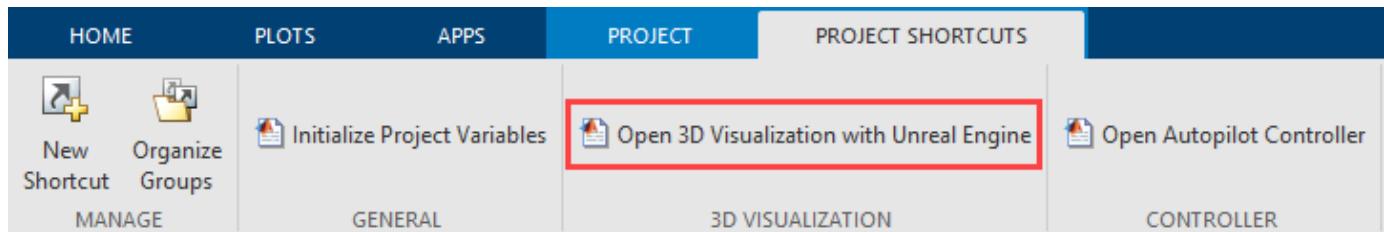
Step 4: Launch Second Session of MATLAB and Open the Flight Visualization Model

Note: Skip this step if you do not opt for flight visualization with PX4 HITL.

1. Open the second instance of the same MATLAB version. In this MATLAB session, The Simulink model for scenario simulation and flight visualization using unreal environment runs.

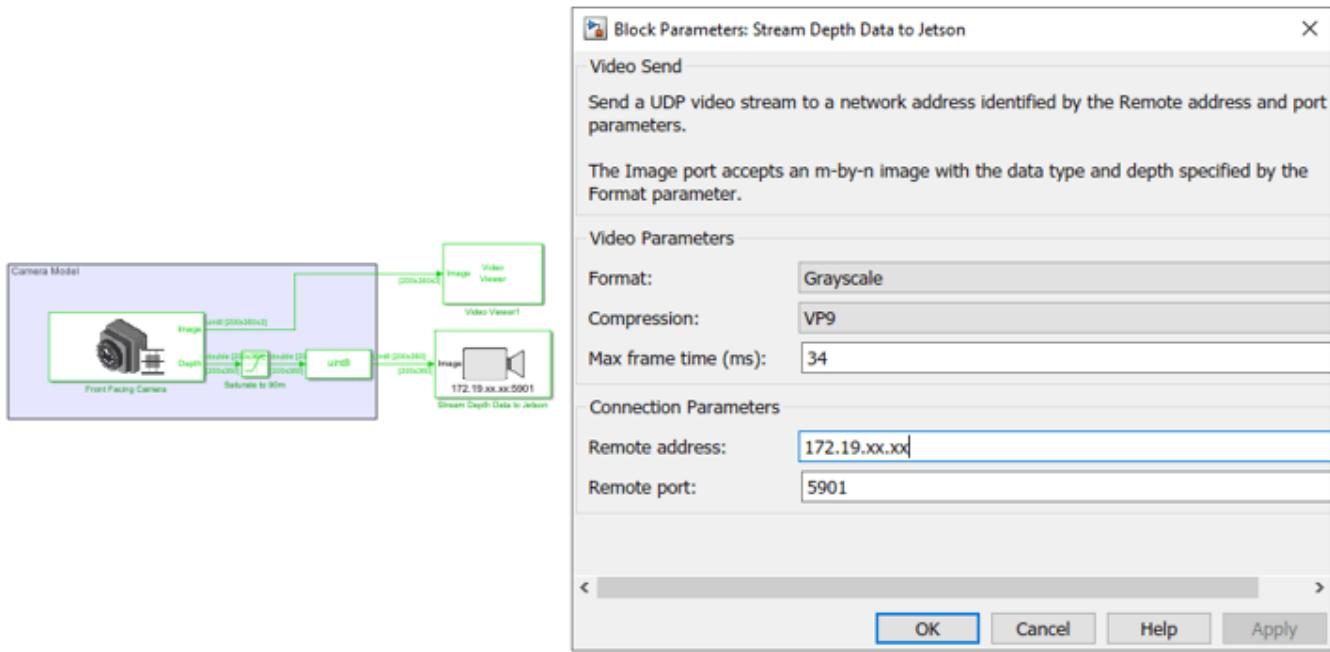
Ensure that your Host PC is Configured with MATLAB supported GPU (GPU with compute capability of more than 5).

2. Open the px4demo_HardwareInLoopWithSimulinkPlant MATLAB project.
3. Once the Simulink project is open, click the **Project Shortcuts** tab on the MATLAB window and click **Open 3D Visualization with Unreal Engine** to launch the onboard model named *Unreal_3DVisualization*.



In this model, The MAVLink data from the PX4 Autopilot is received over UDP (port : 25000) and is used to decode the position and attitude data of the UAV. After coordinate conversion, it is then passed to the Simulation 3D UAV Vehicle block for flight visualization.

4. Enable the streaming of simulated depth sensor data in NVIDIA Jetson by updating the variable `enableOnboardStreaming` to 1.
5. The Simulation 3D Camera block provides the Camera image from the Unreal environment. In this example you stream the depth images from the camera block to NVIDIA Jetson using Video Send block. Double-click block to open the block Parameters dialog box. Add the IP address of onboard computer (NVIDIA Jetson) in the dialog box and click **OK**.



6. On the **Simulation** tab, click **Run** to simulate the model. Once the model starts running, you will see the Unreal simulation environment getting launched. A sample screen is shown below.

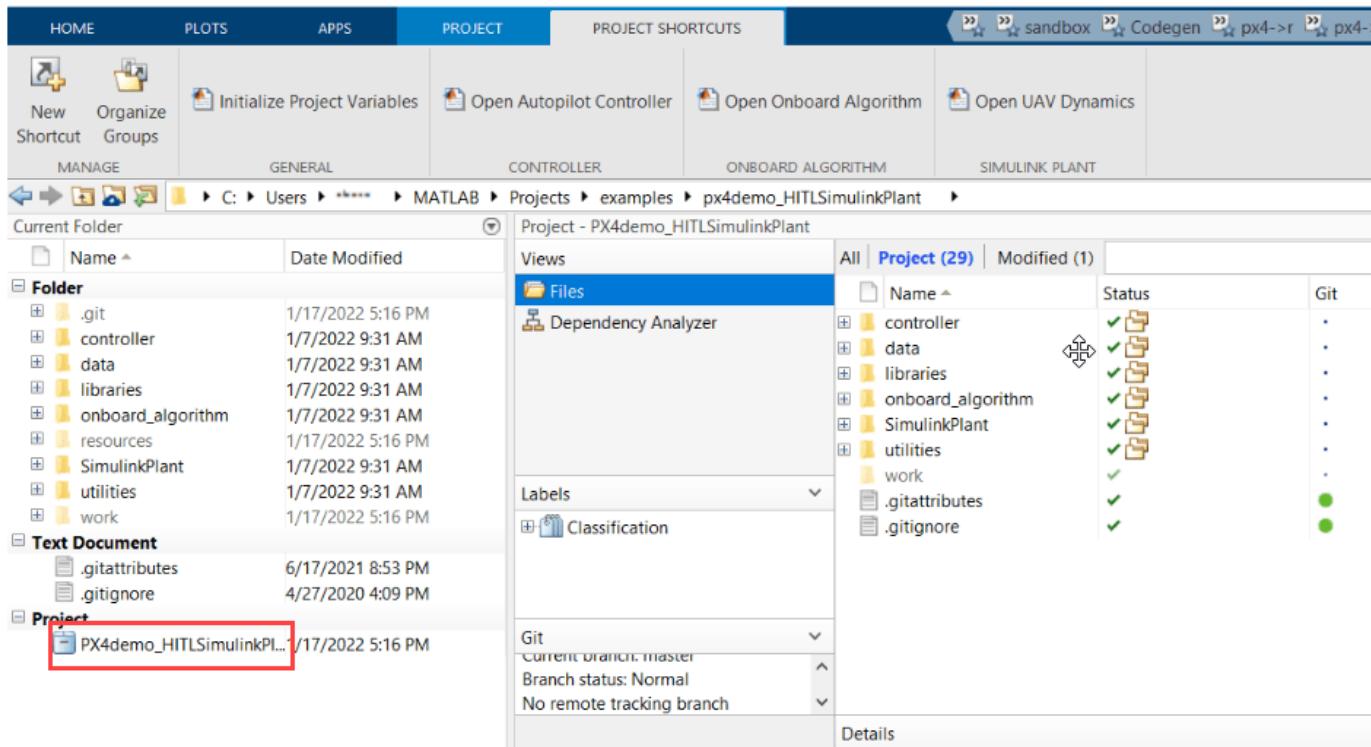
**Step 5: Launch Third Session of MATLAB and the Onboard Model**

1. Open the third instance of the same MATLAB version.
2. Navigate to the project path previously copied in Step 2 in current MATLAB.

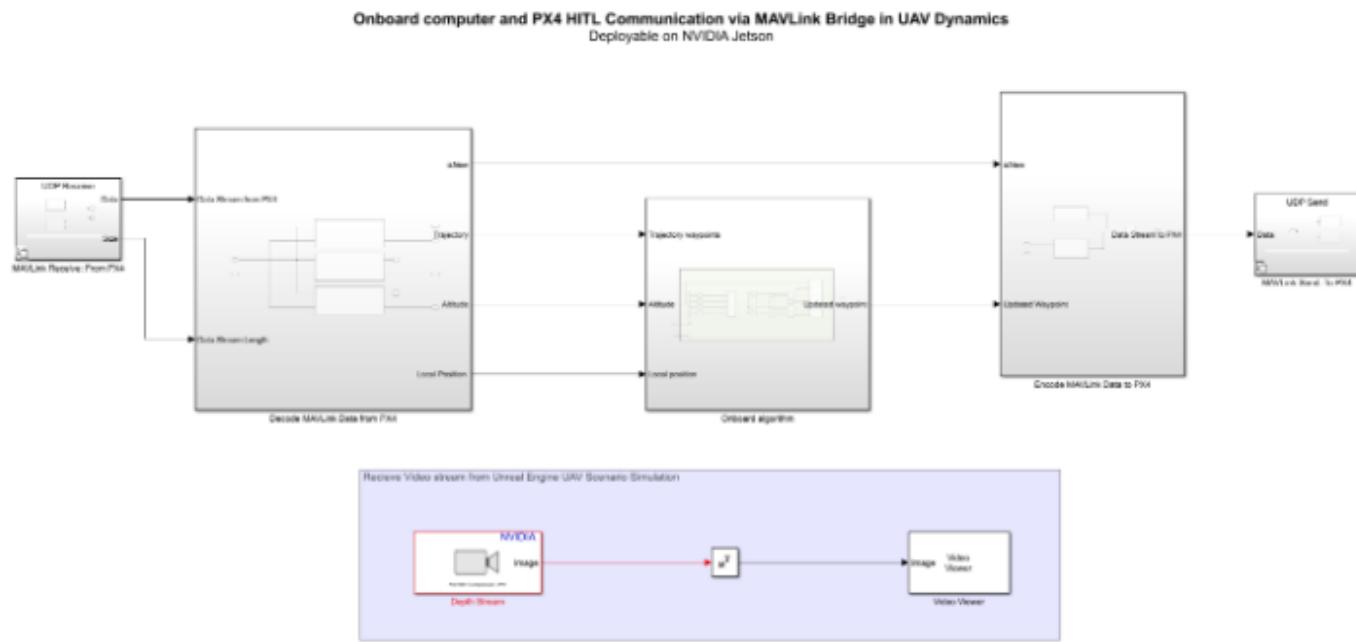
```
fx >> cd C:\Users\██████\MATLAB\Projects\examples\px4demo_HITLSimulinkPlant
```

3. Click on the .prj file to launch the same Project in current MATLAB.

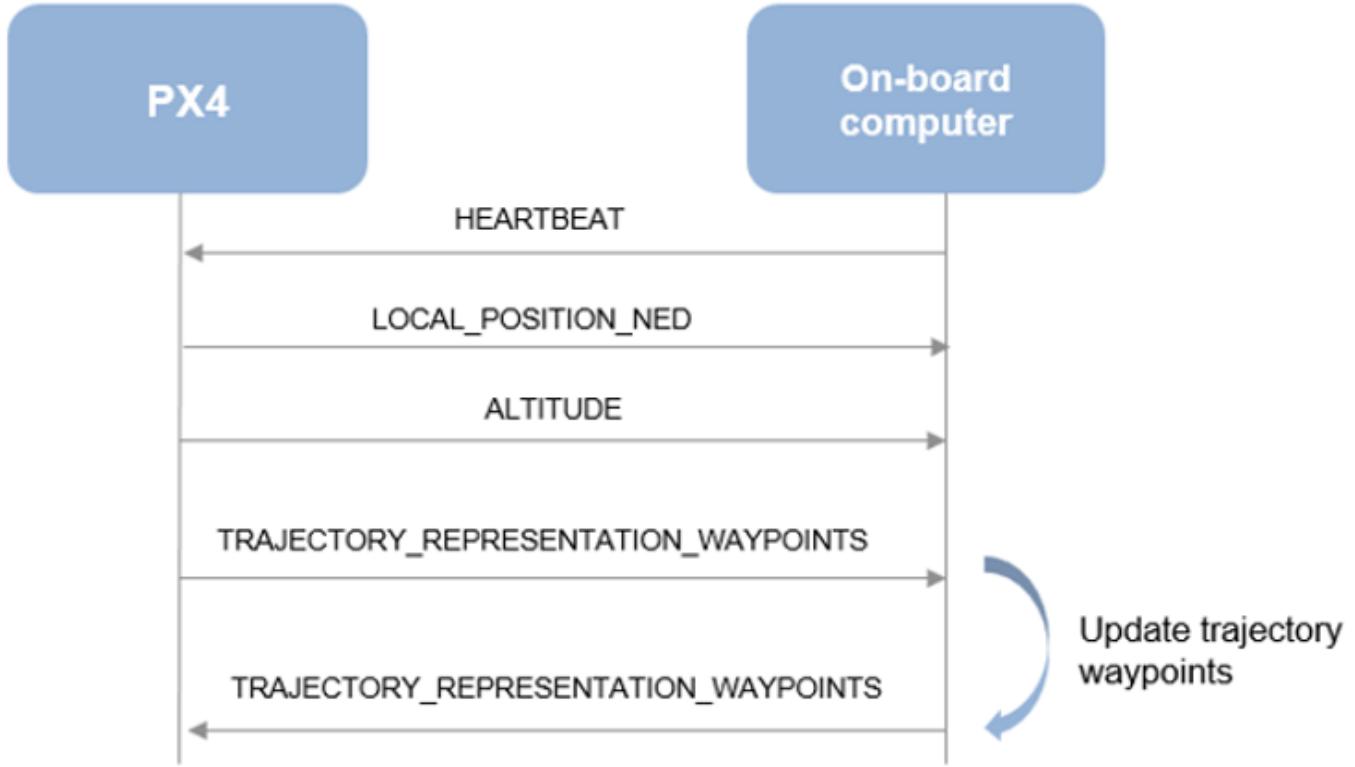
1 Blocks



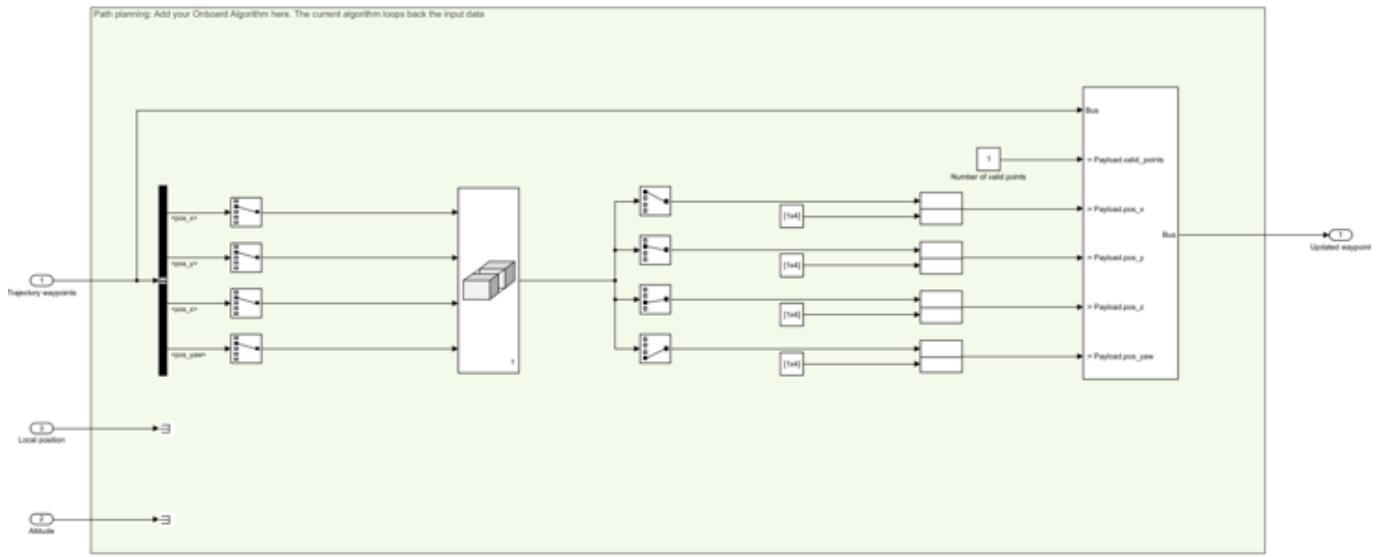
4. In the **Project Shortcuts** tab, click **Open Onboard Algorithm** to launch the onboard model named *Onboard_Autopilot_Communication*.



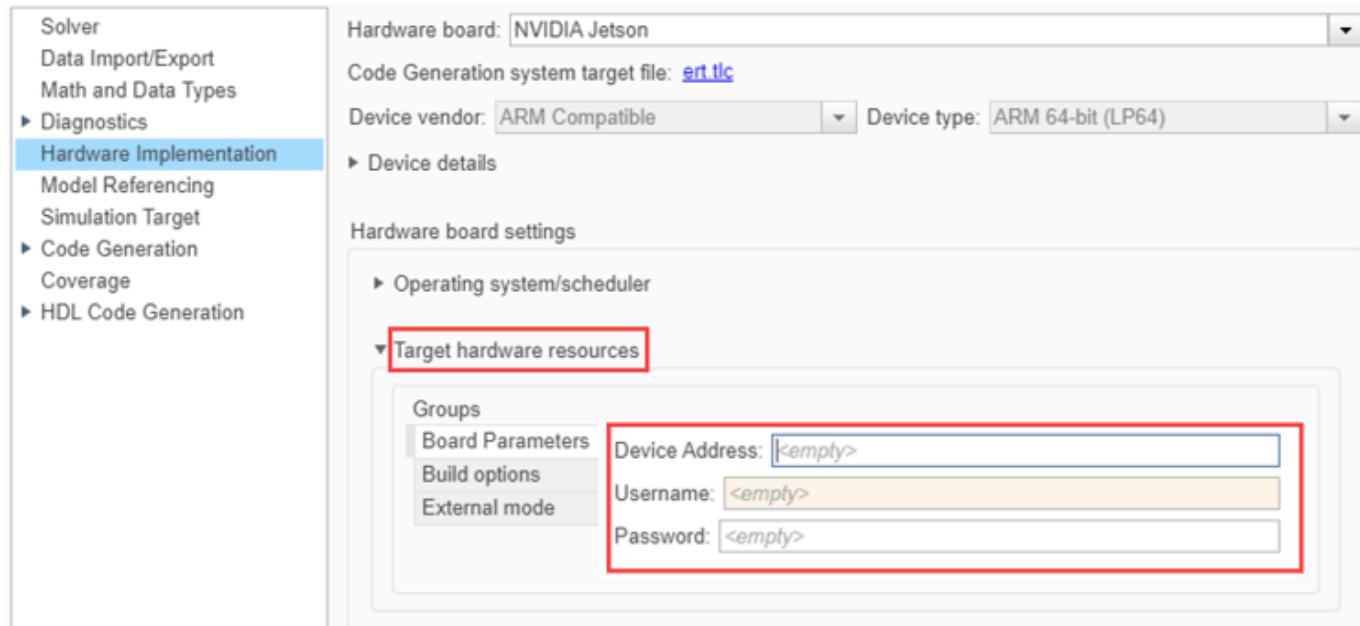
This model implements the PX4 path planning interface using MAVLink Serializer and MAVLink Deserializer blocks. For more information, see [PX4 Path Planning Interface](#). The MAVLink messages that are exchanged as part of this interface are shown in the following diagram.



5. This example provides a basic onboard logic, which is sending back the received waypoint as the updated waypoint. You can consider designing your own onboard logic after completing this basic example.



6. Navigate to Target hardware resource > Board Parameters, enter the IP address of the NVIDIA Jetson and your login credentials.



7. This example uses MAVLink over serial ports to communicate between the Jetson and autopilot board. Activate serial port variant by setting this workspace variable.

```
onboardMAVLinkSource = 1
```

8. Disable any third party tools that might use the serial ports on Jetson such as MAVLink router.

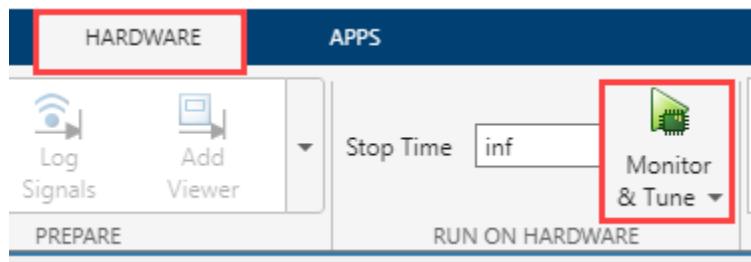
Open QGC and set the following parameters to use the existing serial link between Jetson and UVify autopilot. For more information on UVify serial ports, see “Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards” on page 3-2.

- MAV_0_MODE = onboard
- MAV_0_CONFIG = TELE1
- SER_TEL1_BAUD = 230400 8N1

9. Configure the Network Video Receive block to receive the depth data from Unreal environment. Note that the port number and compression parameters in the Network Video Receive block are same as those of the corresponding Video Send block streaming camera image from Unreal engine.

Note: If you do not opt for flight visualization with PX4 HITL, comment the Network Video Receive block.

9. Click **Monitor & Tune** from **Run on Hardware** section of **Hardware** tab in the Simulink Toolbar.



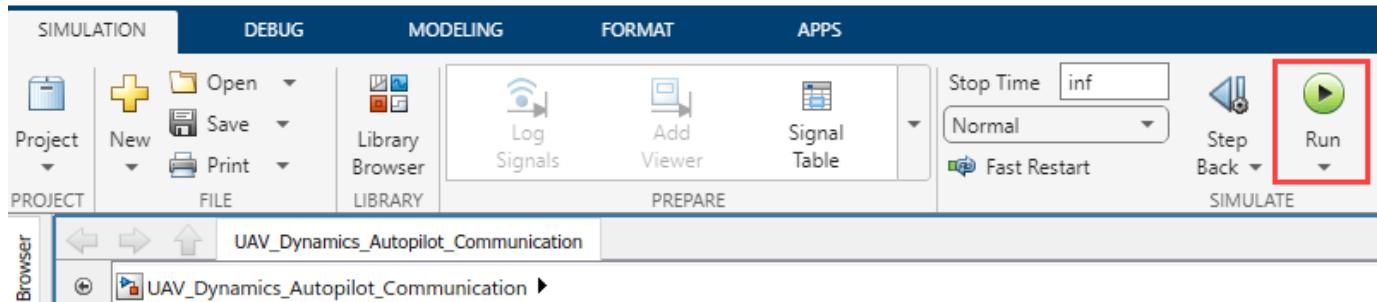
The code will be generated for the Controller model and the same will be automatically deployed to NVIDIA Jetson. NVIDIA Jetson should start communicating with host over Monitor & Tune Simulation.

Note: Ensure that there are no other deployed models from Simulink are running in NVIDIA Jetson. If you are unable to verify, reboot the Pixhawk hardware board before starting the deployment.

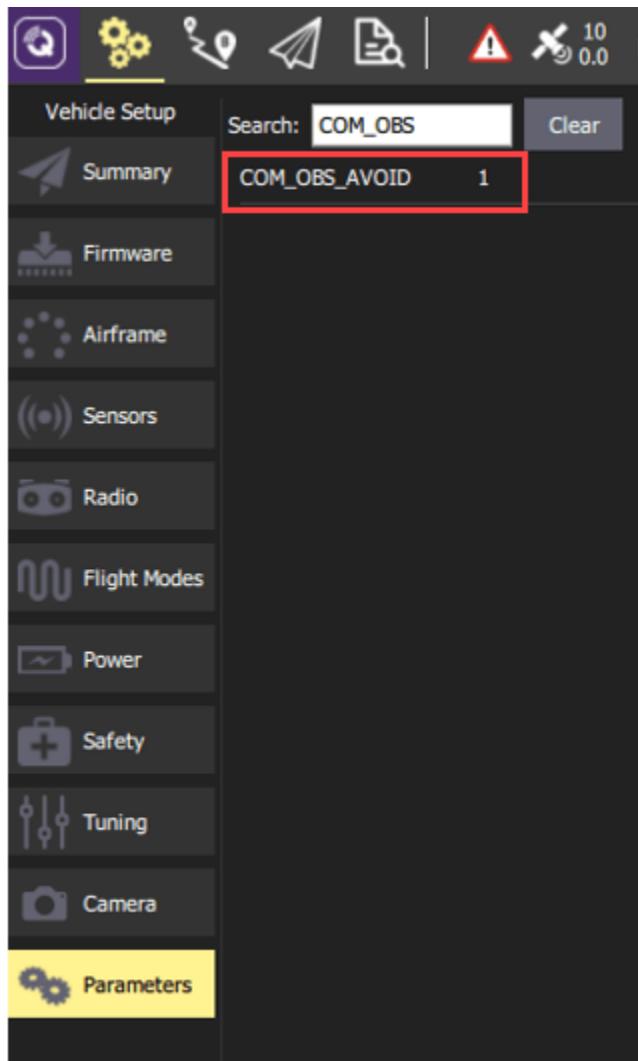
The algorithm in NVIDIA Jetson also communicates with the Plant model *UAV_Dynamics_Autopilot_Communication* over UDP.

Step 6: Run the UAV Dynamics Model, Upload Mission from QGroundControl and Fly the UAV

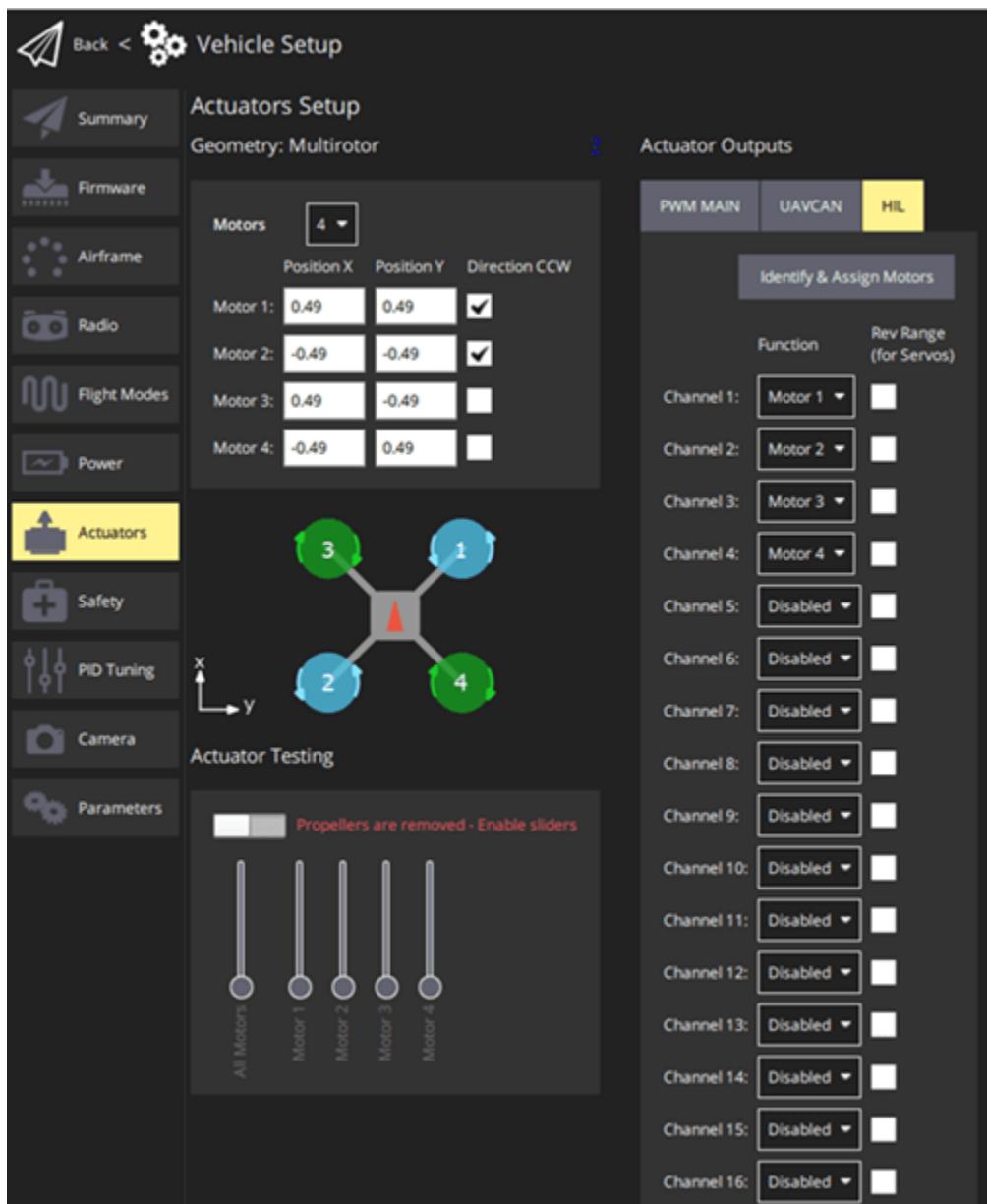
1. In the Simulink toolbar of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model.



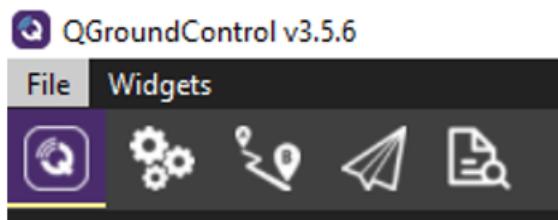
2. Set the PX4 parameter `COM_OBS_AVOID` enabling the PX4 path planning interface. Navigate to **Parameters** from the main menu and set the `COM_OBS_AVOID` parameter value to 1.



3. In the **Parameters** tab, set the `COM_DISARM_PRFLT` parameter value to -1.
2. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



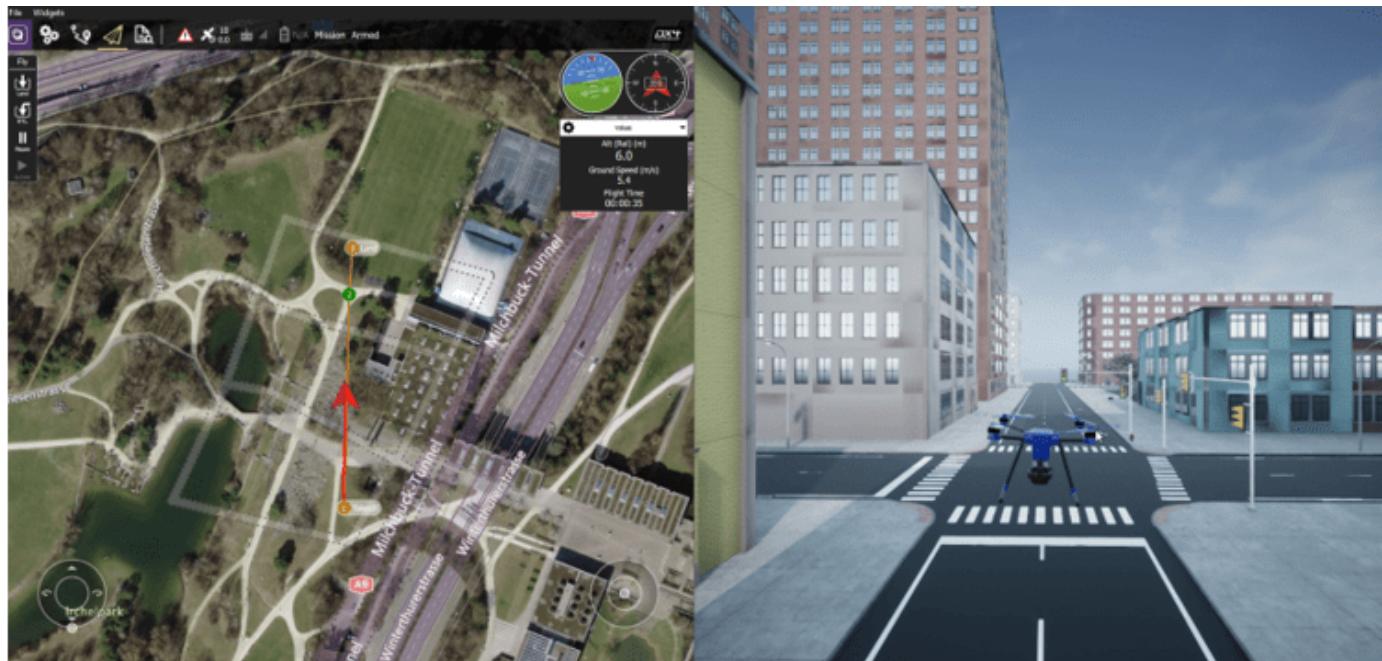
4. In the QGC, navigate to the *Plan View*.



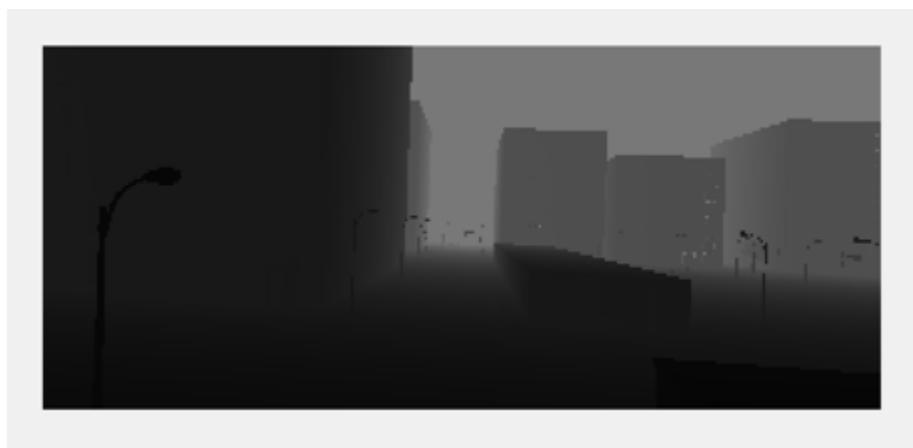
5. Create a mission. For information on creating a mission, see Plan View.

After you create a mission, it is visible in QGC.

6. Click Upload button in the QGC interface to upload the mission from QGroundControl.
7. Navigate to *Fly View* to view the uploaded mission.
8. Start the Mission in QGC. The UAV should follow the mission path.
9. Observe the flight visualization in the Unreal Engine.



10. Go to the *Onboard Autopilot Communication* model and validate the depth image streamed to Jetson during the flight in the Video Viewer (from Computer Vision Toolbox™).



Troubleshooting

- While Simulating the visualization model in Step 4, you might get any STD exception errors such as, "some module could not be found".

Change the compiler to Microsoft Visual C++ 2019 using `mex -setup c++` command to fix the issue.

- "Avoidance system not available" warning when starting a mission. This warning occurs because the communication between NVIDIA Jetson and PX4 HITL is not established.

Ensure that host PC and NVIDIA Jetson are connected to the same network. Try pinging NVIDIA Jetson from the host PC and vice versa. Also, double-check the NVIDIA Jetson IP address entered in the MAVLink Bridge blocks as mentioned in 6th point of Step 2 and Host PC IP address entered in the UDP send block as mentioned in 7th point of Step 5.

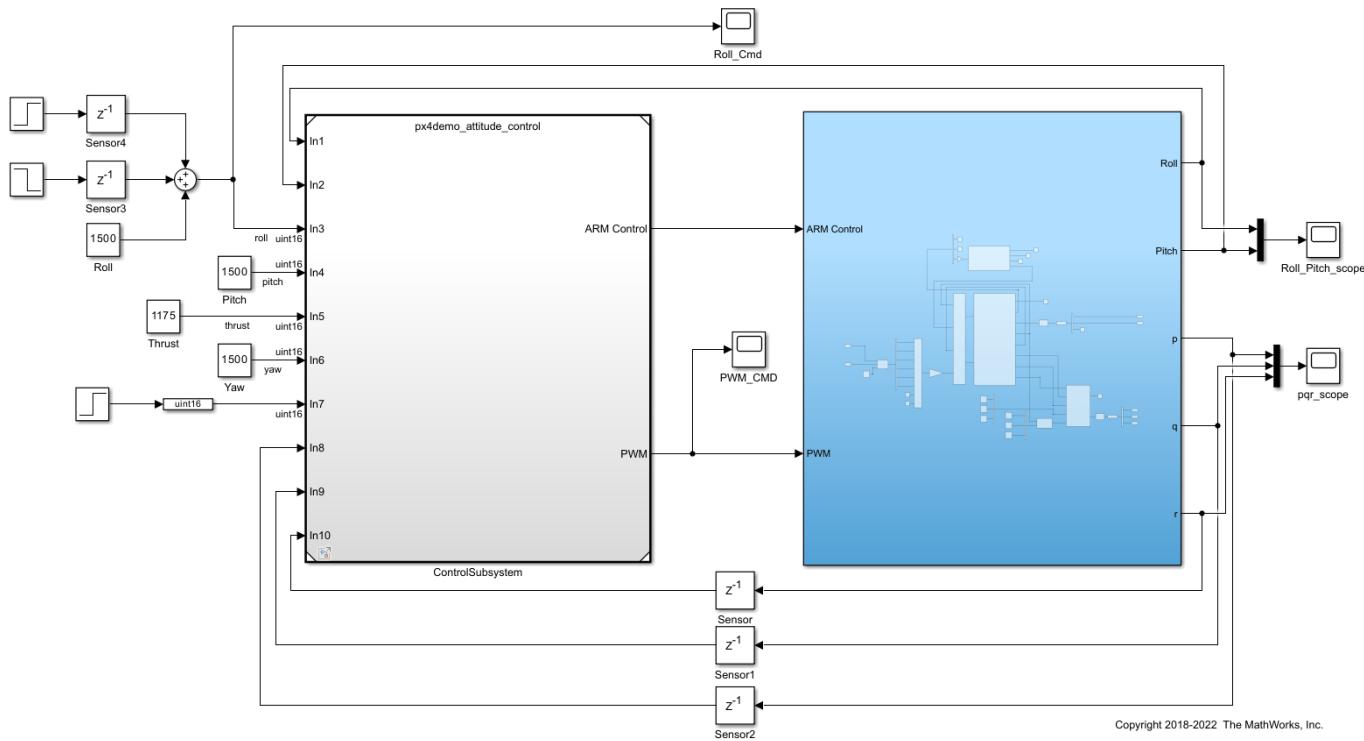
Plant and Attitude Controller Model for Hexacopter

The UAV Toolbox Support Package for PX4® Autopilots contains a plant and an attitude controller model to fly a hexacopter drone that uses a Pixhawk® Series flight controller.

Simulate the Plant Model for Hexacopter

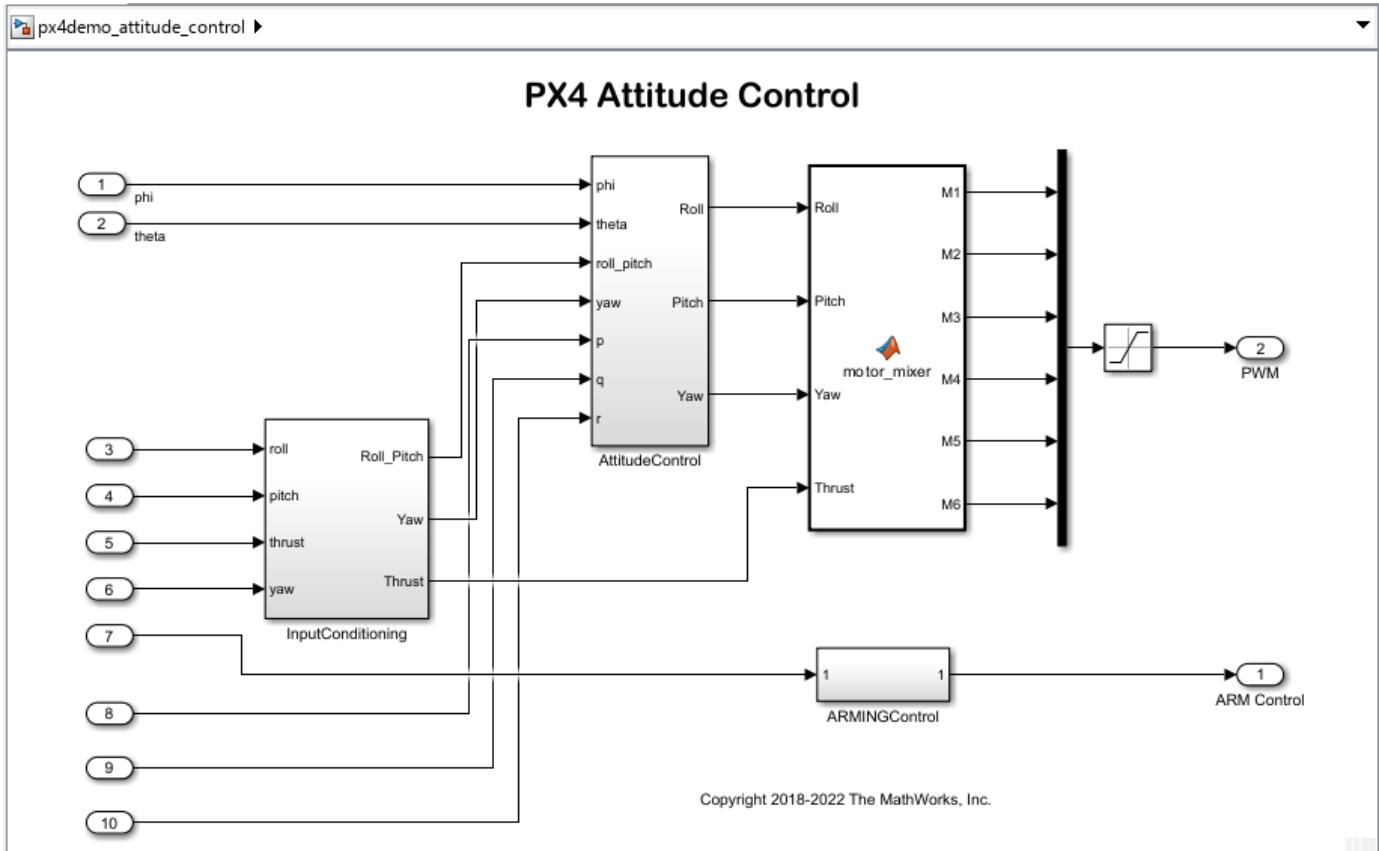
The plant model, *px4demo_attitude_plant*, simulates the 6 DOF, and it outputs the roll, pitch, yaw and thrust values, which are then fed to the InputConditioning subsystem in the *px4demo_attitude_control* model. The attitude controller generates PWM pulse widths which are then provided to the plant as feedback.

Open the *px4demo_attitude_plant* plant model.



This plant model simulates the behavior when the roll command from the RC Transmitter varies; the pitch, yaw and thrust values are kept constant. The model can be modified to simulate changes for pitch and yaw as well.

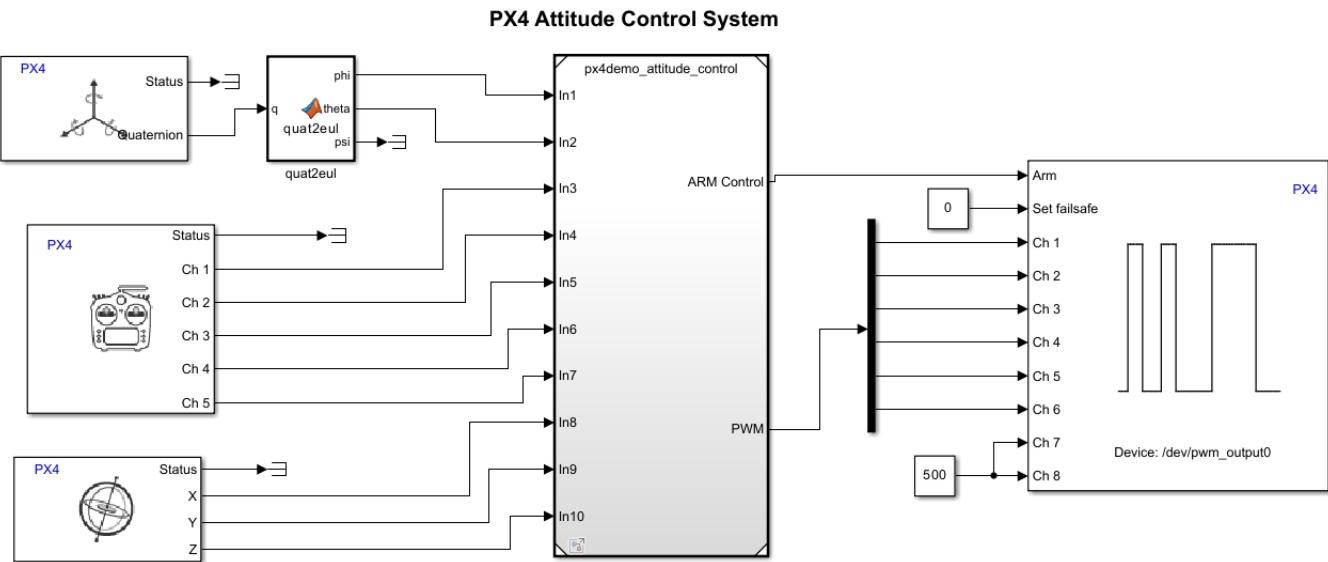
The *px4demo_attitude_control* subsystem takes the roll, pitch, yaw and thrust values as input from user. It also accepts the vehicle attitude and IMU measurements for gyroscope. These values are then fed to the PID controller. The output of PID controller is fed to the mixer matrix for hexacopter to output the PWM pulse widths. The PID controller might need to be tuned according to your airframe. The mixer matrix will vary from airframe to airframe.



Generate Code for Controller for Hexacopter

The controller model, *px4demo_attitudeSystem*, gathers the inputs from Gyroscope, Radio Controller and Vehicle Attitude block and sends them to the attitude controller *px4demo_attitude_control*, which returns the PWM pulse widths which are then published.

Open the *px4demo_attitudeSystem* controller model.



Copyright 2018-2022 The MathWorks, Inc.

Build the above model and deploy it to the selected Pixhawk Series flight controller.

Adapting the Plant and Attitude Controller for Other Airframes

For modifying the controller for any other airframes, the corresponding mixer matrix needs to be added by replacing the existing motor_mixer function block in px4demo_attitude_control model. The PID controller also needs to be tuned according to the airframe.

PX4 Autopilot in Hardware-in-the-Loop (HITL) Simulation with Speedgoat in Simulink

This example shows how you the use the UAV Toolbox Support Package for PX4® Autopilots to deploy a quadcopter plant dynamics model into Speedgoat Real-Time target machine and perform hardware-in-the-loop simulation. Additionally, you can validate your control logic running on PX4 Autopilot against nominal and adverse conditions such as sensor failures.

In summary, this example demonstrates the following workflows.

- Real-Time simulation of plant and sensors in Speedgoat®.
- Controller deployment to PX4 Autopilot.
- Photorealistic simulation using Unreal Engine®, based on plant states obtained from Speedgoat.

The plant model establishes communication with PX4 Autopilot to exchange sensor and actuator data. Furthermore, it shares the UAV ground truth, including position and orientation, with the host PC running the photorealistic simulation.

Prerequisites

- If you are new to Simulink®, watch the Simulink Quick Start video.
- Complete the “Speedgoat and Simulink Real-Time Setup” instructions to setup Speedgoat and Simulink Real-Time™.
- Configure and set up Pixhawk® in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.
- Setup the PX4 Firmware as mentioned in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. In the **Select a PX4 Autopilot and Build Target** screen, select any Pixhawk Series board as the PX4 Autopilot board and corresponding build target having `_multicopter` keyword from the drop-down list.

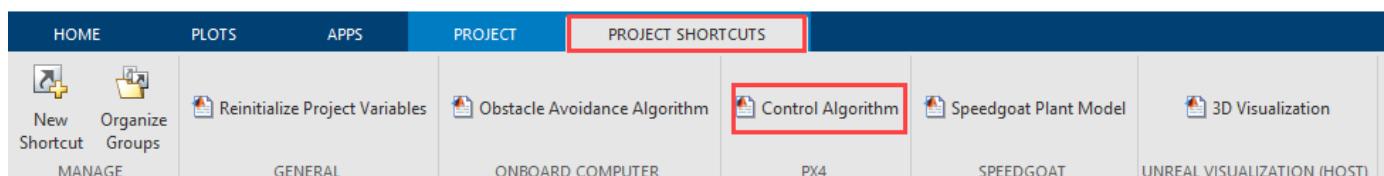
Required Third-Party Software

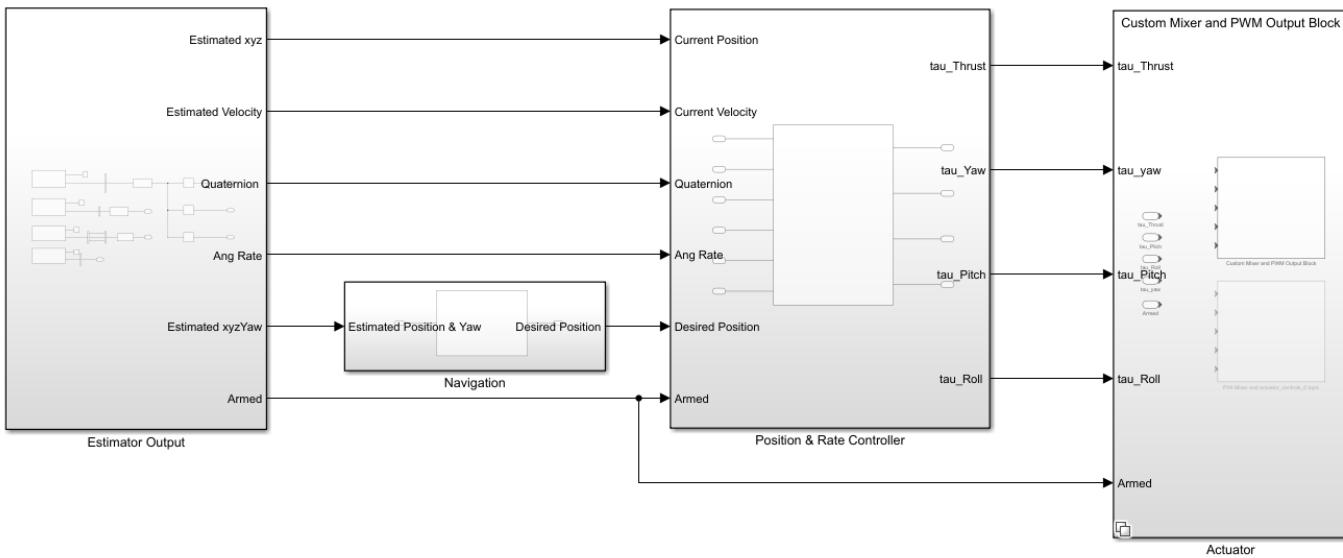
This example requires this third-party software:

- QGroundControl (QGC)

Launch First Session of MATLAB and MATLAB Project

1. Launch first session of MATLAB®.
2. Open the px4demo_HITLSimulinkPlantSpeedgoat project.
3. Once you open the project, click the **Project Shortcuts** tab on the MATLAB toolbar and click **Control Algorithm** to launch the controller model (*Quadcopter_ControllerWithNavigation*).





4. In the **Project Shortcuts** tab, click **Speedgoat Plant Model** to launch the Simulink plant model (*UAV_Dynamics_Speedgoat*).



Configure Simulink Controller Model for HITL Mode

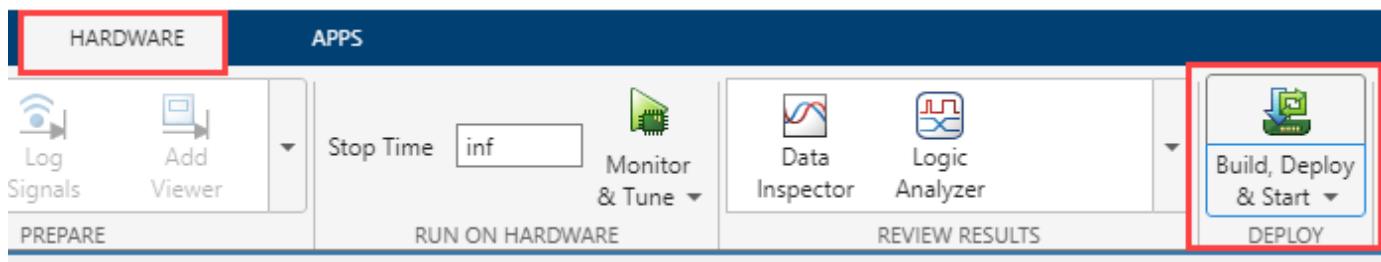
1. The controller model is configured to run in HITL mode. If you are not using this example model, follow the instructions as described in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”.

Note: This step is not required in the pre-configured model. Complete this step if you have changed the hardware or not using the pre-configured model.

2. PX4 PWM Output block does not work for targets such as Uvify IFO-S board. If you want to deploy the controller to Uvify IFO-S or if you want to allow PX4 to handle the mixing aspect, then select PX4 Mixer and actuator_controls_0 topic variant by setting the following parameter.

```
useCustomMixer = false;
```

2. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip for the Controller model *Quadcopter_ControllerWithNavigation*.



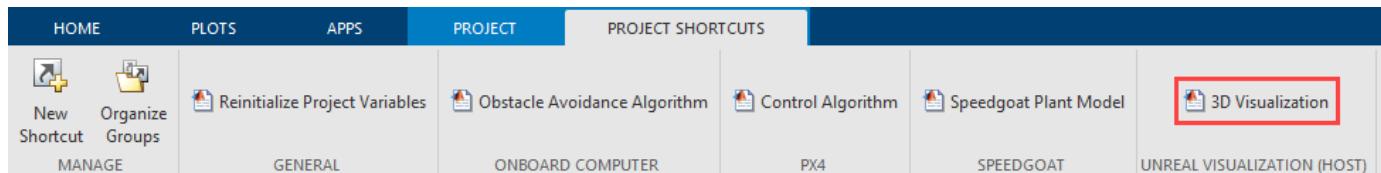
This deploys the controller code on the Pixhawk hardware and launches the QGroundControl (QGC).

Note: If you are using Ubuntu®, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

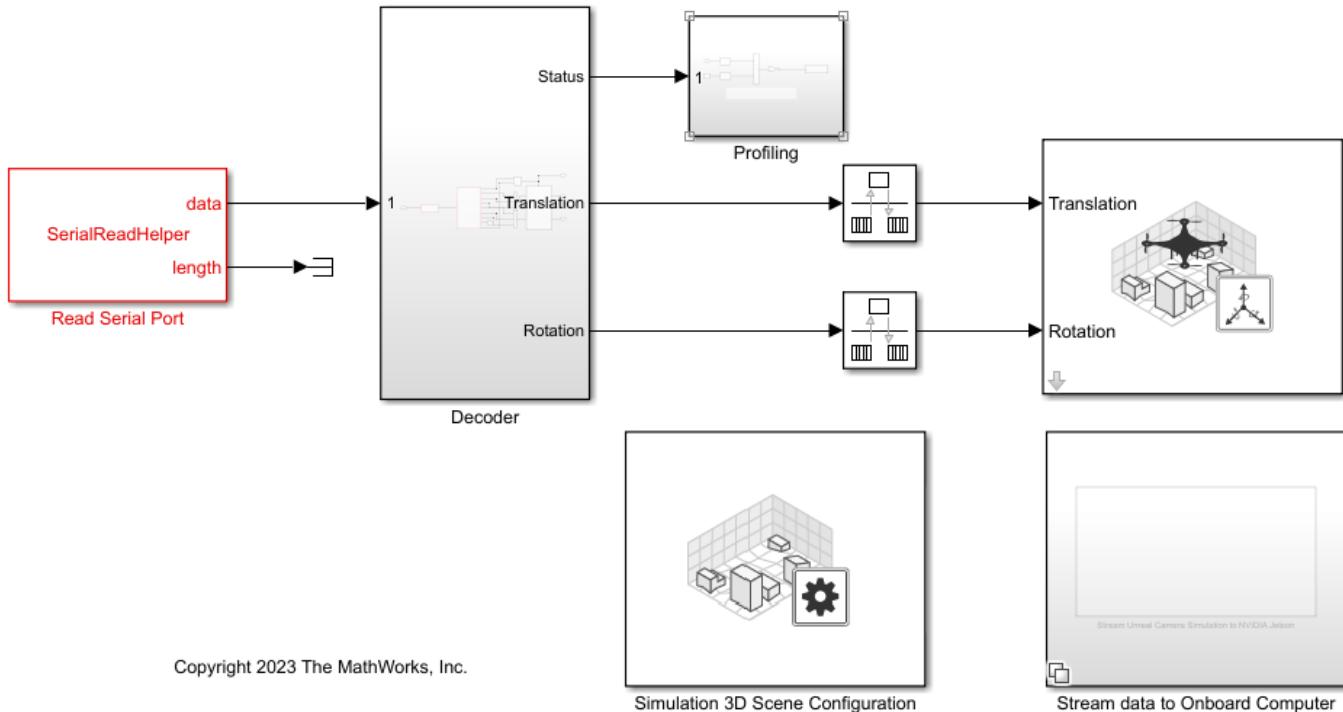
```
./QGroundControl.AppImage
```

Launch Second Session of MATLAB and Open the Flight Visualization Model

1. Launch second instance of the same MATLAB version.
2. Open the px4demo_HITLSimulinkPlantSpeedgoat project.
3. In the **Project Shortcuts** tab, click **3D Visualization** to launch the Flight Visualization model named *Unreal_3DVisualization*.



PX4 HITL Flight visualization

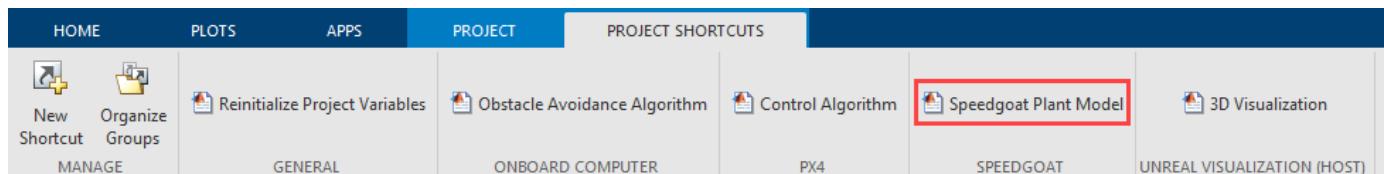


4. Open Read Serial Port helper block and select the serial port connecting Speedgoat with host PC.

5. On the **Simulation** tab, click **Run** to simulate the model. Once the model start running, you will see the Unreal® simulation environment getting launched.

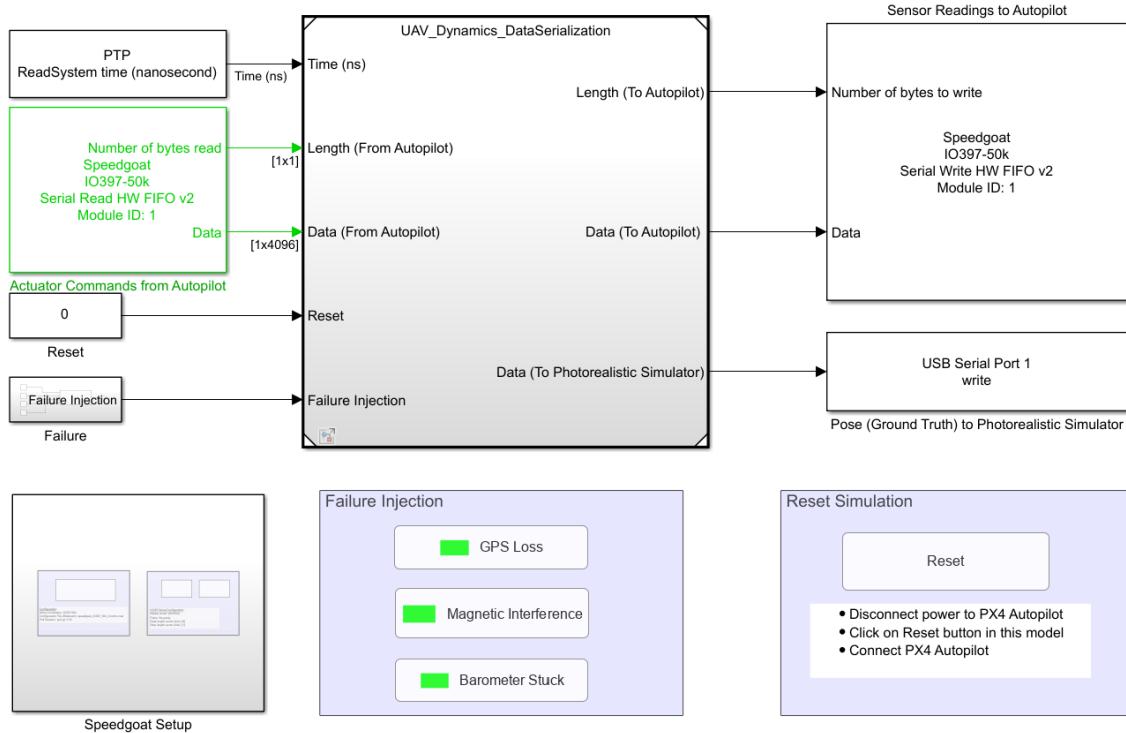
Deploy Plant Model to Speedgoat

1. In the **Project Shortcuts** tab, click **Speedgoat Plant Model** to launch the Simulink plant model (*UAV_Dynamics_Speedgoat*).



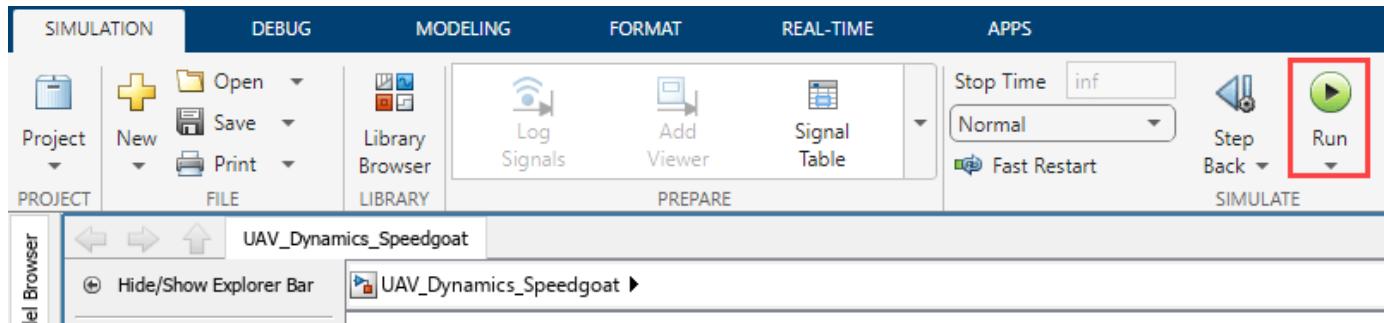
Quadcopter Plant Model for PX4 Hardware-in-the-Loop Simulation

Deployable to Speedgoat



2. In the **Real-Time** tab on the Simulink toolbar, select the Speedgoat kit under **Target Platform**.

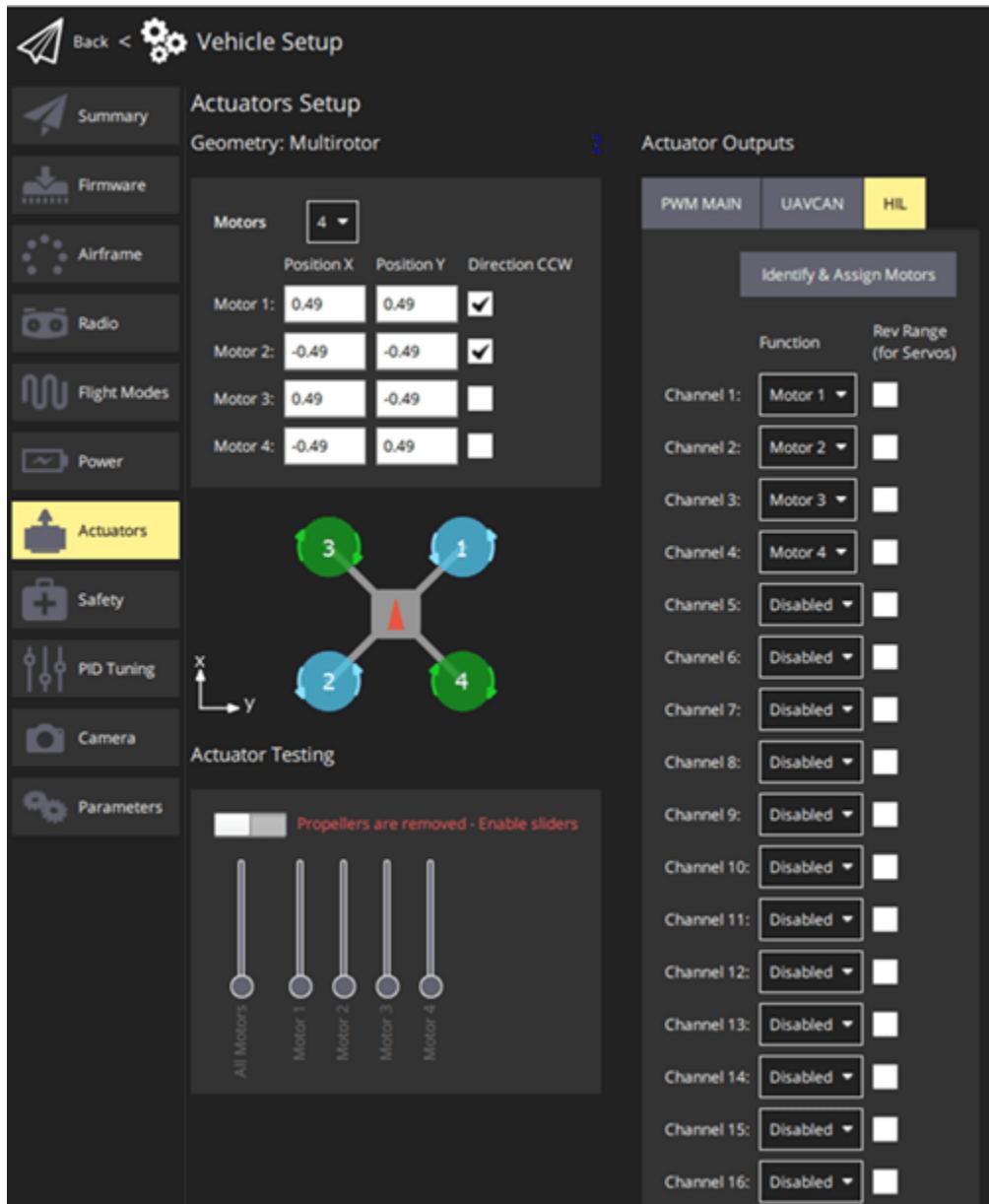
3. In the Simulink toolbar, on the **Simulation** tab, click **Run** to simulate the model.



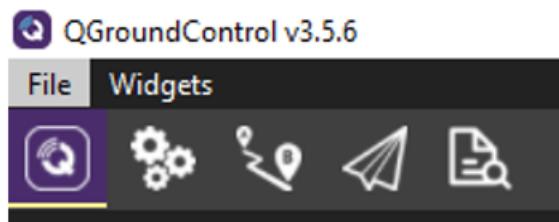
4. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to deploy the Plant model.

Upload Mission from QGroundControl and Fly the Autopilot

1. In the QGC, configure the actuators. For more information, see “Configure and Assign Actuators in QGC”.



2. Navigate to the *Plan View*.



3. Create a mission. For information on creating a mission, see Plan View.

After you create a mission, it will be visible in QGC.

4. Click Upload button in the QGC interface to upload the mission from QGroundControl.
5. Navigate to *Fly View* to view the uploaded mission.
6. Start the Mission in QGC. The PX4 Autopilot should follow the mission path.

Other Things to Try

- Try testing the controller in adverse conditions such as GPS failure and so on.

PX4 Stock Autopilot in HITL Simulation with UAV Dynamics modeled in Simulink

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to design UAV Dynamics in Simulink and verify them by using the PX4 Stock Autopilot in HITL mode.

You can design the UAV Dynamics in Simulink, simulate sensor values, and use these simulations to communicate with the Autopilot in HITL mode instead of relying on third-party simulators.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Configure and set up PX4 Autopilot in HITL mode. For more information, see “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.

Required Third-Party Software This example requires this third-party software:

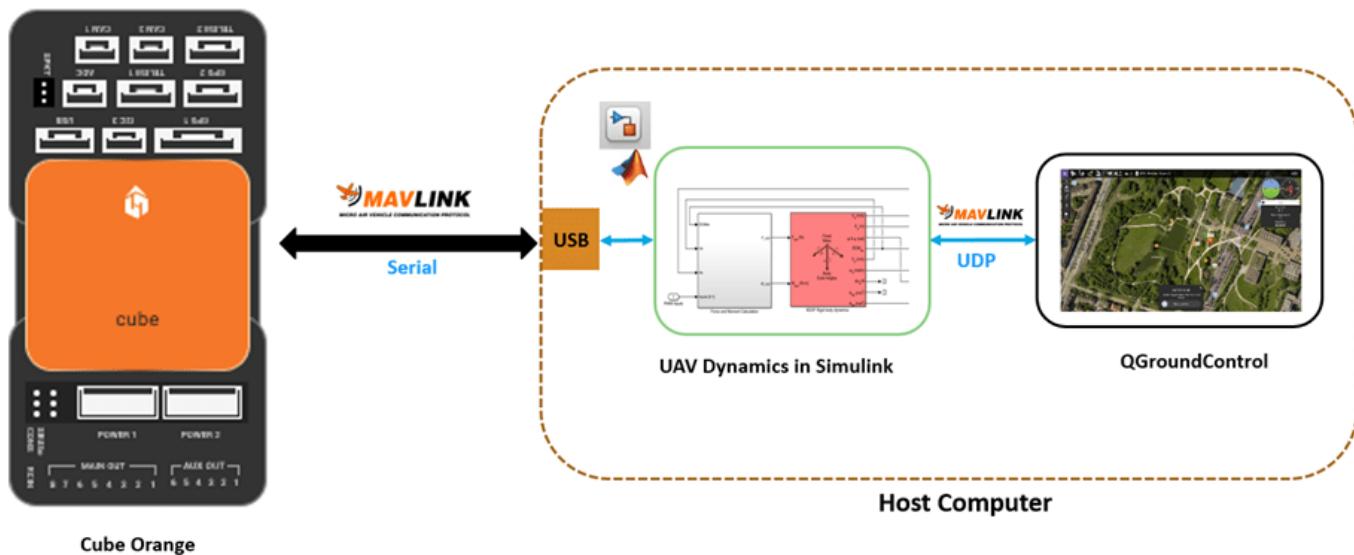
- QGroundControl (QGC)

Required Hardware To run this example, you will need the following hardware:

- PX4 Autopilot flight controller
- Micro USB (Universal Serial Bus) type-B cable
- Micro-SD card

Make Hardware Connections and setup the PX4 Autopilot in HITL Mode

The diagram below illustrates the HITL setup and the physical communication between various modules.



1. Connect your PX4 Autopilot board to the host computer using the USB cable.

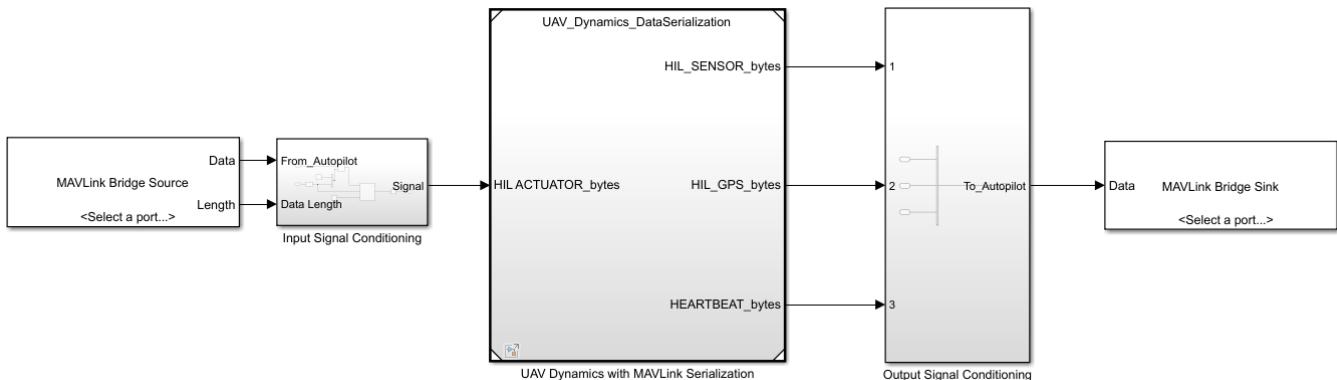
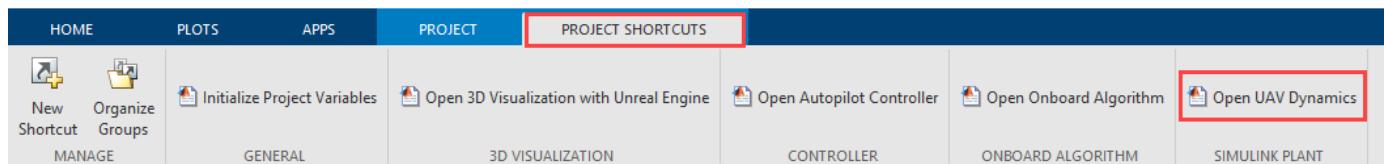
2. Ensure that you have configured the PX4 Autopilot board in HITL mode as documented in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”.

Open MATLAB Project and Controller and UAV Dynamics Model

The support package includes an example project having the PX4 controller and the UAV to follow the mission set in the QGroundControl (QGC).

1. Open MATLAB.
2. Open the example MATLAB project by executing this command at the MATLAB command prompt:

```
openExample('px4/PX4StockAutopilotHITLUAVDynamicsExample')
```
3. In the **Project Shortcuts** tab, click **Open UAV Dynamics** to launch the Simulink UAV Dynamics model named *UAV_Dynamics_Autopilot_Communication*.



4. Ensure that the COM ports are set in the MAVLink Bridge Source and MAVLink Bridge Sink blocks. To set the COM port:

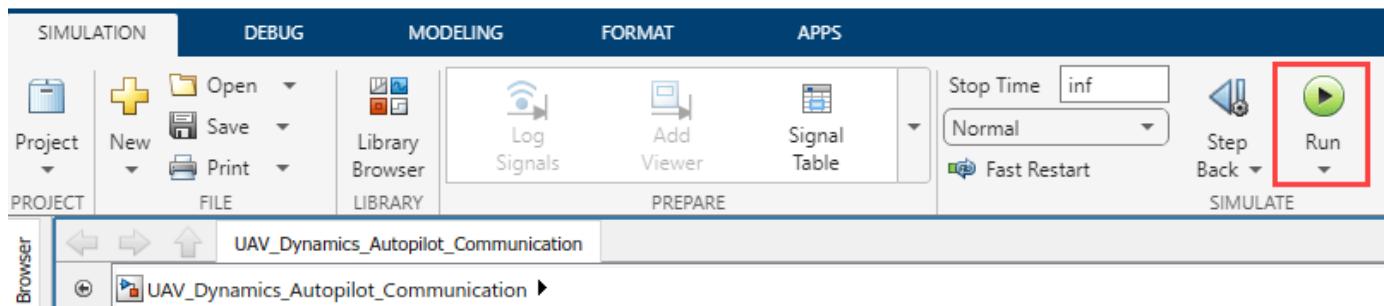
- Double-click the block to open the Block Parameters dialog box and then specify the value for the **Serial Port (Specify manually)** parameter.

Upload Firmware to PX4 Autopilot

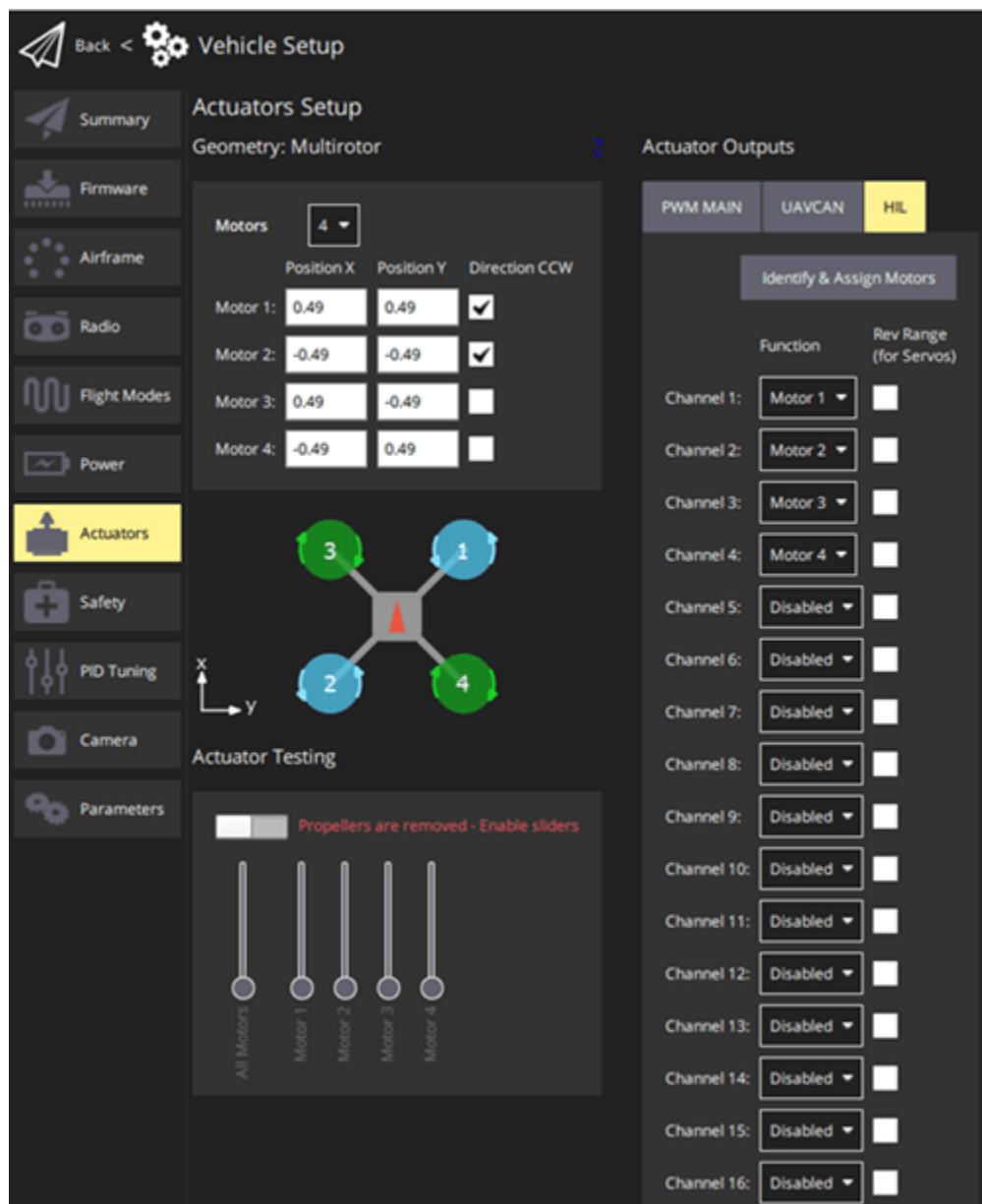
Use QGC to upload the latest stable firmware. For steps to upload the firmware, see Loading Firmware.

Run the UAV Dynamics Model, Upload Mission from QGroundControl, and Fly the UAV

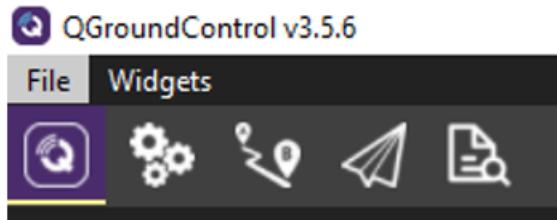
1. In the Simulink toolbar of the Plant model (*UAV_Dynamics_Autopilot_Communication*), on the **Simulation** tab, click **Run** to simulate the model.



2. Configure the actuators in QGC. For more information, see “Configure and Assign Actuators in QGC”.



3. Navigate to the *Plan View*.



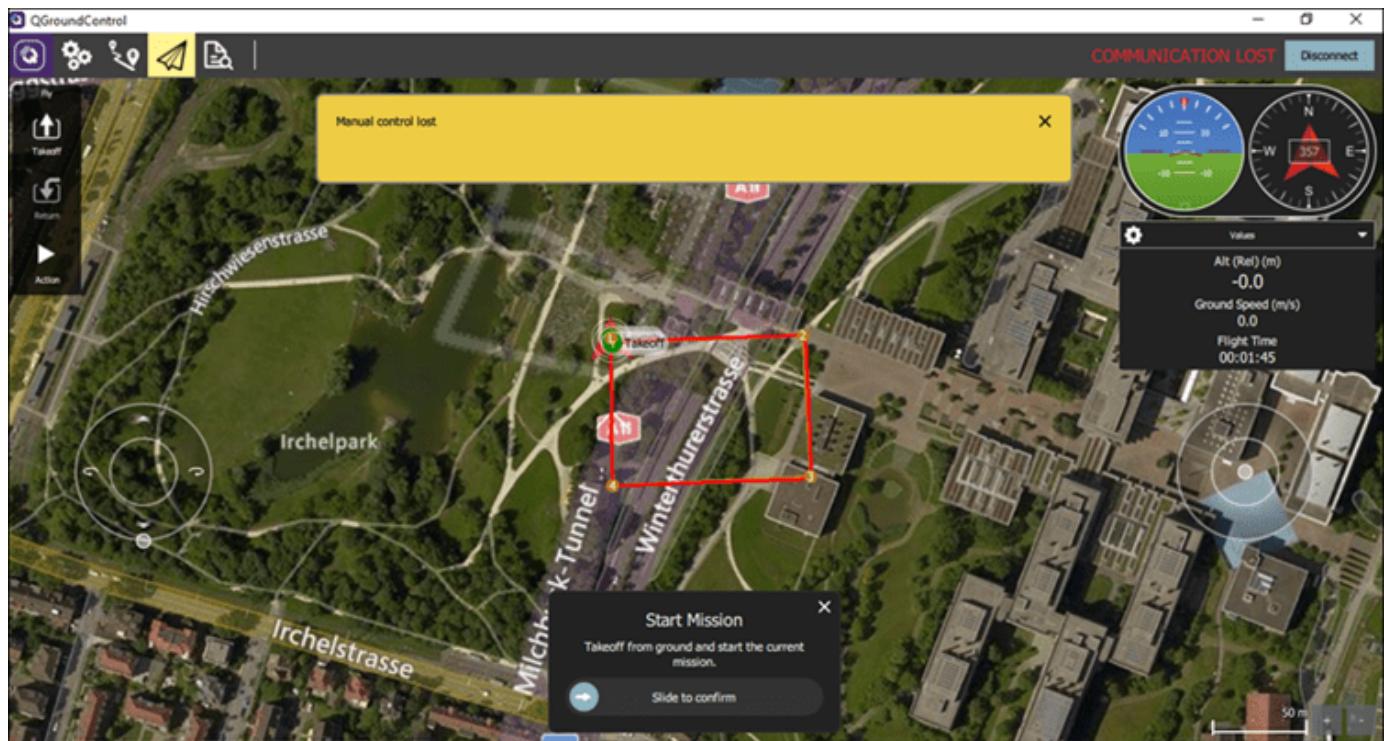
4. Create a mission. For information on creating a mission, see Plan View.

After you create a mission, it is visible in QGC.

5. Click the Upload button in the QGC interface to upload the mission from QGroundControl.

6. Navigate to *Fly View* to view the uploaded mission.

7. Start the Mission in QGC. The UAV should follow the mission path.



PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Tilt-Rotor Plant in Simulink

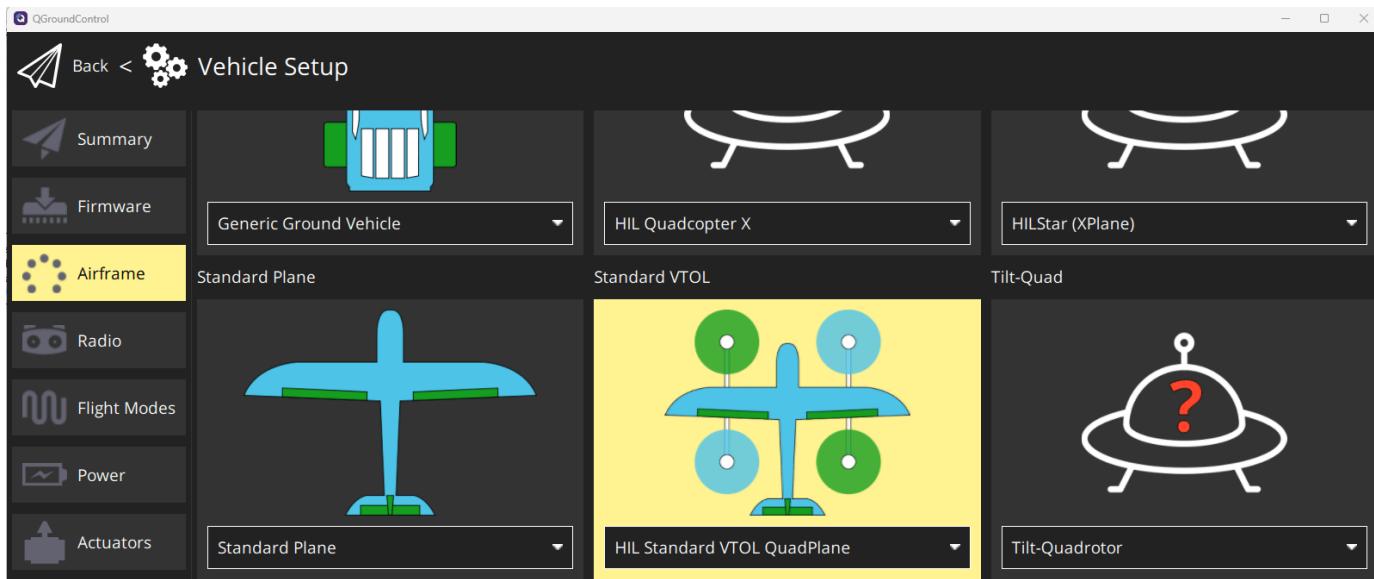
This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to verify a UAV controller design by using Hardware-in-the-Loop (HITL) simulation on a PX4 Autopilot autopilot and simulating the VTOL tilt rotor UAV Dynamics in Simulink®.

You can use the Simulink model in this example as a reference to design a flight controller algorithm for take-off, hover, waypoint following, forward transition, back transition, and landing for a VTOL tilt-rotor UAV plant model. The plant model in this example simulates the aerodynamics, propulsion, ground contact, and sensors of a VTOL tilt-rotor UAV.

Prerequisites

You must complete these before proceeding with this example:

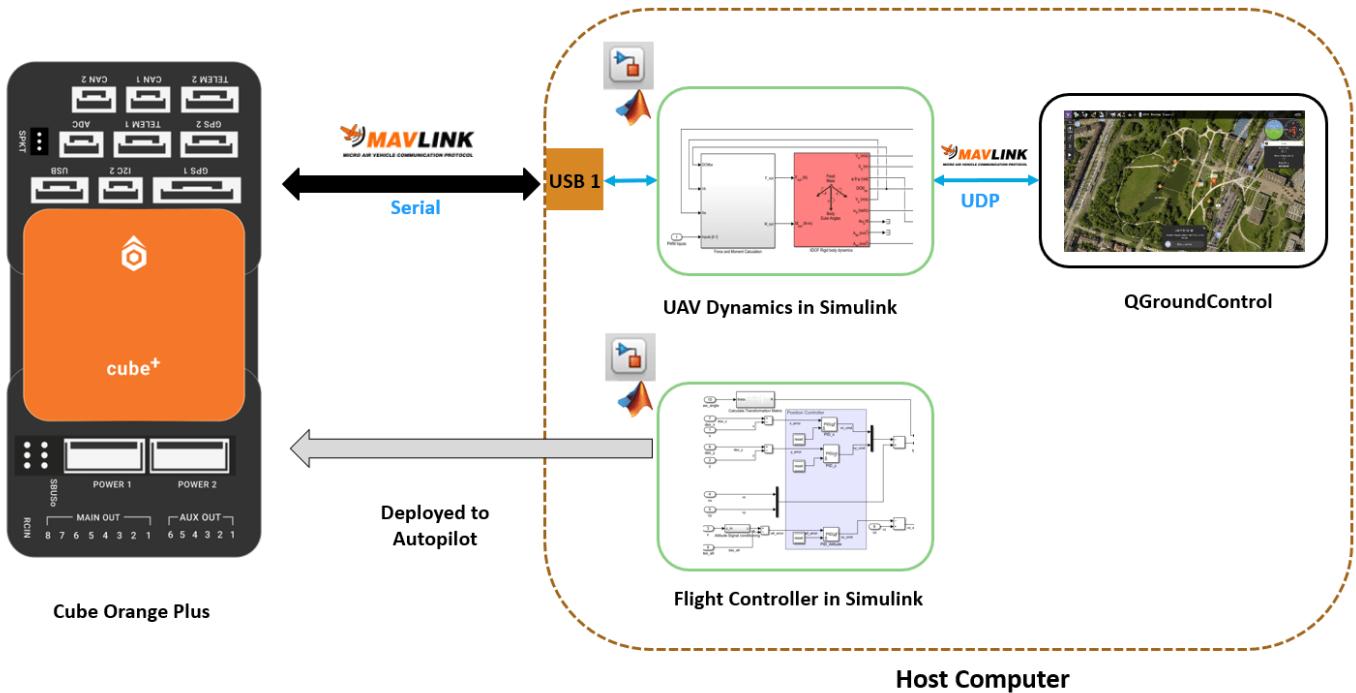
- 1 If you are new to Simulink, watch the Simulink Quick Start video.
- 2 Set up the physical connection for HITL simulation between the PX4 Autopilot and host computer running Simulink as shown in “PX4 Hardware-in-the-Loop System Architecture”.
- 3 Configure and set up the PX4 Autopilot in HITL mode as shown in “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”. When selecting an airframe, choose the **HIL Standard VTOL QuadPlane**.
- 4 Set up the PX4 firmware as described in “Set Up PX4 Firmware for Hardware-in-the-Loop (HITL) Simulation”. At the **Select a PX4 Autopilot and Build Target** step, set any PX4 Autopilot as the **PX4 Autopilot board**. Select the VTOL specific px4board build target from the dropdown list. For example if you are using Cube Orange + as the Autopilot, choose `cubepilot_cubearangeplus_VTOL` as the px4board build target. This example uses `Cube Orange +`.



Specify these parameters for the airframe in QGroundControl to avoid triggering auto-landing when the VTOL is hovering:

- **LNDMC_XY_VEL_MAX** — 0 m/s
- **LNDMC_Z_VEL_MAX** — 0 m/s
- **LNDMC_ROT_MAX** — 0 rad/s

This diagram illustrates the HITL configuration and the physical communication between various modules.



Required Third-Party Software

- QGroundControl (QGC)

Required Hardware

- PX4 Autopilot flight controller
- Micro-USB (Universal Serial Bus) type-B cable
- Micro-SD card

Getting Started

To open the example livescript and directory which contains the Simulink project file, first run `openExample('uav/PX4HITLSimulationWithVT0LTiltRotorSimulinkExample')` in the command window.

You must open the `VT0LRefApp.prj` project file to access the Simulink model, supporting files, and project shortcuts that this example uses.

```
% Open the Simulink project after running openExample('uav/PX4HITLSimulationWithVT0LTiltRotorSimulinkExample');
proj = openProject("VT0LApp");
```

Use the highlighted project shortcuts to set up the HITL plant and controller models, open the HITL plant and controller models, and select the type of mission that you want to simulate.

VTOLREFAPP: 1. SETUP

-  Getting Started
-  Open Model

VTOLREFAPP: 2. HOVER

-  1. Set Hover Configuration
-  2. Manual Mode
-  3. Guidance Mode

VTOLREFAPP: 3. FIXED WING

-  1. Set Fixed Wing Configuration
-  2. Manual Mode
-  Guidance Mode

VTOLREFAPP: 4. TRANSITION

-  1. Set up Transition Sensitivity Experiment
-  2. Set up Full Transition Mission

VTOLREFAPP: 5. PHOTOREALISTIC SIMULATION

-  Set up City Mission

VTOLREFAPP: 6. HITL SETUP

-  Initialize Controller Parameters
-  Initialize Plant Parameters

VTOLREFAPP: 7. HITL MODELS

-  Open HITL Controller Model
-  Open HITL Plant Model

VTOLREFAPP: 8. HITL MISSION

-  HITL Full Transition Mission
-  HITL Hover Mission
-  HITL Unreal City Mission

VTOLREFAPP: 9. ALTERNATIVE UAV CONFIGURATION MISSION

-  Set up Full Transition Mission

To initialize the parameters and configurations of the HITL plant model, click the **Initialize Plant Parameters** shortcut or run the `setupHITLConfiguration` helper function.

```
setupHITLConfiguration
```

To initialize the parameters and configurations of the HITL controller model, click the **Initialize Controller Parameters** shortcut or run the `setupHITLController` helper function.

```
setupHITLController
```

Set Up Mission with Transition

This example contain a sample mission in which the UAV takes off, transitions to fixed wing, and performs guided flight that is defined in the `exampleHelperTransitionMissionData.m` file.

Load the sample mission by clicking the **HITL Full Transition Mission** shortcut or by using the `setupHITLTransitionGuidanceMission` helper function.

```
setupHITLTransitionGuidanceMission
```

Alternatively, you can load a predefined hover guidance mission by clicking the **HITL Hover Mission** shortcut or by using the `setupHITLHoverGuidanceMission` helper function.

To create your own mission, you can customize the mission defined in the `exampleHelperTransitionMissionData.m` file. The following table summarize different mission modes and the corresponding flight modes. Note that the modes are based on the format that the Path Manager uses.

Mission Mode	Action	Supported Flight Mode
1	Takeoff	Hover
2	Waypoint	Hover, Fixed Wing
3	Orbit	Hover, Fixed Wing
4	Land	Hover
5 (Not used in this example)	RTL (Not used in this example)	N/A (Not used in this example)
6	Transition	Hover, Fixed Wing

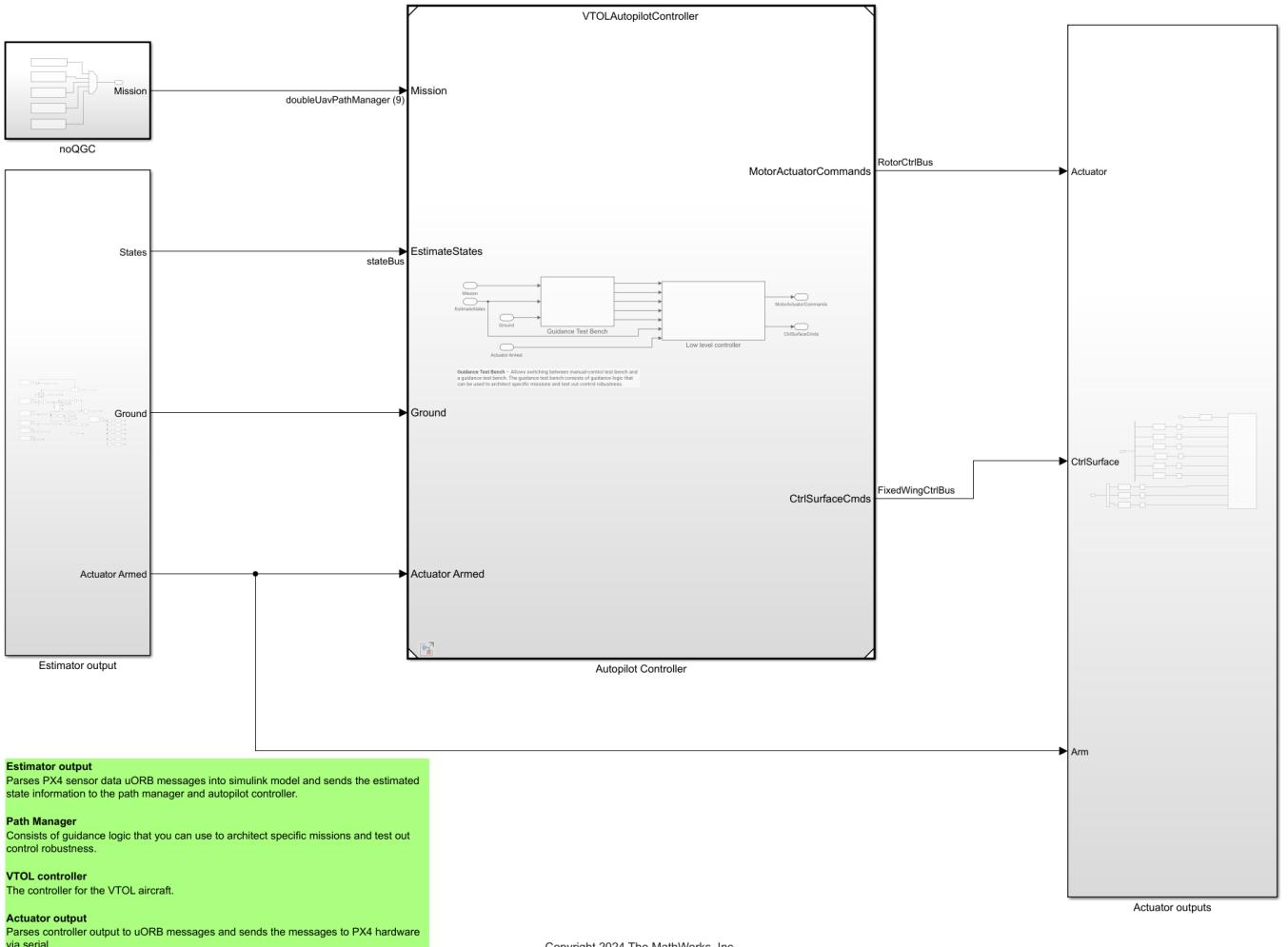
Deploy HITL Controller Model to PX4 Autopilot Target

The `HITL_controller_top.slx` file contains the HITL controller model. The model consists of the following subsystems:

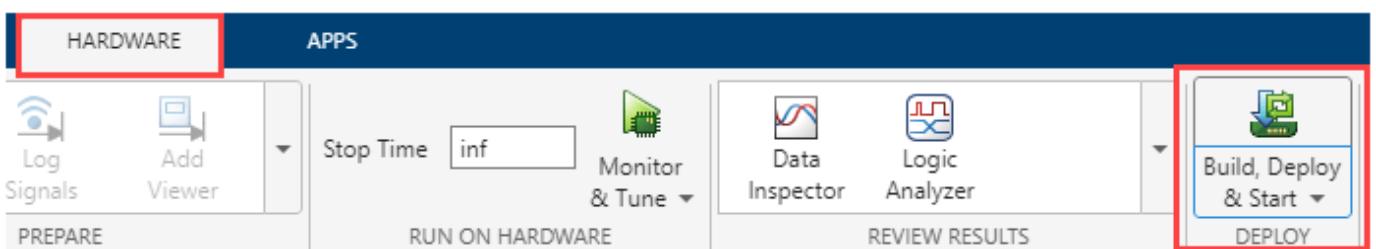
- `VTOLAutopilotController` — Reference flight controller design for a VTOL tilt-rotor UAV model.
- `Actuator outputs` — Parses the `VTOLAutopilotController` subsystem outputs into uORB messages.
- `Estimator` — Obtains VTOL UAV states from PX4 estimator for controller feedback.

Open the `HITL_controller_top.slx` model using the **Open HITL Controller Model** shortcut, or use the following code block.

```
controller_mdl = "HITL_Controller_top.slx";
open_system(controller_mdl)
```



To generate the code of the flight controller model and deploy it to the PX4 Autopilot board, select **Build, Deploy & Start** on the **Hardware** tab of the Simulink Toolstrip.



After completing deployment, the model launches QGroundControl.

Note :

- If you are using Ubuntu, QGroundControl might not launch automatically. To launch QGroundControl, open the terminal and go to the folder where QGroundControl is downloaded and , and then enter this command: `./QGroundControl.AppImage`

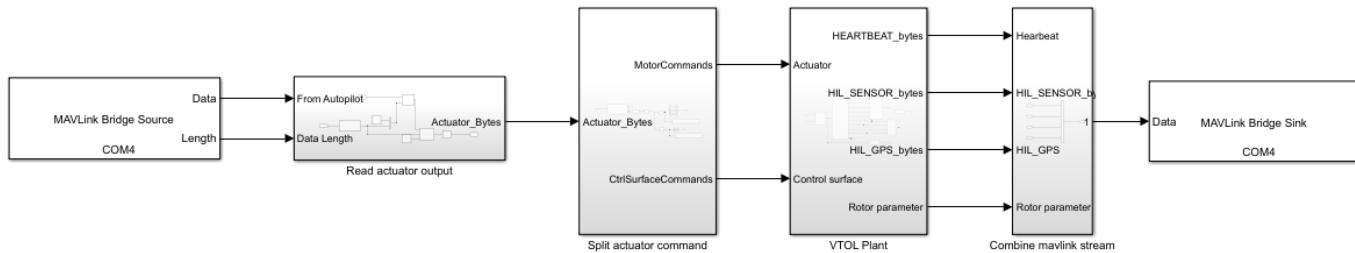
- 2 If you have changed the hardware, or are not using the pre-configured Simulink model, then you must configure your Simulink model as explained in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”. Run the following code block to launch the Configuration Parameters dialog in a new window.

```
configObj = getActiveConfigSet('VTOLAutopilotController');
deploymentConfigSet = getRefConfigSet(configObj);
openDialog(deploymentConfigSet);
```

Run HITL Simulation

The HITL_Plant_top.slx file contains the HITL plant model. The model consists of the following blocks:

- MAVLink Bridge Source block — Reads the MAVLink messages from the PX4 Autopilot board and sends it to VTOL Plant subsystem
- VTOL Plant subsystem — Plant model for the VTOL tilt rotor UAV, which simulates the aerodynamics, propulsion, ground contact, and sensors. The subsystem outputs sensor, rotor parameter, and GPS data.
- MAVLink Bridge Sink block — Parses VTOL Plant subsystem as MAVLink messages and sends the messages back to PX4 Autopilot by using serial connection

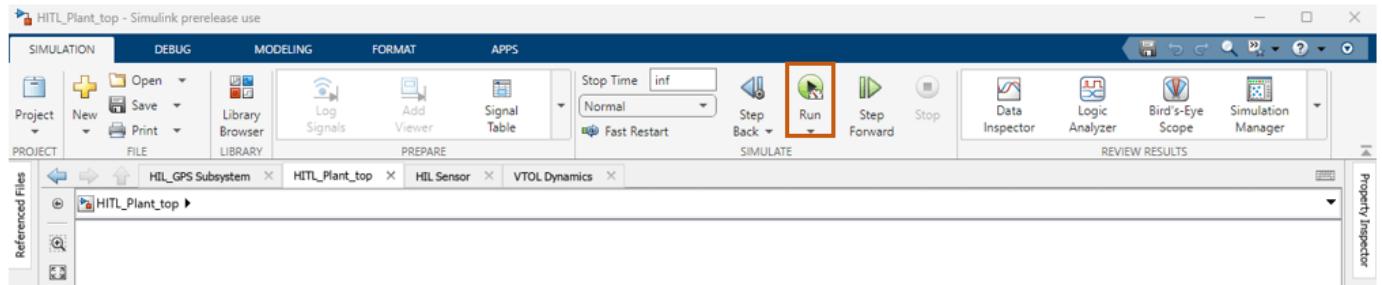


Open the HITL_Plant_top.slx model by using the **Open HITL Plant Model** shortcut or by running the following code block.

```
plant_mdl = "HITL_Plant_top.slx";
open_system(plant_mdl)
```

Configure the **Serial Port** parameter of the MAVLink Bridge Source and the MAVLink Bridge Sinks blocks to match the COM port that you use to connect the PX4 autopilot.

To start the HITL simulation, run the HITL_Plant_top.slx model

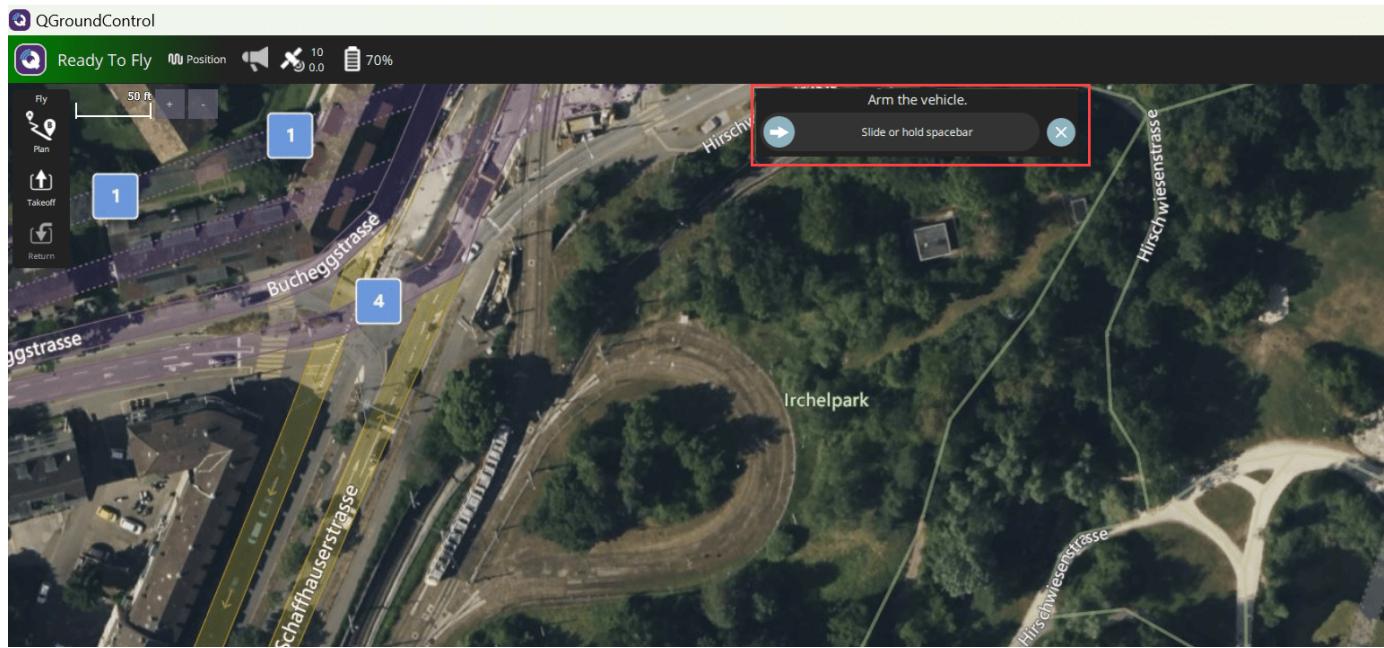


Then, Open QGroundControl and configure the actuators as shown. For more information, see “Configure and Assign Actuators in QGC”.



Next, select **Fly View** in QGroundControl. Arm the VTOL UAV to start the mission by selecting **Arm**.

1 Blocks



After the VTOL UAV is armed, the UAV takes off and transitions into fixed-wing mode to complete the mission.



Discard changes to reset the example.

```
discardChanges(myDictionaryObj);
close_system(controller_mdl,0);
```

Before resuming to other examples in this example series, you must close the `VTOLRefApp.prj` Simulink project by running this command in the Command Window:

```
close(prj)
```

See Also

Related Examples

- “Design and Tune Controller for VTOL UAV”

More About

- “Configure and Assign Actuators in QGC”
- “Setting Up PX4 Autopilot in Hardware-in-the-Loop (HITL) Mode from QGroundControl”
- “Troubleshooting Deploy to Hardware Issues”

Visualize PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Over Urban Environment

This example shows how to visualize VTOL UAV Hardware-in-the-Loop (HITL) simulation within an urban environment by using Unreal Engine®.

Prerequisites

- 1 **It is recommended to run two instances of MATLAB for this example.** Use the first instance of MATLAB® deploy the controller model to Pixhawk® and run the HITL plant model. Use the second instance of MATLAB to set up and run the Unreal Engine visualization.
- 2 Set up and run the “PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Tilt-Rotor Plant in Simulink” example. This example will help you to understand the required hardware setup, as well as the HITL controller and plant models.
- 3 Set up and run the “Visualize VTOL UAV Mission Over Urban Environment” example. This example shows how use to custom actor workflow in UAV Toolbox to visualize aircraft performance in a photorealistic environment.
- 4 Download and install UAV Toolbox Interface for Unreal Engine Projects. For more information about installing and setting up the UAV Toolbox Interface, see “Install Support Package for Customizing Scenes”.
- 5 This example uses a 3D Game Asset Urban Air Mobility Vehicle dataset, which contains a fbx file of a VTOL UAV and is approximately 6 MB in size. Download the VTOLAsset ZIP file from the MathWorks® website, then unzip the file.

```
zipFile = matlab.internal.examples.downloadSupportFile("uav","data/VTOLAsset.zip");
filepath = fileparts(zipFile);
dataFolder = fullfile(filepath,'VTOLAsset');
unzip(zipFile,dataFolder)
```

Getting Started in First MATLAB Instance

To open the example livescript and directory which contains the Simulink project file, first run `openExample('uav/VisualizeVTOLHITLFlightMissionOverUrbanEnvironmentExample')` in the command window using the first MATLAB instance.

You must open the VTOLRefApp.prj project file to access the Simulink model, supporting files, and project shortcuts that this example uses.

```
% Open the Simulink project after running openExample('uav/VisualizeVTOLHITLFlightMissionOverUrbanEnvironmentExample')
prj = openProject('VTOLApp');
```

Use the highlighted project shortcuts to set up the HITL plant and controller models, open the HITL plant and controller models, and select the type of mission that you want to simulate.

VTOLREFAPP: 1. SETUP

-  Getting Started
-  Open Model

VTOLREFAPP: 2. HOVER

-  1. Set Hover Configuration
-  2. Manual Mode
-  3. Guidance Mode

VTOLREFAPP: 3. FIXED WING

-  1. Set Fixed Wing Configuration
-  2. Manual Mode
-  Guidance Mode

VTOLREFAPP: 4. TRANSITION

-  1. Set up Transition Sensitivity Experiment
-  2. Set up Full Transition Mission

VTOLREFAPP: 5. PHOTOREALISTIC SIMULATION

-  Set up City Mission

VTOLREFAPP: 6. HITL SETUP

-  Initialize Controller Parameters
-  Initialize Plant Parameters

VTOLREFAPP: 7. HITL MODELS

-  Open HITL Controller Model
-  Open HITL Plant Model

VTOLREFAPP: 8. HITL MISSION

-  HITL Full Transition Mission
-  HITL Hover Mission
-  HITL Unreal City Mission

VTOLREFAPP: 9. ALTERNATIVE UAV CONFIGURATION MISSION

-  Set up Full Transition Mission

To initialize the parameters and configurations of the HITL plant model, click the **Initialize Plant Parameters** shortcut or run the `setupHITLConfiguration` helper function.

```
setupHITLConfiguration;
```

To initialize the parameters and configurations of the HITL controller model, click the **Initialize Controller Parameters** shortcut or run the `setupHITLController` helper function.

```
setupHITLController;
```

Set Up City Mission in First MATLAB Instance

This example contains a sample mission in which the UAV takes off, performs guided flight, and lands in hover mode as defined in the `setupHITLCityMission.m` file.

Load the sample mission by clicking the **HITL Unreal City Mission** shortcut or by using the `setupHITLCityMission` helper function.

```
setupHITLCityMission;
```

Initialized city mission.

To create your own mission, you can customize the mission defined in the `setupHITLCityMission.m` file. The following table summarizes different mission modes and the corresponding flight modes. Note that the modes are based on the format that the Path Manager uses.

Mission Mode	Action	Supported Flight Mode
1	Takeoff	Hover
2	Waypoint	Hover, Fixed Wing
3	Orbit	Hover, Fixed Wing
4	Land	Hover
5 (Not used in this example)	RTL (Not used in this example)	N/A (Not used in this example)
6	Transition	Hover, Fixed Wing

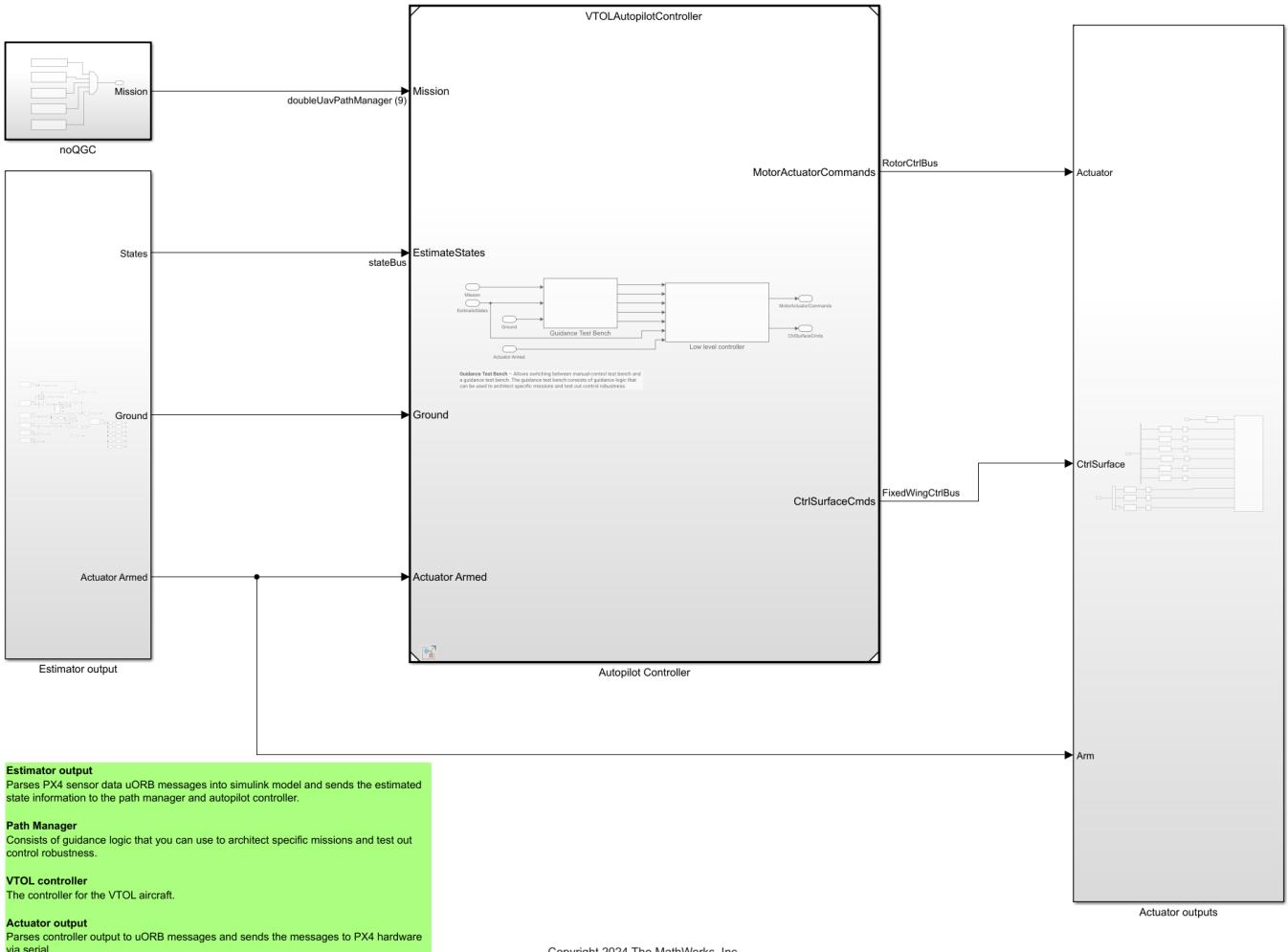
Deploy HITL Controller Model to Pixhawk Target in First MATLAB Instance

The `HITL_controller_top.slx` file contains the HITL controller model. The model consists of the following subsystems:

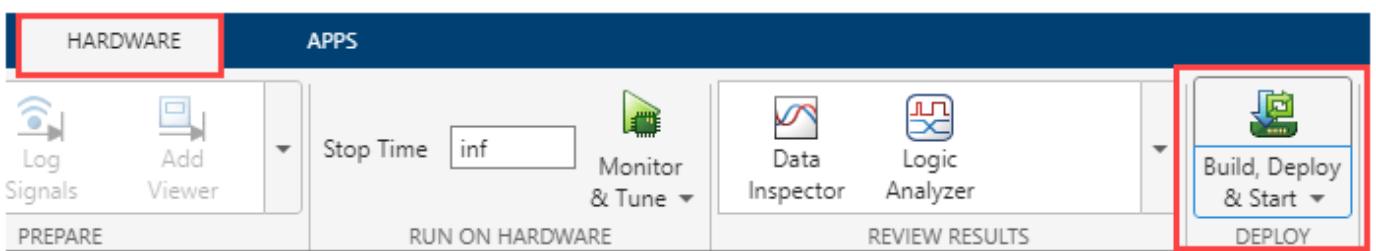
- `VTOLAutopilotController` — Reference flight controller design for a VTOL tilt-rotor UAV model.
- `Actuator outputs` — Parses the `VTOLAutopilotController` subsystem outputs into uORB messages.
- `Estimator` — Obtains VTOL UAV states from PX4 estimator for controller feedback.

Open the `HITL_controller_top.slx` model using the **Open HITL Controller Model** shortcut, or use the following code block.

```
controller_mdl = "HITL_Controller_top.slx";
open_system(controller_mdl)
```



To generate the code of the flight controller model and deploy it to the Pixhawk board, select **Build, Deploy & Start** on the **Hardware** tab of the Simulink Toolstrip.



After completing deployment, the model launches QGroundControl.

Note :

- If you are using Ubuntu, QGroundControl might not launch automatically. To launch QGroundControl, open the terminal and go to the folder where QGroundControl is downloaded and , and then enter this command: `./QGroundControl.AppImage`

- 2 If you have changed the hardware, or are not using the preconfigured Simulink model, then you must configure your Simulink model as explained in “Configure Simulink Model for Deployment in Hardware-in-the-Loop (HITL) Simulation”.

Configure Unreal Engine Visualization in Second MATLAB Instance

Open Example and Project File

Launch the second instance of MATLAB to set up and run the Unreal Engine visualization. Open the example live script by running `openExample('uav/VisualizeVTOLHITLFlightMissionOverUrbanEnvironmentExample')`. Then, open the project file to access the Simulink model and supporting files.

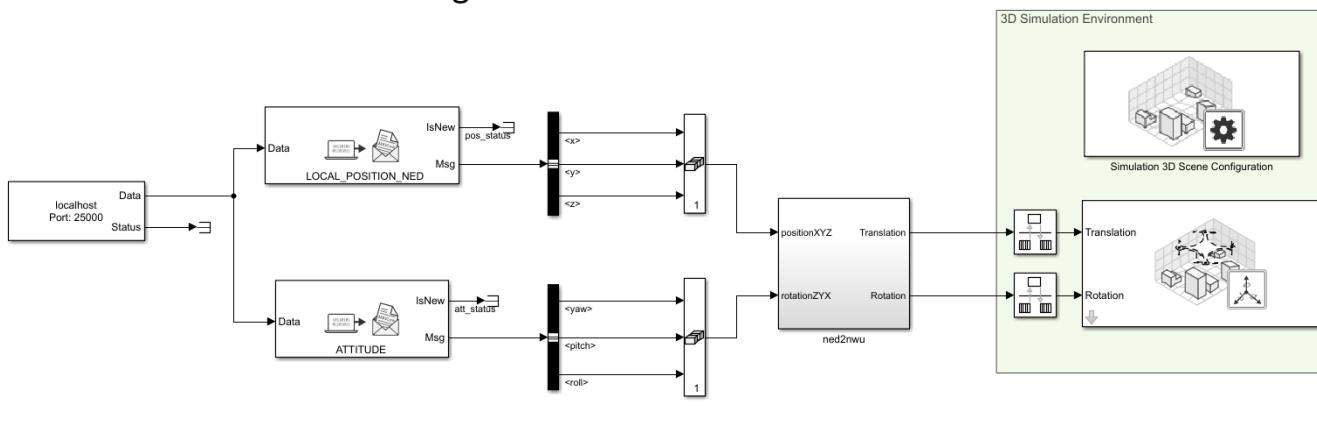
```
% Open the Simulink project after running openExample('uav/VisualizeVTOLHITLFlightMissionOverUrbanEnvironmentExample')
prj = openProject('VTOLApp');
```

Point Model to Unreal Engine Project

Open the `HITL_Unreal_Visualization.slx` Simulink model.

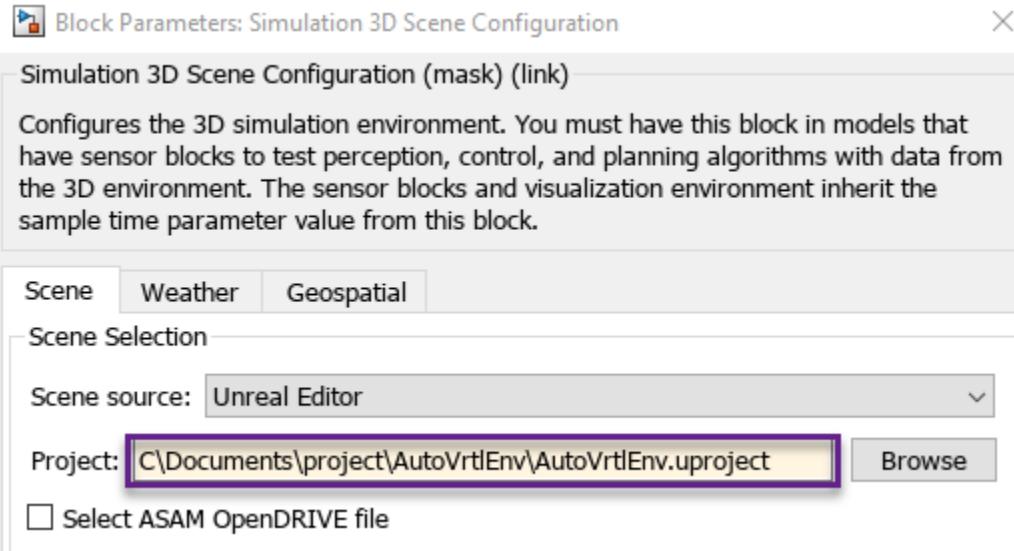
```
open_system("HITL_Unreal_Visualization")
```

PX4 VTOL HITL Flight visualization



Copyright 2024 The MathWorks, Inc.

Open the **Simulation 3D Scene Configuration** block mask and set the **Scene source** parameter to **Unreal Editor** and supply the path to the `AutoVrtlEnv` Unreal Engine project on your system.



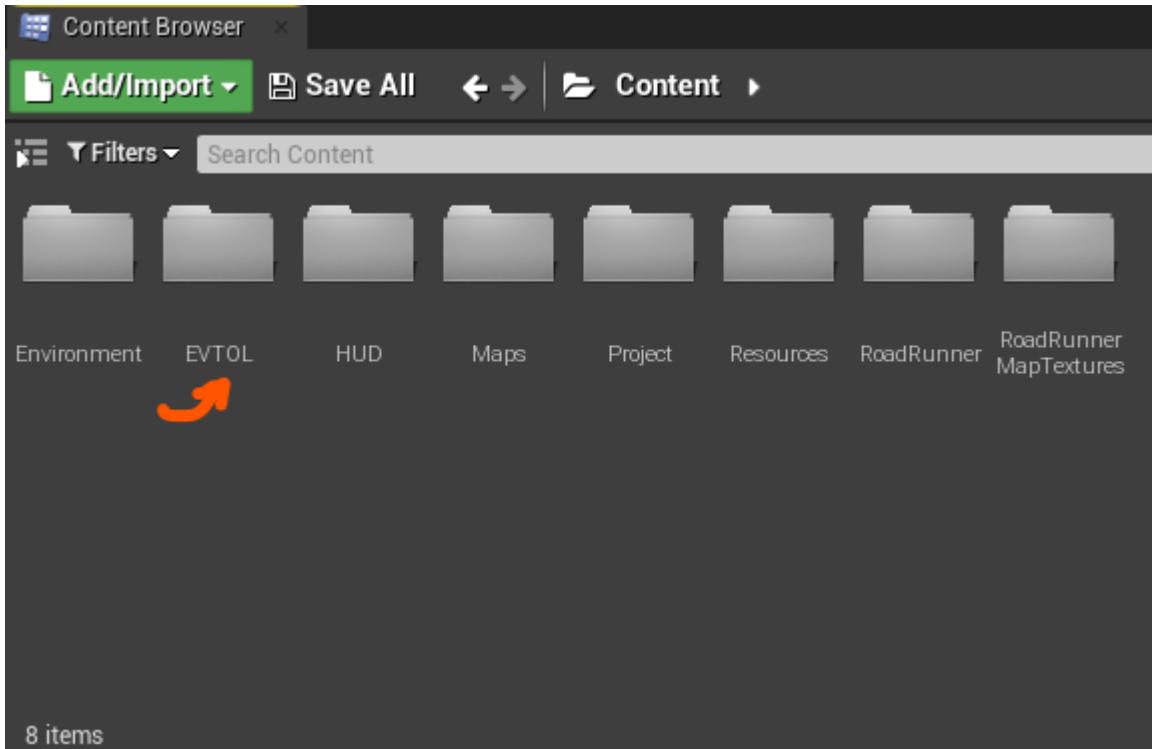
If you do not have the AutoVrtlEnv file, you must download the “Install Support Package for Customizing Scenes” support package to complete this step. The installation and setup process for the UAV Toolbox Interface for Unreal Engine Projects includes setting up the AutoVrtlEnv file.

Import VTOL UAV into Unreal Engine Scene

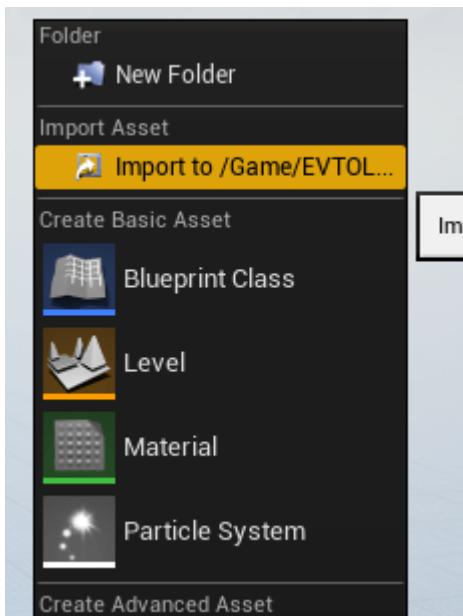
To launch Unreal Editor, open the Photorealistic subsystem. Then, open the **Simulation 3D Scene Configuration** block and click **Open Unreal Editor**



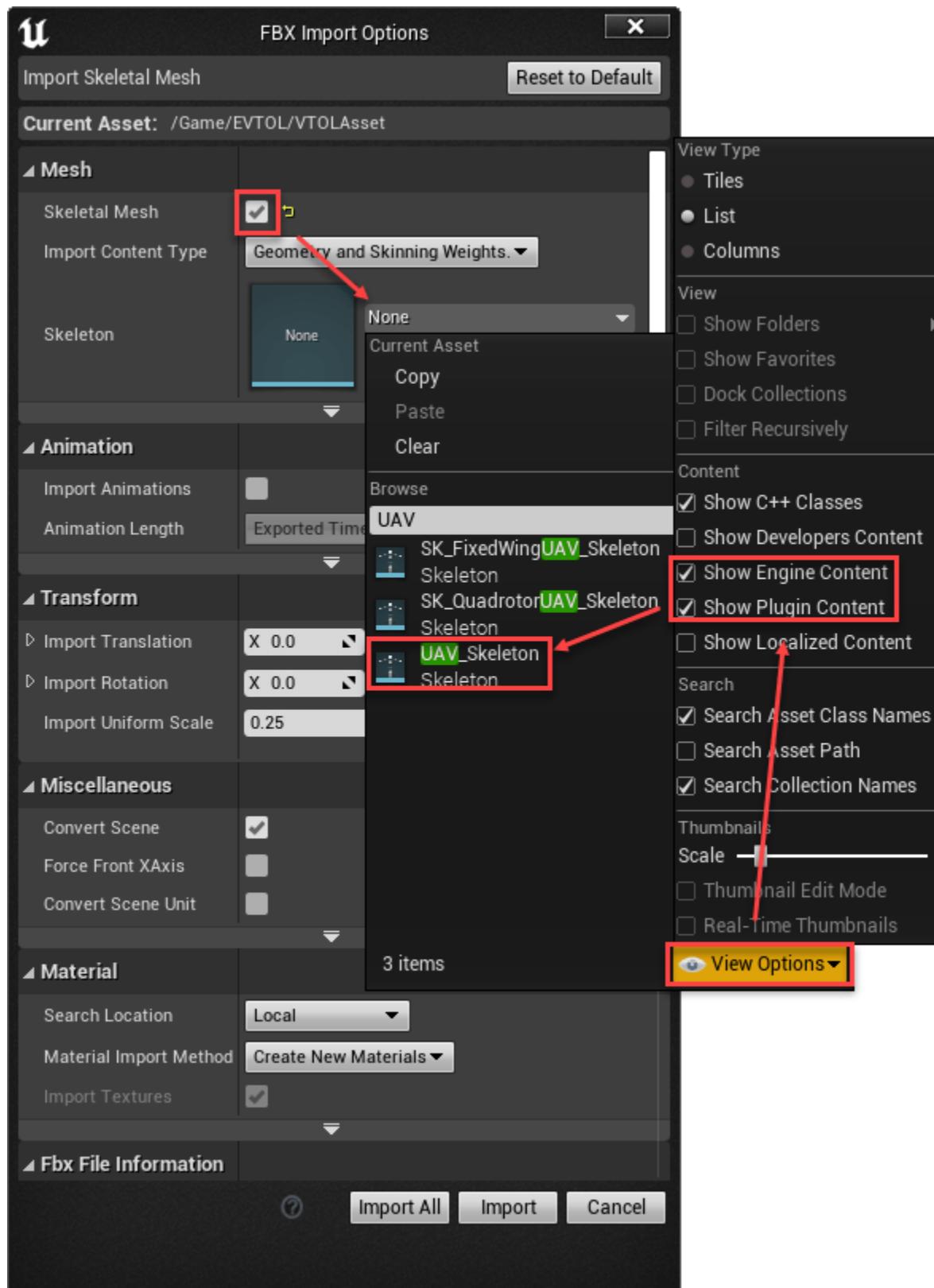
Click **Add/Import** in Unreal Editor and select **New Folder** from the context menu. Create a folder called EVTOL in the Content Browser pane, and then open the EVTOL folder.



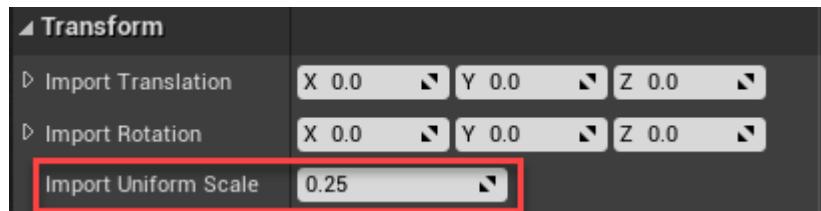
Click **Add/Import** again and then select **Import to Game/EVTOL**. Navigate to the folder that the VTOLAsset is saved in and select the VTOLAsset file to open the **FBX Import Options** dialog box.



To select a skeleton mesh, you must first open the **Skeleton** drop down list and click the **View Options** drop down. Then, select both **Show Engine Content** and **Show Plugin Content** to add the **UAV_Skeleton** mesh option to the **Skeleton** drop down list. Finally, select **UAV_Skeleton**.



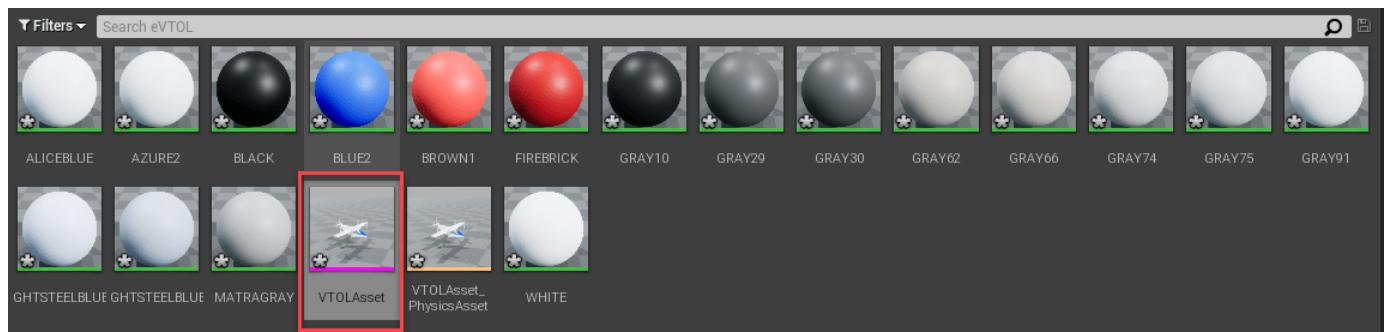
Set the **Import Uniform Scale** parameter to **0.25** in FBX Import Options to scale the VTOL asset to a smaller, more realistic size.



To finish the import of asset, click **Import** in Unreal Editor.

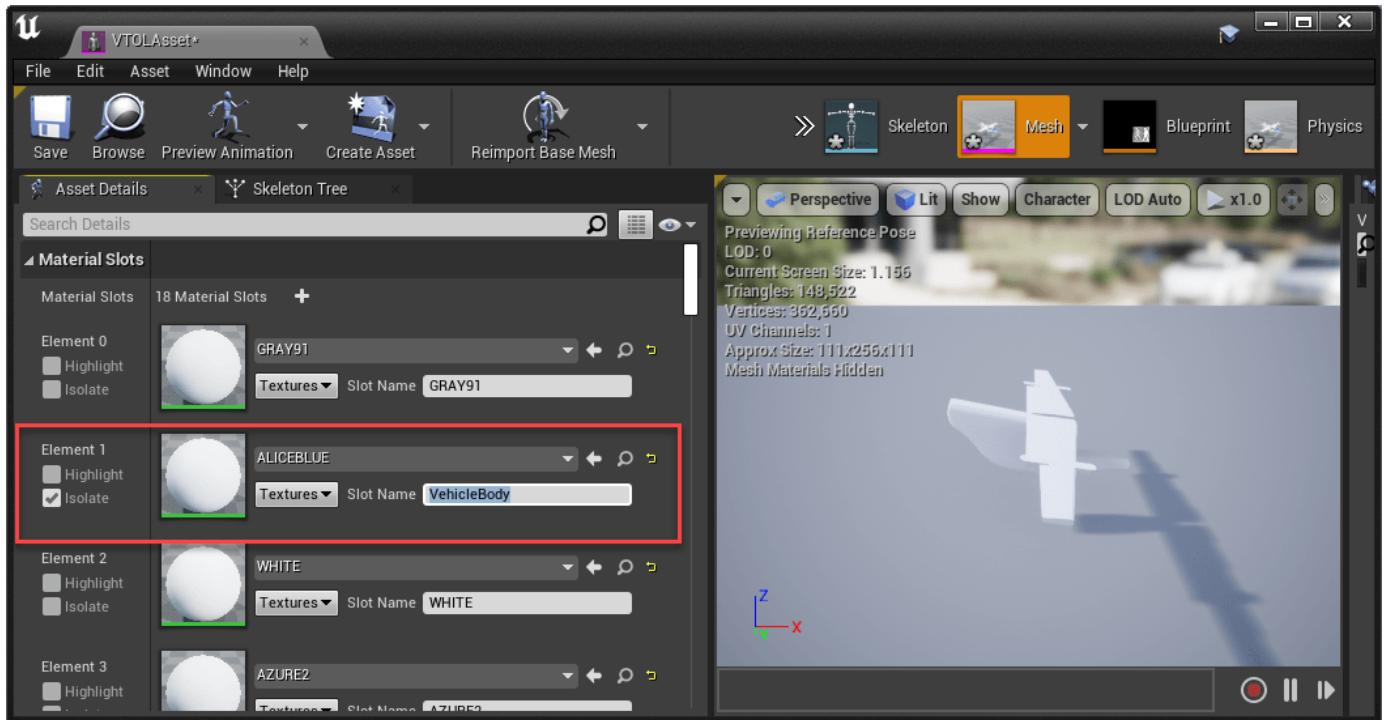
Customize VTOL UAV Asset

Double click **VTOLAsset** in the **Content Browser** pane to open the **Asset Details** window.

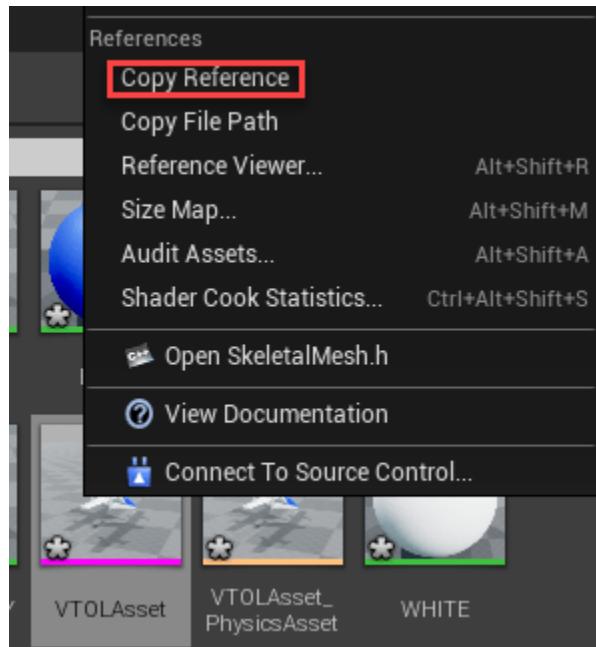


Select the element of the VTOL skeletal mesh that to be the main body of the vehicle and name it **VehicleBody**. For this VTOL asset, select **Element 1**, the fuselage, as the main body.

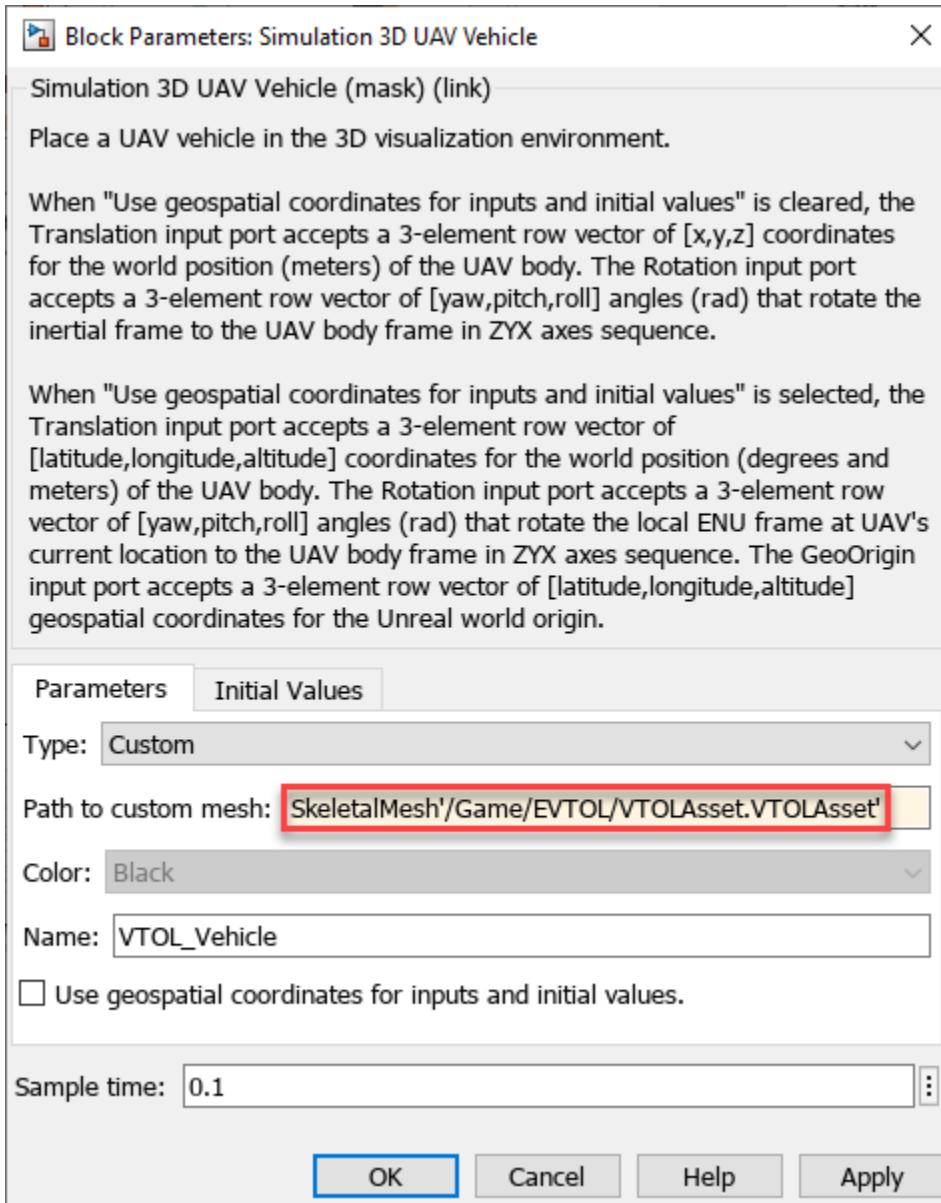
You can verify that **Element 1** is the fuselage by selecting **isolate** to see which body the element is representing. Set the **Slot Name** of Element 1 to **VehicleBody**. Then, save the VTOL asset.



To copy the reference path to this asset, right-click the **VTOLAsset** skeletal mesh object in **Content Browser** pane and click **Copy Reference**.

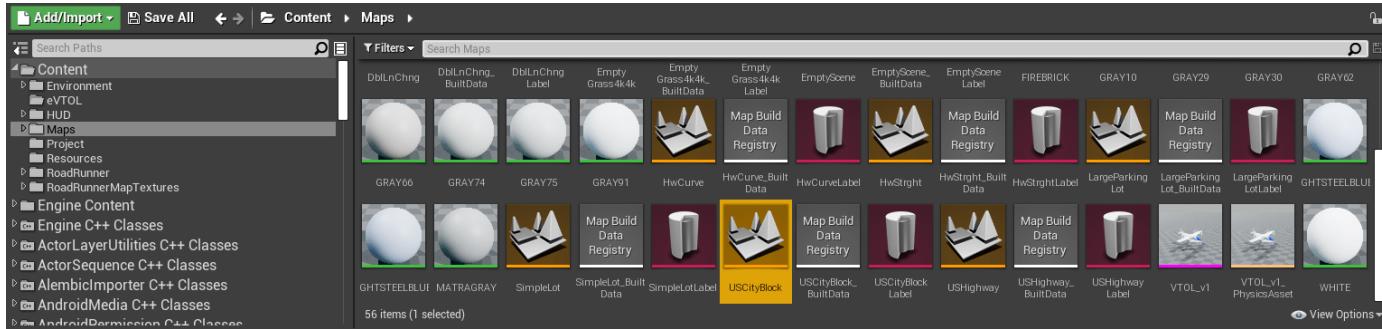


In the **Simulation 3D UAV Vehicle** block, set the **Type** parameter to **Custom** and then set the **Path to custom mesh** parameter to the reference path.



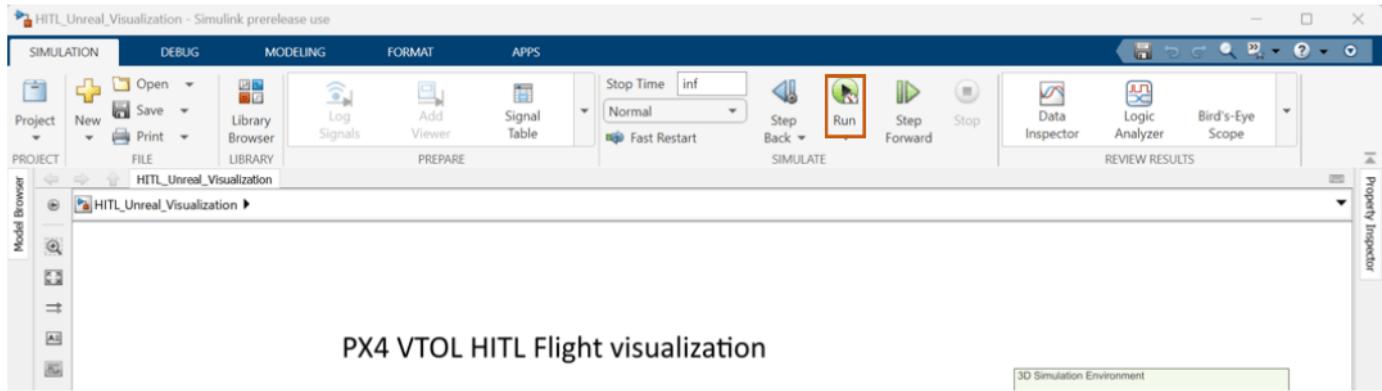
Load US City Block Scene

Navigate to Maps in Content Browser and click on USCityBlock to load the US City Block scene.



Run Unreal Engine Visualization in Second MATLAB Instance

Run the HITL_Unreal_Visualization.slx model.



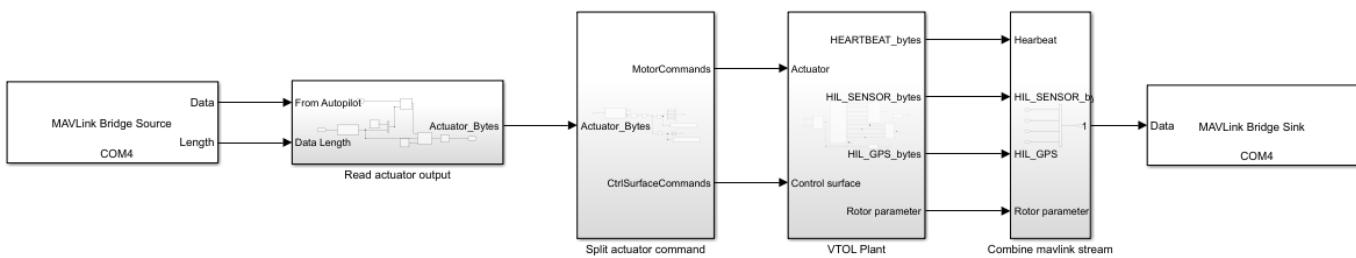
When the Simulink mode is initializing, click **Play** in Unreal Editor. Zoom out using mouse scroll to view the VTOL UAV. **The VTOL UAV will remain stationary in the starting position until you run the HITL simulation in the next step.**



Run HITL Simulation from First MATLAB Instance

The HITL_Plant_top.slx file contains the HITL plant model. The model consists of the following blocks:

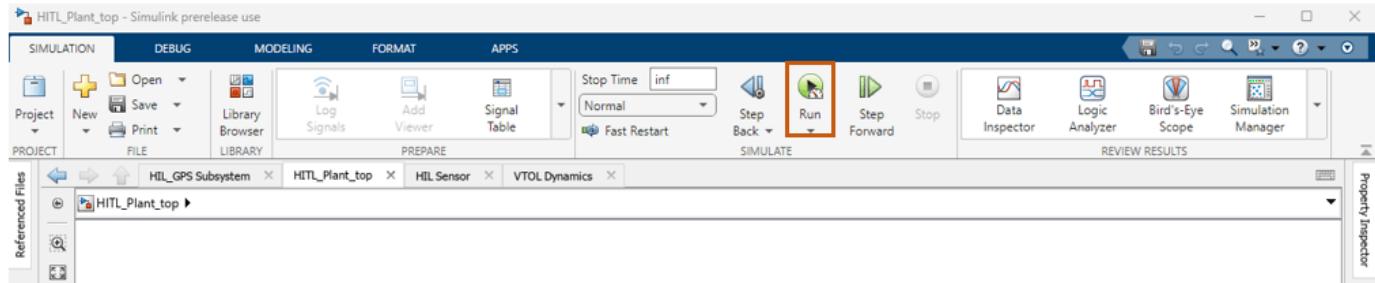
- MAVLink Bridge Source block — Reads the MAVLink messages from the Pixhawk board and sends it to VTOL Plant subsystem
- VTOL Plant subsystem — Plant model for the VTOL tilt rotor UAV, which simulates the aerodynamics, propulsion, ground contact, and sensors. The subsystem outputs sensor, rotor parameter, and GPS data.
- MAVLink Bridge Sink block — Parses VTOL Plant subsystem as MAVLink messages and sends the messages back to Pixhawk by using serial connection



Open the HITL_Plant_top.slx model by using the **Open HITL Plant Model** shortcut or by running the following code block in the first instance of MATLAB.

```
plant_mdl = "HIL_Plant_top.slx";
open_system(plant_mdl)
```

To start the HIL simulation, first run the HIL_Plant_top.slx



Then, Open QGroundControl and configure the actuators as shown. For more information, see "Configure and Assign Actuators in QGC".

Vehicle Setup

Geometry: Standard VTOL

Motors: 4

	Position X	Position Y	Direction CCW	Axis
Motor 1:	0.49	0.49	<input checked="" type="checkbox"/>	Upwards
Motor 2:	-0.49	-0.49	<input checked="" type="checkbox"/>	Upwards
Motor 3:	0.49	-0.49	<input type="checkbox"/>	Upwards
Motor 4:	-0.49	0.49	<input type="checkbox"/>	Upwards

Control Surfaces: 5

Type	Roll Scale	Pitch Scale	Yaw Scale	Trim
Servo 1: Left V-Tail	0.50	0.50	0.00	
Servo 2: Right V-Tail	0.50	-0.50	0.00	
Servo 3: Left Aileron	-0.50		0.00	
Servo 4: Elevator	1.00		0.00	
Servo 5: Rudder		1.00	0.00	

Lock control surfaces in hover: (Param not available)

Actuator Testing

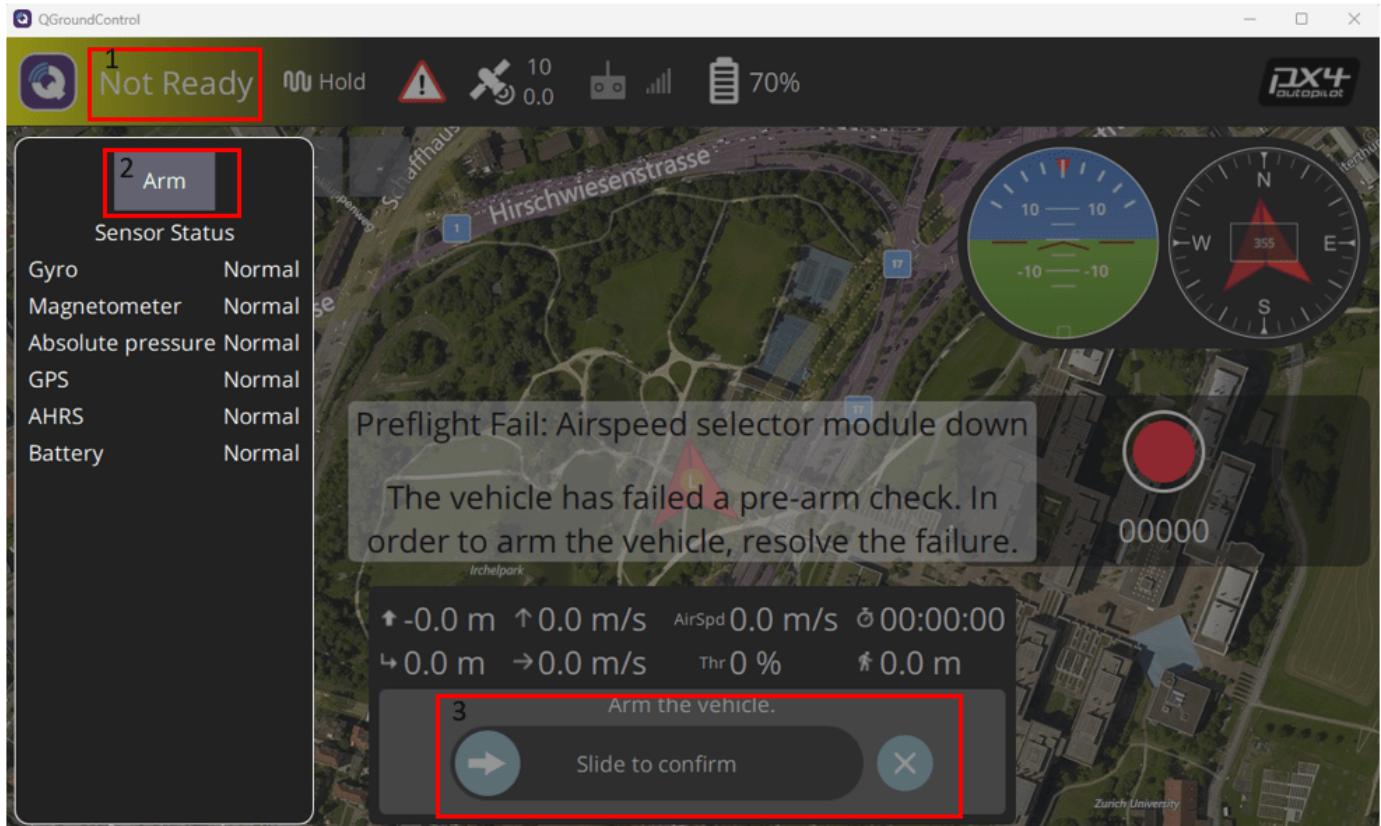
Propellers are removed - Enable sliders

Actuators: All Motors, Motor 1, Motor 2, Motor 3, Motor 4, Left V-Tail, Right V-Tail, Left Aileron, Elevator, Rudder

Actuator Outputs

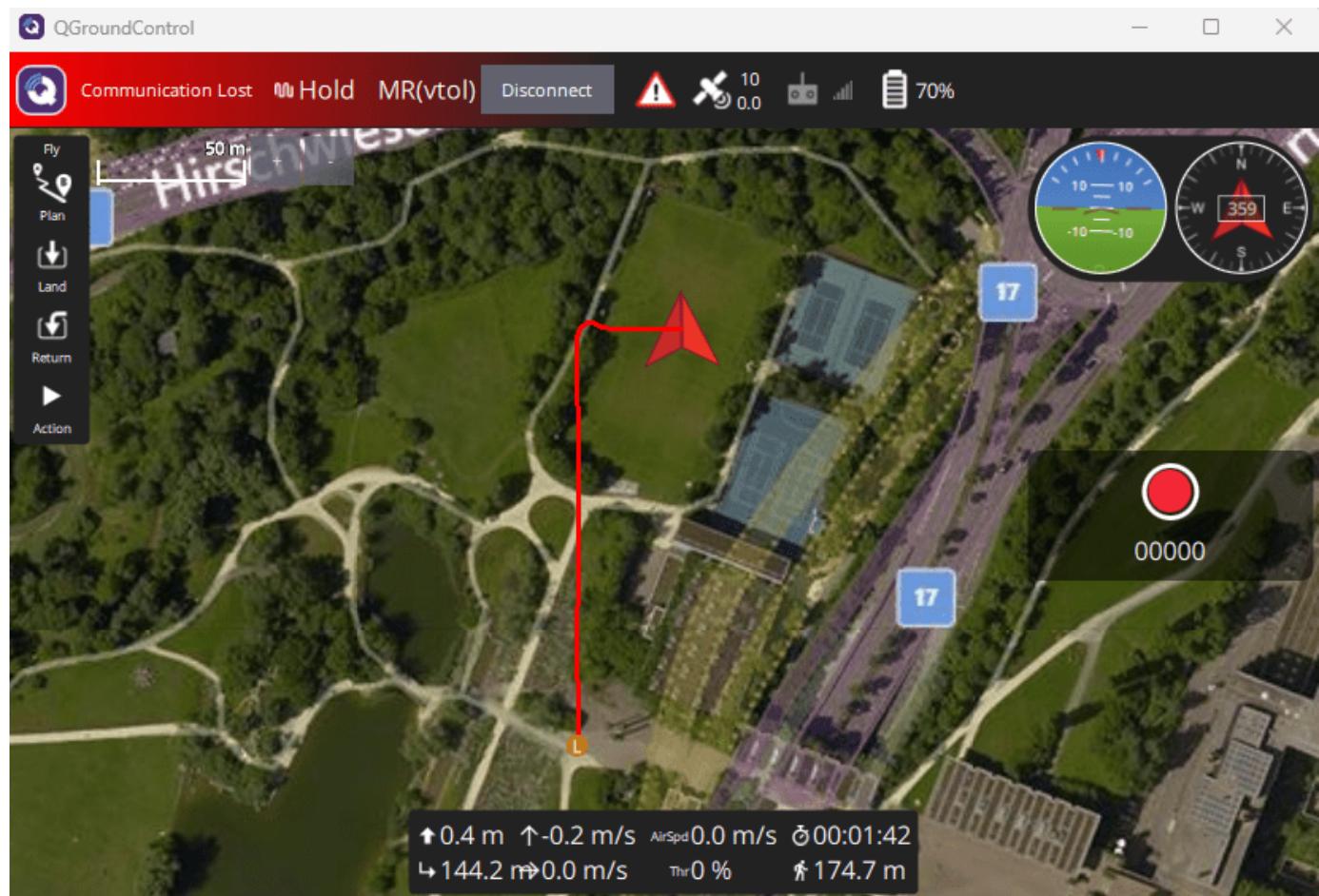
Function	Rev Range (for Servos)
Channel 1: Motor 1	
Channel 2: Motor 2	
Channel 3: Motor 3	
Channel 4: Motor 4	
Channel 5: Left V-Tail	
Channel 6: Right V-Tail	
Channel 7: Left Aileron	
Channel 8: Elevator	
Channel 9: Rudder	
Channel 10: Disabled	
Channel 11: Disabled	
Channel 12: Disabled	
Channel 13: Disabled	
Channel 14: Disabled	
Channel 15: Disabled	
Channel 16: Disabled	

Next, select **Fly View** in QGroundControl. Arm the VTOL UAV to start the mission by selecting **Arm**.



After the VTOL UAV is armed, the UAV takes off starts the mission, as shown in QgroundControl and the Unreal Engine visualization.

1 Blocks





Before resuming to other examples in this example series, you must close the `VTOLRefApp.prj` Simulink project by running this command in the Command Window:

```
close(prj)
```

See Also

[“Design and Tune Controller for VTOL UAV”](#) | [Simulation 3D Scene Configuration](#) | [Simulation 3D UAV Vehicle](#)

Related Examples

- [“Customize Unreal Engine Scenes for UAVs”](#)

Getting Started with Actuator Control over PWM

This example shows you how to use the PX4 Actuator Write block to write actuator values over PWM.

Introduction

UAV Toolbox Support Package for PX4® Autopilots enables you to write actuator values to the Motor ESCs and Servos connected over PWM or UAVCAN/DroneCAN interface.

In this example, you will learn how to create and run a Simulink® model to write actuator values for controlling Motors and Servos.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

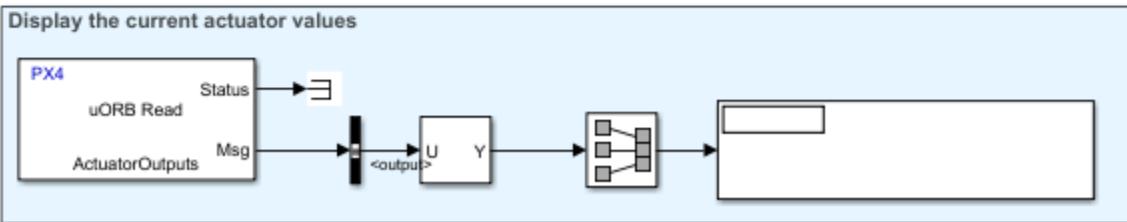
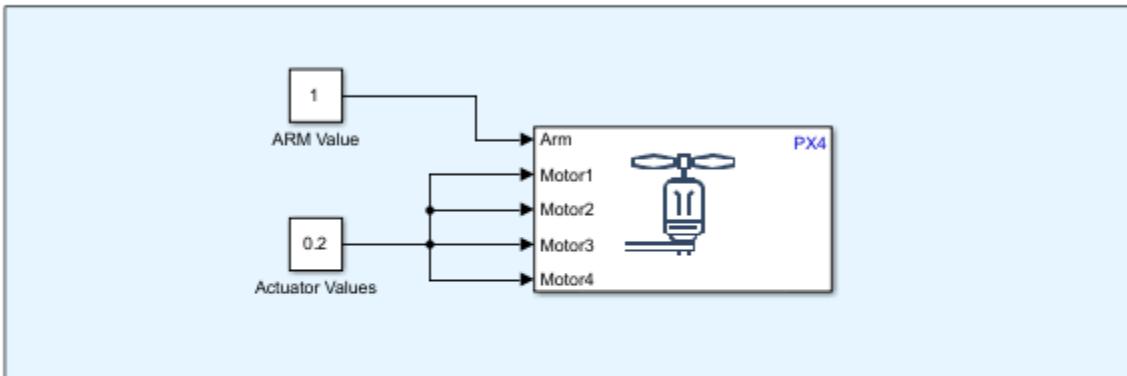
To run this example, you will need the following hardware:

- Supported PX4 Autopilot
- Micro USB type-B cable
- Cathode Ray Oscilloscope (CRO) or Logic analyzer (for PWM)

Model

Open the px4ActuatorWrite model.

Getting Started with Writing Actuator values



Assuming you are designing a quadcopter and aim to send actuator values to the four motors, this model utilizes the PX4 Actuator Write block to transmit actuator values to the four motors. Passing a true value to the "Arm" input, arms the actuators. In this model, a single value of 0.2 is sent to all four motors to provide 20% thrust.

If your motors are connected and powered, ensure the propellers are removed for safety.

The `ActuatorOutputs` uORB topic is used to read the actuator values back into Simulink and display them.

Note: `ActuatorOutputs` will reflect only the values for PWM Main channels. For PWM AUX and UAVCAN, `ActuatorOutputs` will not show the actuator values that you are writing.

Writing Actuator values over PWM

In this section, you will learn how to configure and write actuator values to the PWM pins.

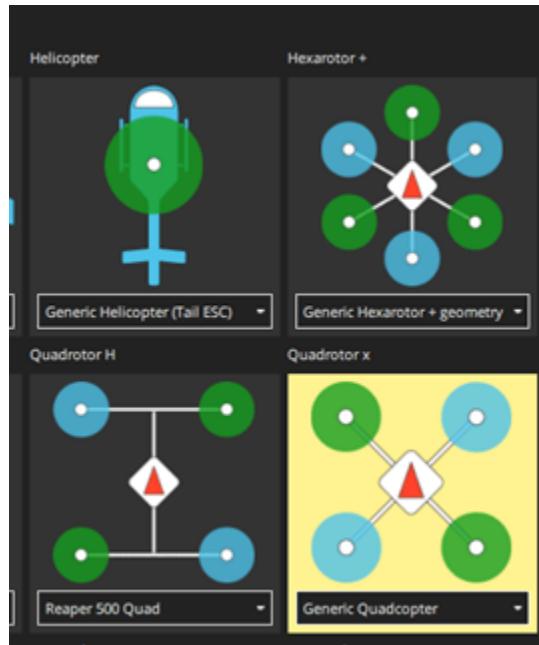
Step1: Configure Actuators from QGroundControl

For Firmware version 1.14, the actuators must be configured in QGroundControl (QGC) before you can use the PX4 Actuator Write block in Simulink. For more information, see “Configure and Assign Actuators in QGC”.

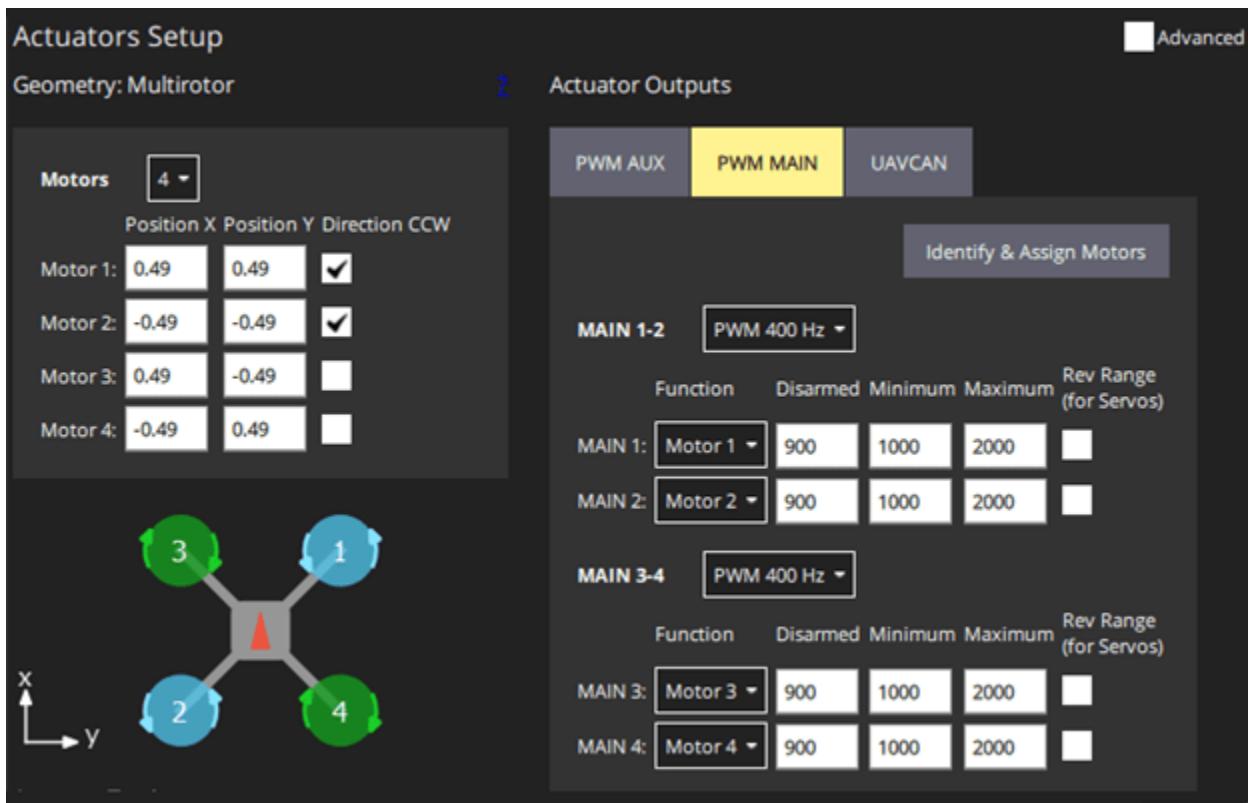
1. Connect the PX4 Autopilot to your machine and wait until QGroundControl connects with the hardware.

Note: MAVLink must be enabled over USB (`/dev/ttyACM0`) for QGC connectivity. For more information, see “Enabling MAVLink in PX4 Over USB”.

2. For this example, as we are focusing on a quadcopter, select **Generic Quadcopter** as the airframe in the **Vehicle Setup -> Airframe** tab of QGC. Reboot the hardware for this change to take effect.



3. After selecting the airframe, go to the **Actuator** tab (**Vehicle Setup -> Actuators**) and assign the four motors to the first 4 Main PWM channels, respectively. Based on this selection, the actuator values will now be sent over the PWM Main channels. For example, the values written to Motor1 will be sent to PWM Main Channel 1. Note that the PWM frequency is set to 400Hz.



4. Leave the other Main channels and PWM AUX channels as **Disabled**.

5. You can also adjust the minimum and maximum values based on your requirements and the specifications of your ESC. For example, to maintain consistency with the PX4 PWM Output block, the Disarm, Min, and Max values are updated in this example. However, you can also use the default values.

After completing these selections, close or disconnect QGC.

Step2: Configure Model for Supported Pixhawk Hardware

In this section, you will configure the model for the supported Pixhawk hardware.

The model used in this example is configured for Pixhawk 6x hardware. If you are using a different supported Pixhawk hardware, perform these steps.

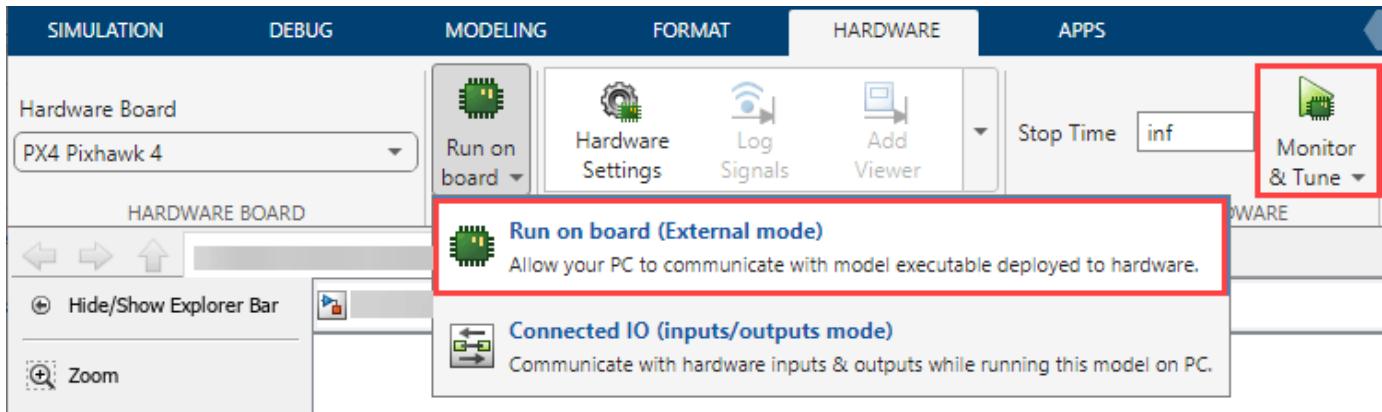
1. In your Simulink model, Go to **Modeling > Model Settings** to open the Configuration Parameters dialog box.
2. Click **Hardware Implementation** pane, and select the required Pixhawk hardware from the **Hardware board** list.
3. For running in external mode, navigate to **Target hardware resources > MAVLink** and clear **Enable MAVLink on /dev/ttyACM0**.
4. Click **Apply** and **OK**.

Step3: Configure Model for Supported Pixhawk Hardware

1. Connect one of the first four MAIN PWM channels (for example, Main 1) of the PX4 autopilot to an oscilloscope or a logic analyzer.

2. Run the model using one of the following options.

- **Monitor & Tune:** To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

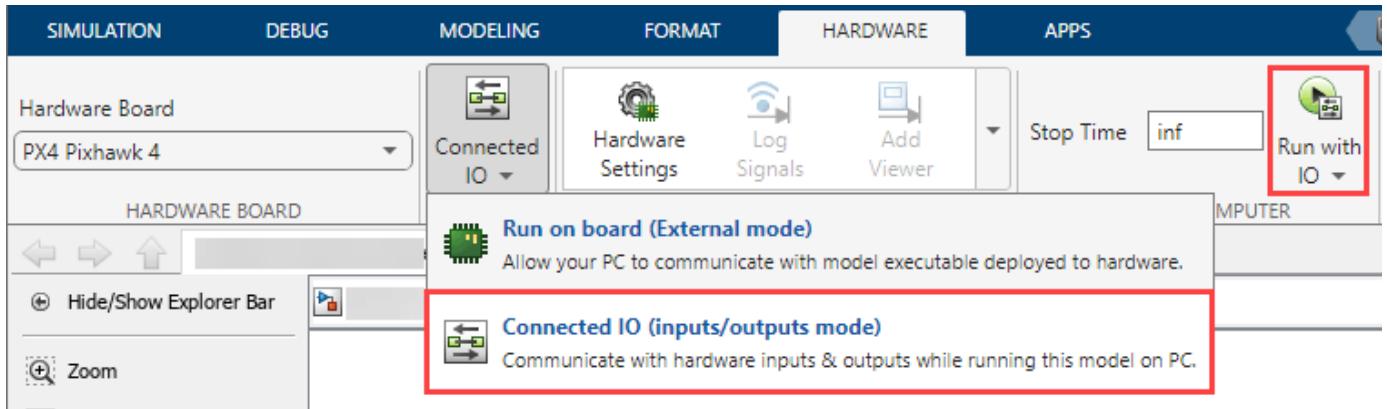


Wait for the code generation to be completed. Whenever the dialog box appears instructing you to reconnect the flight controller to the serial port, click **OK** and then reconnect the PX4 Autopilot on the host computer.



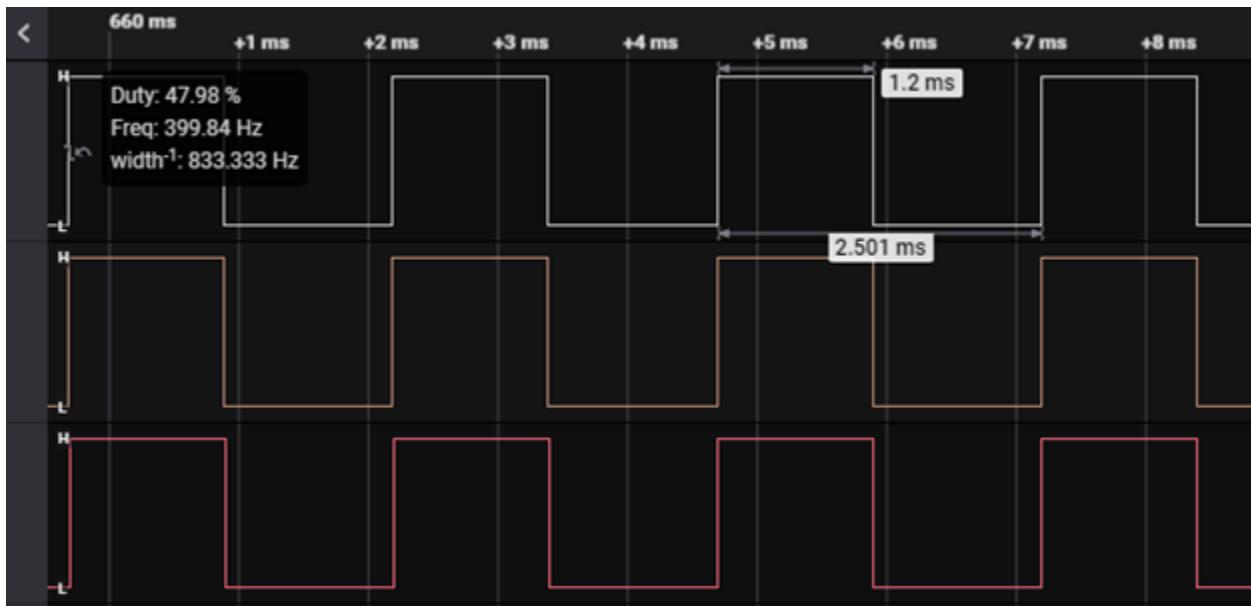
The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

- **Connected IO:** To run this model in the Connected I/O mode, on the **Hardware** tab, in the **Mode** section, select **Connected IO** and then click **Run with IO**.

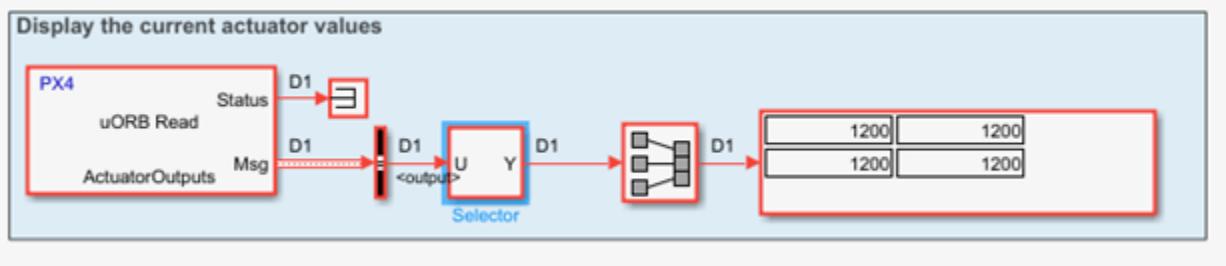


If the Connected IO firmware is not deployed on the hardware, then the dialog box instructing you to reconnect the flight controller to the serial port appears otherwise the dialog box is not displayed. If the dialog box appears, click **OK** and then reconnect the PX4 Autopilot on the host computer.

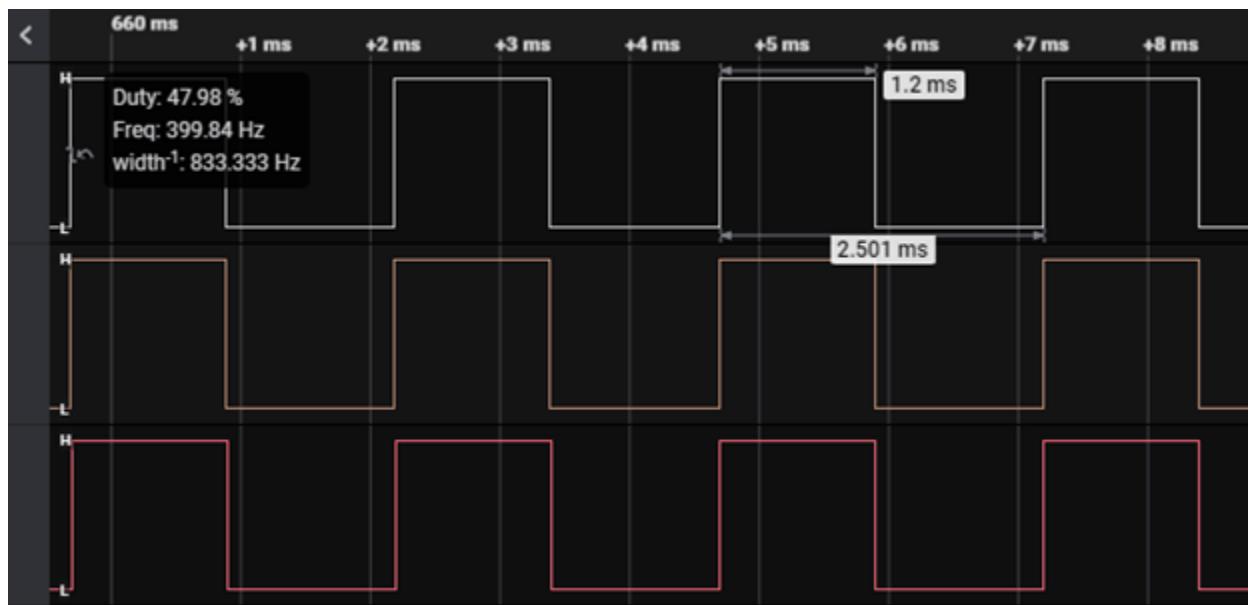
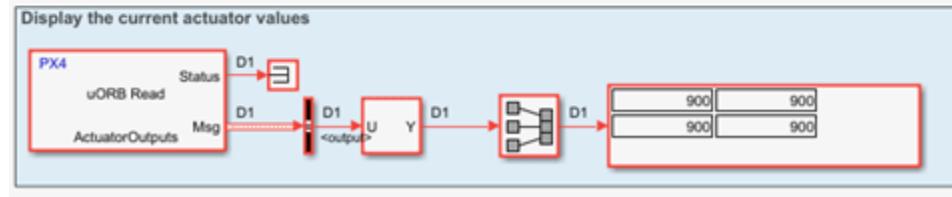
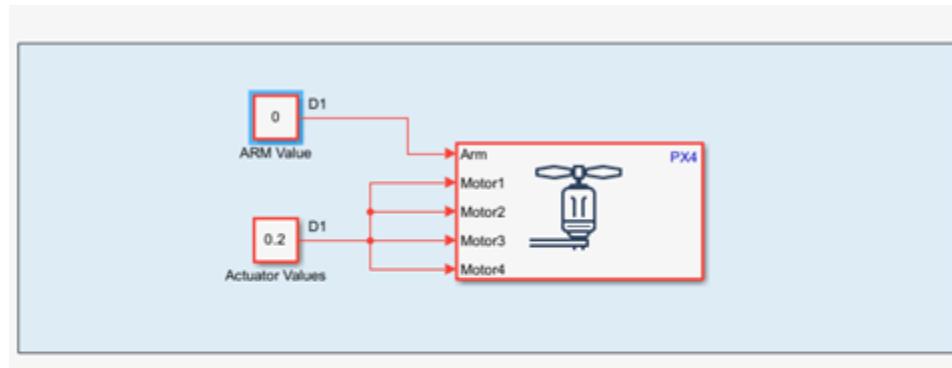
3. Connect the CRO or Logic analyzer to the PWM Main channels 1, 2, 3, and 4.
4. Observe the PWM signals and verify the ON time, which should correspond to the values you have input into the block. In this example, an input of 0.2 is provided, and based on the QGC actuator configuration where the Max is 2000 and Min is 1000, you should observe a PWM signal with a width of 1200 microseconds. The frequency is set to |400|Hz, as specified in QGC.



5. The `ActuatorOutputs uORB` topic will display the PWM values as well. Note that this information will reflect only for Main channels and not for Aux channels.



6. Change the actuator value and observe the PWM signal. The PWM ON value should change according to the actuator value input. Providing a false value to the Arm input will result in the disarm PWM values being displayed.



Writing to Actuator values to Servos

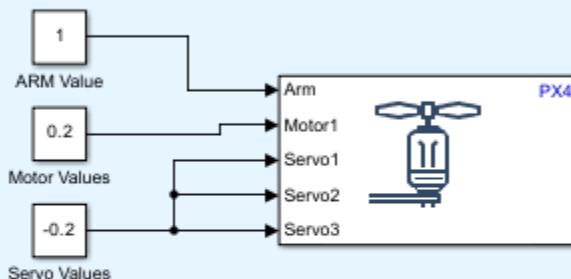
In this example, you learned how to write actuator values to motors using both PWM and UAVCAN interfaces for a Quadcopter airframe. If your airframe also requires servos, you can follow a similar approach.

For example, if you choose a fixed wing airframe, then you need to write actuator values to servo as well along with Motor. You need to configure the servos as well in QGC before writing actuator values from Simulink.

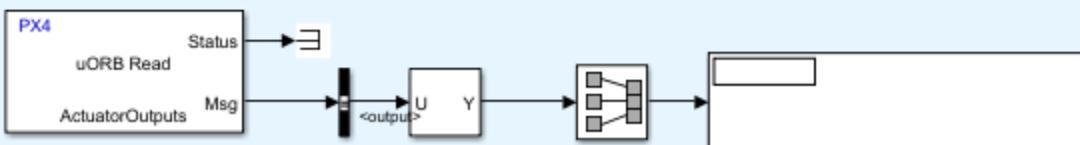
Model

Open the px4ActuatorServoWrite model.

Getting Started with Writing Actuator Servo values

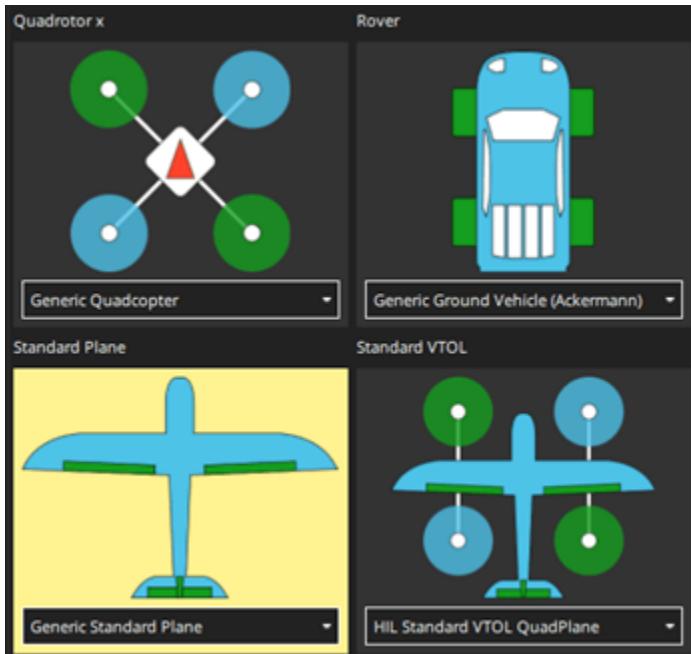


Display the current actuator values

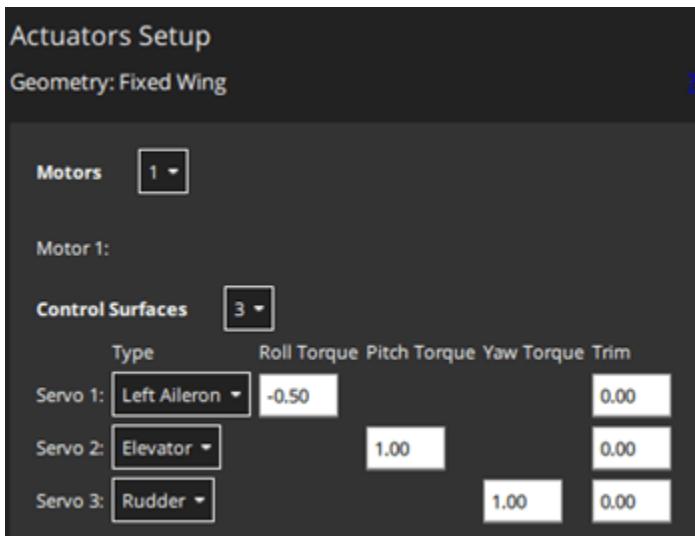


Step1: Configure the Actuators from QGroundControl

1. Open QGC and after PX4 Autopilot is connected, select **Generic Standard Plane** as the airframe in the **Vehicle Setup -> Airframe** tab of QGC. Reboot the hardware for this change to take effect.



2. After selecting the airframe, go to the **Actuator** tab (**Vehicle Setup -> Actuators**). Now, the option to configure servos appears. This example uses three servos.



3. Assign Motor 1 to PWM Main Channel 1, and the servos to the first three Aux channels.

Function	Disarmed	Minimum	Maximum	Rev Range (for Servos)
AUX 1: Left Aileron	1500	1000	2000	
AUX 2: Elevator	1500	1000	2000	
AUX 3: Rudder	1500	1000	2000	
AUX 4: Disabled	1000	1000	2000	

Note: It is recommended to assign Motors to the Main channels and servos to the Aux channels only.

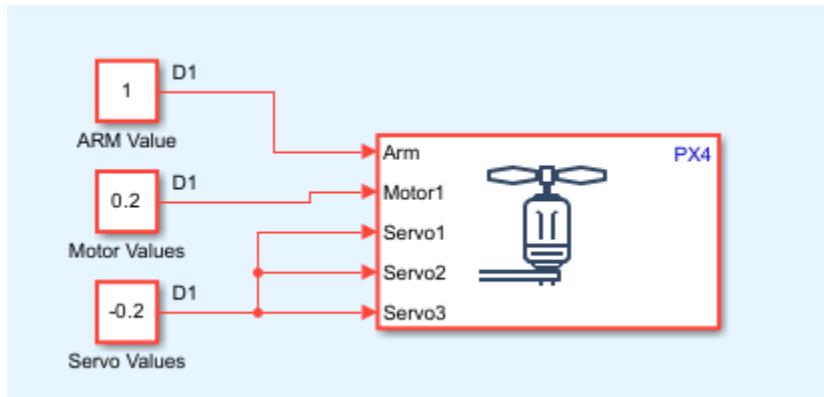
4. Based on the selection you have made, the actuator values written to Servo 1, 2, and 3 will now be sent to Aux channel 1, 2, and 3, respectively. Note that the PWM Main channel frequency is set to | 400|Hz, and the Aux channel frequency is set to 50Hz. After completing this setup, close QGC.

Step2: Configure the Model for Servo Write

In this step, modify the example model to direct values to the servos.

1. Double-click the Actuator Write block in your Simulink model. Select only one Motor in the **Motors** tab and three servos from the **Servos** tab.

2. Provide the actuator values to both the motor and servos. For example, assign a value of 0.2 to Motor 1 and -0.2 to all three servos.



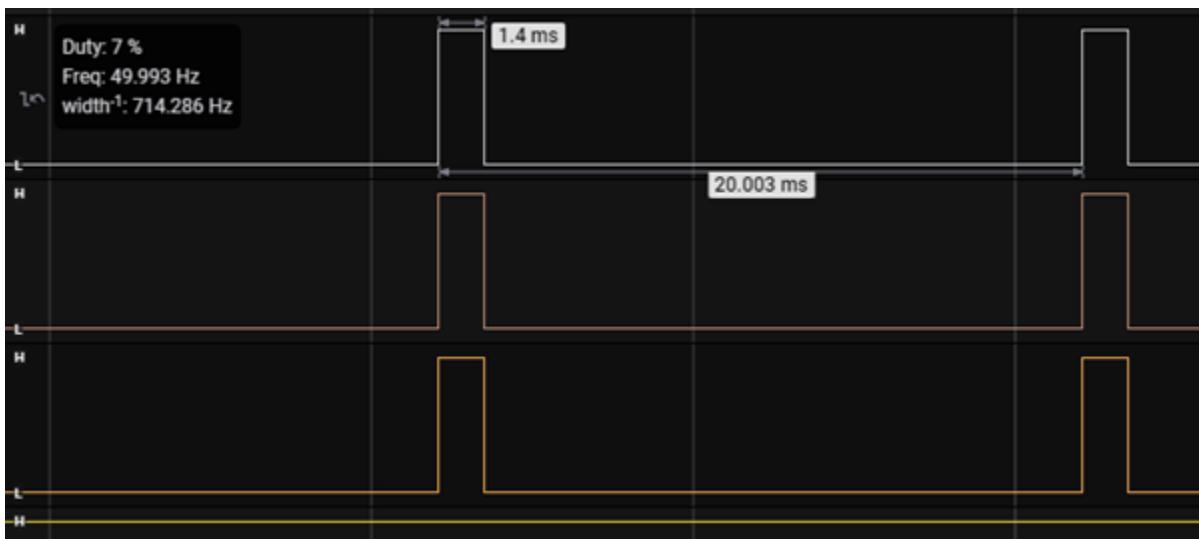
Step3: Run the Model and Validate the actuator values

1. Connect the CRO or Logic analyzer to the AUX Channel 1,|2| or 3 and PWM Main channel 1.
2. Run the Simulink model using Monitor and Tune option or in Connected IO mode. Observe the PWM signals in AUX channels.
3. Since the minimum and maximum PWM values are set as 1100 and 1900 respectively in QGroundControl (QGC), a 20% PWM value for motors would be calculated as follows:

$$|(1900 - 1100) * 0.2 + 1100 = 1260\text{us}|$$



For AUX channels, the Min value is set as 1000 and Max value is 2000. Since for servos the actuator range is [-1, 1], the zero-actuator value will correspond to PWM value 1500. For an actuator value of -0.2, the corresponding PWM value can be calculated using the formula:
 $1500 - 0.2 * (2000 - 1000) / 2 = 1400$. Frequency is |50|hz.



Getting Started with Actuator Control over UAVCAN/DroneCAN

This example shows you how to use the PX4 Actuator Write block to write actuator values over UAVCAN/DroneCAN interface.

Introduction

UAV Toolbox Support Package for PX4® Autopilots enables you to write actuator values to the Motor ESCs and Servos connected over PWM or UAVCAN/DroneCAN interface.

In this example, you will learn how to create and run a Simulink® model to write actuator values for controlling Motors and Servos.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

Required Hardware

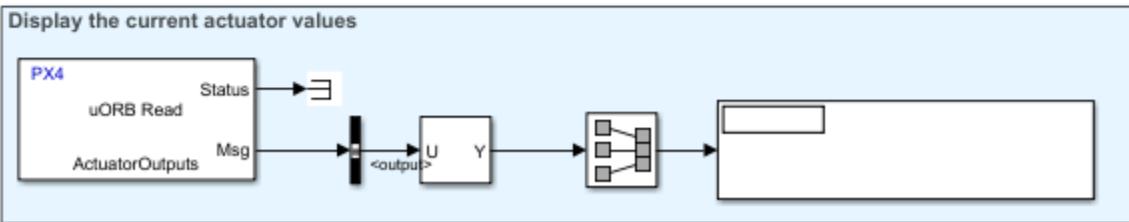
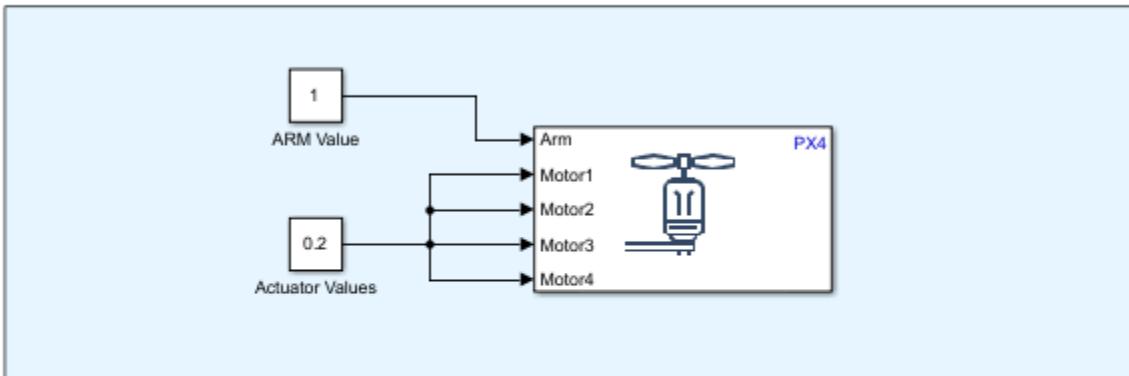
To run this example, you will need the following hardware:

- Supported PX4 Autopilot
- Micro USB type-B cable

Model

Open the px4ActuatorWrite model.

Getting Started with Writing Actuator values



Assuming you are designing a quadcopter and aim to send actuator values to the four motors, this model utilizes the PX4 Actuator Write block to transmit actuator values to the four motors. Passing a true value to the "Arm" input, arms the actuators. In this model, a single value of 0.2 is sent to all four motors to provide 20% thrust.

If your motors are connected and powered, ensure the propellers are removed for safety.

The `ActuatorOutputs` `uORB` topic is used to read the actuator values back into Simulink and display them.

Note: `ActuatorOutputs` will only reflect the values for PWM Main channels. For PWM AUX and UAVCAN, `ActuatorOutputs` will not show the actuator values that you are writing.

Writing Actuator Values Over UAVCAN/DroneCAN

In this section, you will explore how to configure and write actuator values over the UAVCAN/DroneCAN Interface.

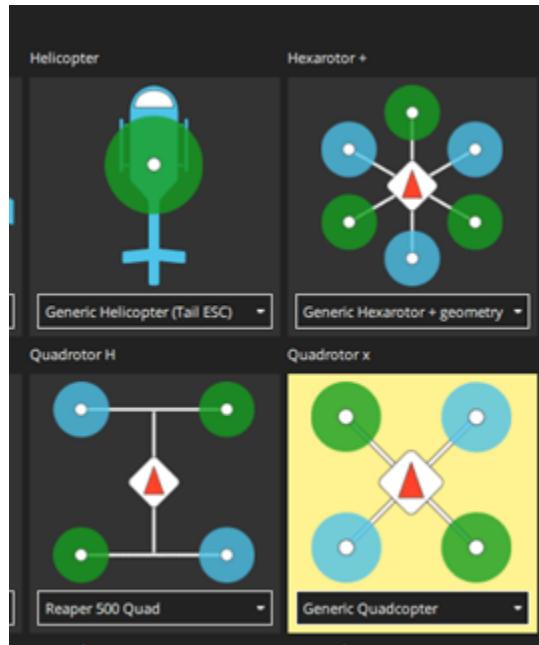
Step 1: Configure the Actuators from QGroundControl

Perform these steps to configure QGroundControl (QGC) for UAVCAN/DroneCAN.

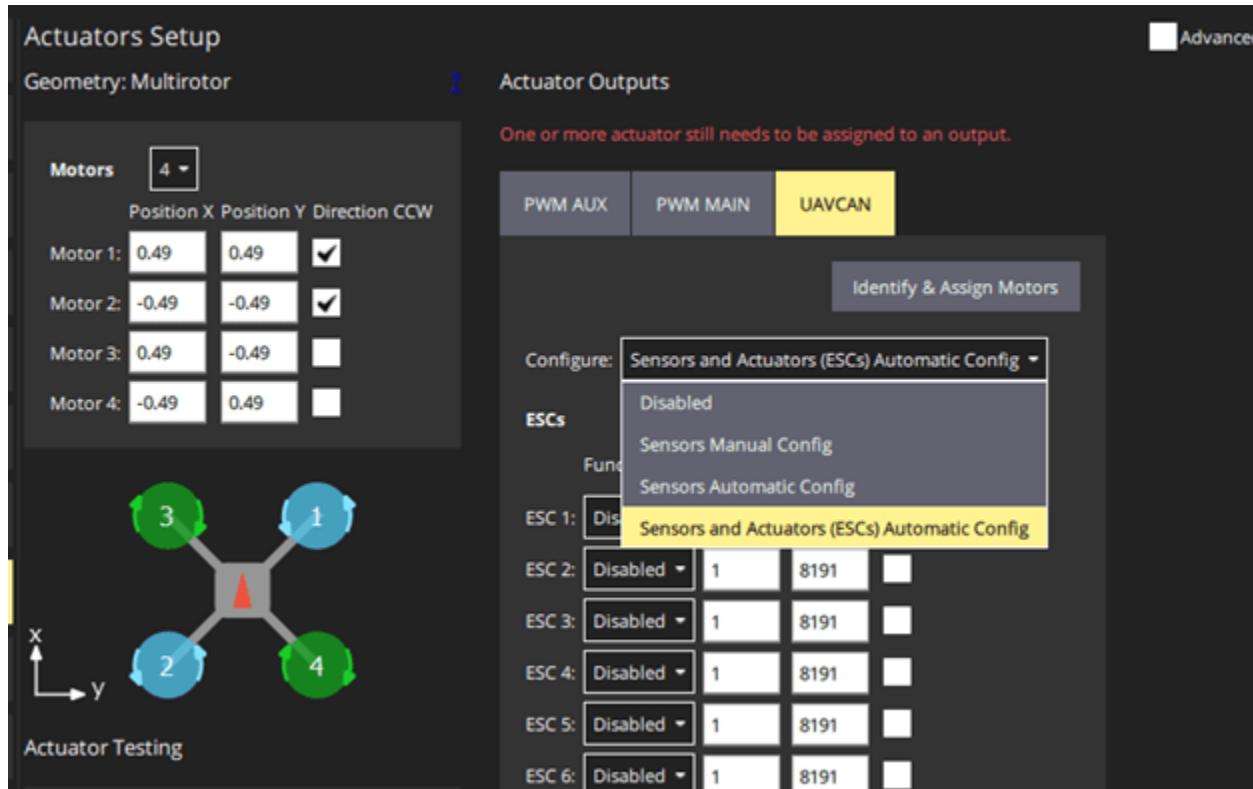
Note: MAVLink needs to be enabled over USB (`/dev/ttyACM0`) for QGC connectivity. For more information, see "Enabling MAVLink in PX4 Over USB".

1. Connect the PX4 Autopilot to your computer and wait until QGroundControl establishes a connection with the hardware.

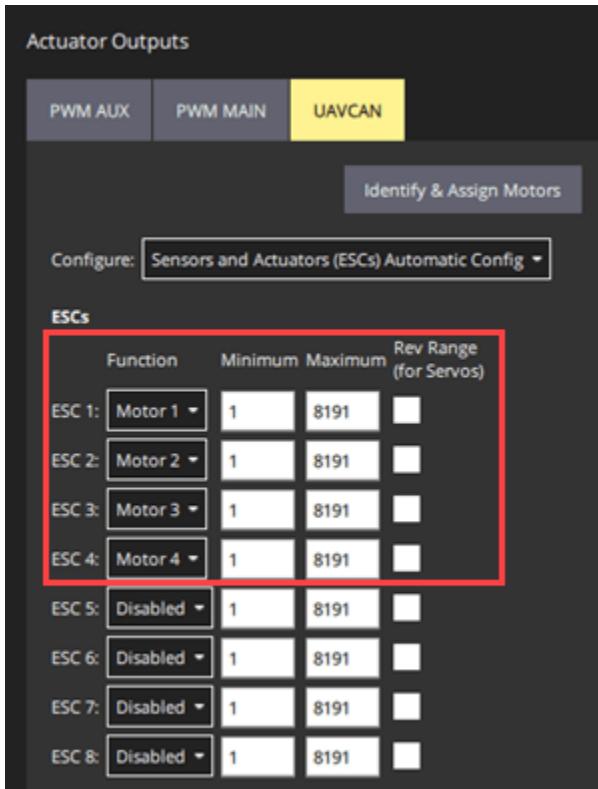
2. For this example, as we are focusing on a quadcopter, select **Generic Quadcopter** as the airframe in the **Vehicle Setup -> Airframe** tab of QGC. Reboot the hardware for this change to take effect.



3. After selecting the airframe, go to the **Actuator** tab (**Vehicle Setup -> Actuators**) and choose **Sensor and Actuators (ESCs) Automatic Config** option. Reboot the hardware again for this change to take effect.



4. Assign the four motors to the first 4 UAVCAN ESC channels respectively. With this configuration, the actuator values will now be sent over the UAVCAN ESCs.



5. Keep the other UAVCAN ESCs, Servos, PWM Main channels, and PWM AUX channels as **Disabled**.

After completing these steps and ensuring your configurations are correct, you can close or disconnect from QGroundControl. If prompted, restart the QGroundControl.

Step2: Configure Model for Supported Pixhawk Hardware

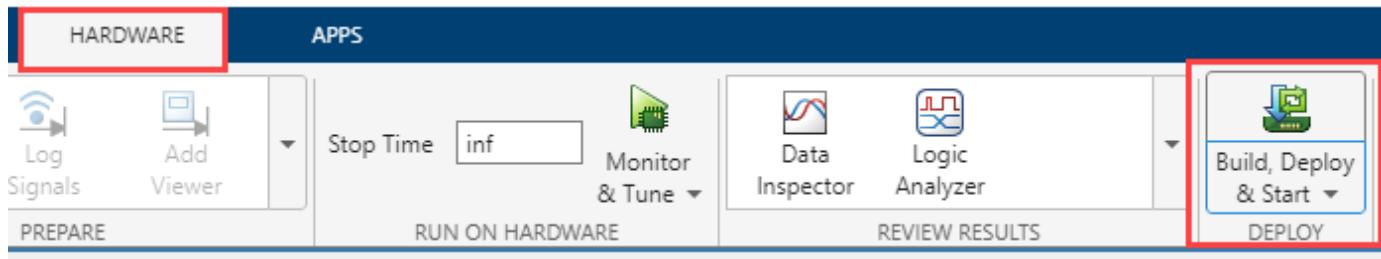
In this section, you will configure the model for the supported Pixhawk hardware.

This example is configured to use Pixhawk 6x hardware. If you are using a different supported Pixhawk hardware, perform these steps. If you are using the Pixhawk 6x hardware, only points 3 and 4 are necessary.

1. In your Simulink model, Go to **Modeling > Model Settings** to open the Configuration Parameters dialog box.
2. Click **Hardware Implementation** pane, and select the required Pixhawk hardware from the **Hardware board** list.
3. Navigate to **Target hardware resources > MAVLink** and select **Enable MAVLink on /dev/ttyACM0**. For more information, see “Enabling MAVLink in PX4 Over USB”.
4. Click **Apply** and **OK**.

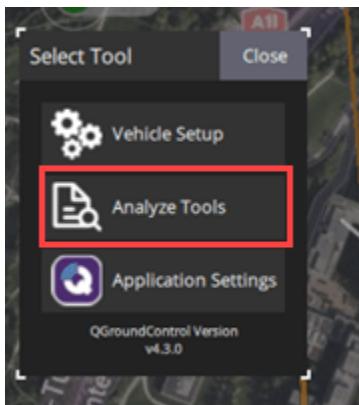
Step 3: Run the Model and Validate the Actuator Values

1. Click **Build, Deploy & Start** from **Deploy** section of **Hardware** tab in the Simulink Toolstrip.



The code will be generated for the Controller model and the same will be deployed to the Pixhawk board (Pixhawk 6x in this example).

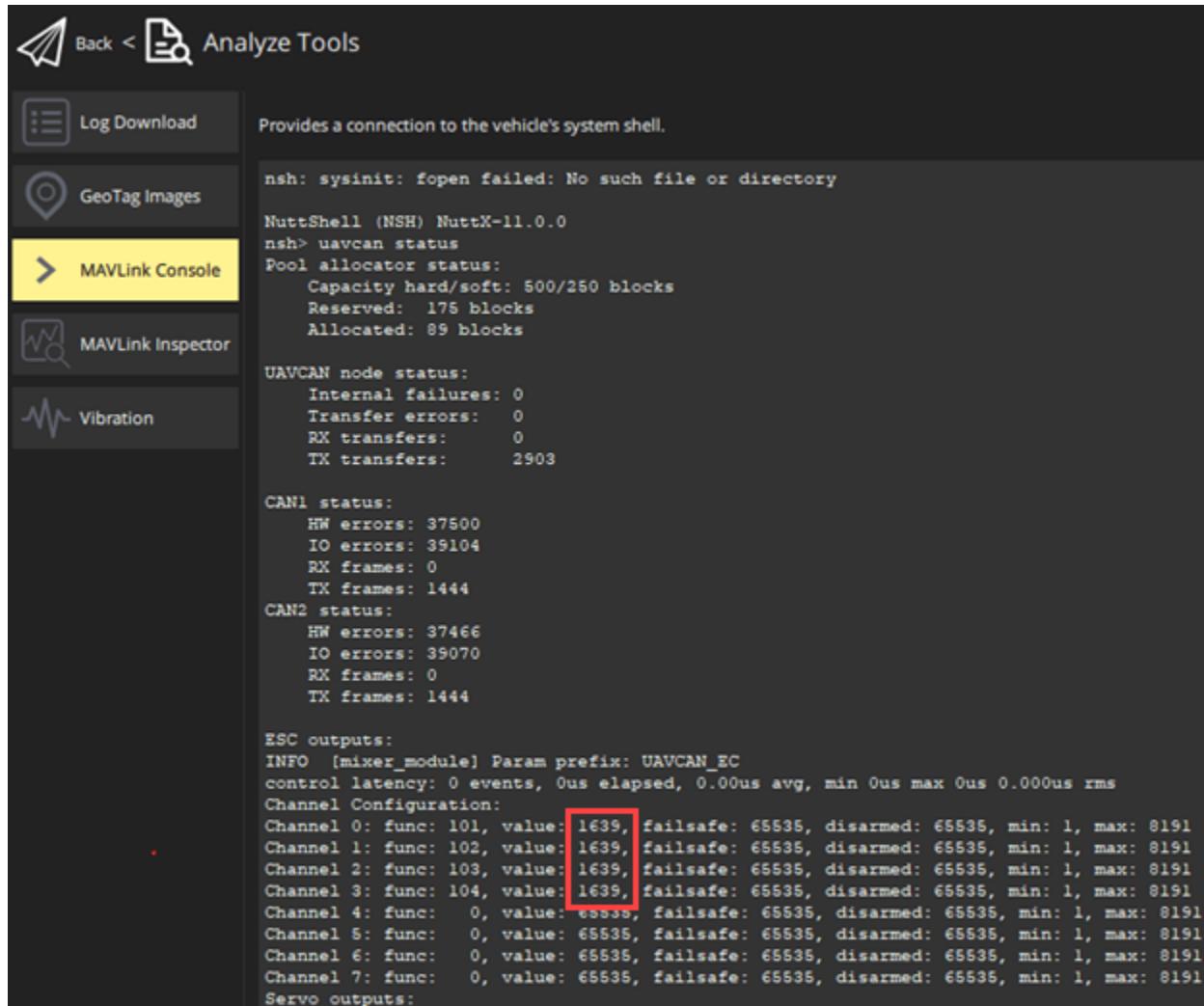
2. Start QGC and wait for it to get connected. Go to **Analyze Tools** and navigate to **MAVLink Console**.



3. Enter the `uavcan status` command in the console.

```
nsh > uavcan status
```

This action will display the actuator values being updated from Simulink. Since the actuator value of 0.2 was written, you should observe the value 1639, which represents 20% of the maximum value.



The screenshot shows the MAVLink Console interface with the following details:

- Log Download**: Provides a connection to the vehicle's system shell.
- GeoTag Images**
- MAVLink Console**: Selected tab, showing system status output.
- MAVLink Inspector**
- Vibration**

```

nsh: sysinit: fopen failed: No such file or directory
NuttShell (NSH) NuttX-11.0.0
nsh> uavcan status
Pool allocator status:
    Capacity hard/soft: 500/250 blocks
    Reserved: 175 blocks
    Allocated: 89 blocks

UAVCAN node status:
    Internal failures: 0
    Transfer errors: 0
    RX transfers: 0
    TX transfers: 2903

CAN1 status:
    HW errors: 37500
    IO errors: 39104
    RX frames: 0
    TX frames: 1444

CAN2 status:
    HW errors: 37466
    IO errors: 39070
    RX frames: 0
    TX frames: 1444

ESC outputs:
INFO [mixer_module] Param prefix: UAVCAN_EC
control latency: 0 events, 0us elapsed, 0.00us avg, min 0us max 0us 0.000us rms
Channel Configuration:
Channel 0: func: 101, value: 1639, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 1: func: 102, value: 1639, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 2: func: 103, value: 1639, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 3: func: 104, value: 1639, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 4: func: 0, value: 65535, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 5: func: 0, value: 65535, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 6: func: 0, value: 65535, failsafe: 65535, disarmed: 65535, min: 1, max: 8191
Channel 7: func: 0, value: 65535, failsafe: 65535, disarmed: 65535, min: 1, max: 8191

Servo outputs:

```

If the UAVCAN/DroneCAN ESC is connected to the CAN bus and has been correctly configured, these values should also be transmitted to the corresponding ESC. For more information on configuring UAVCAN/DroneCAN ESCs, see PX4 documentation.

Simulate Manual Control for Fixed-Wing with PX4 Host Target

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to take manual control inputs from Joystick / RC Transmitter and control the fixed-wing flight.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.

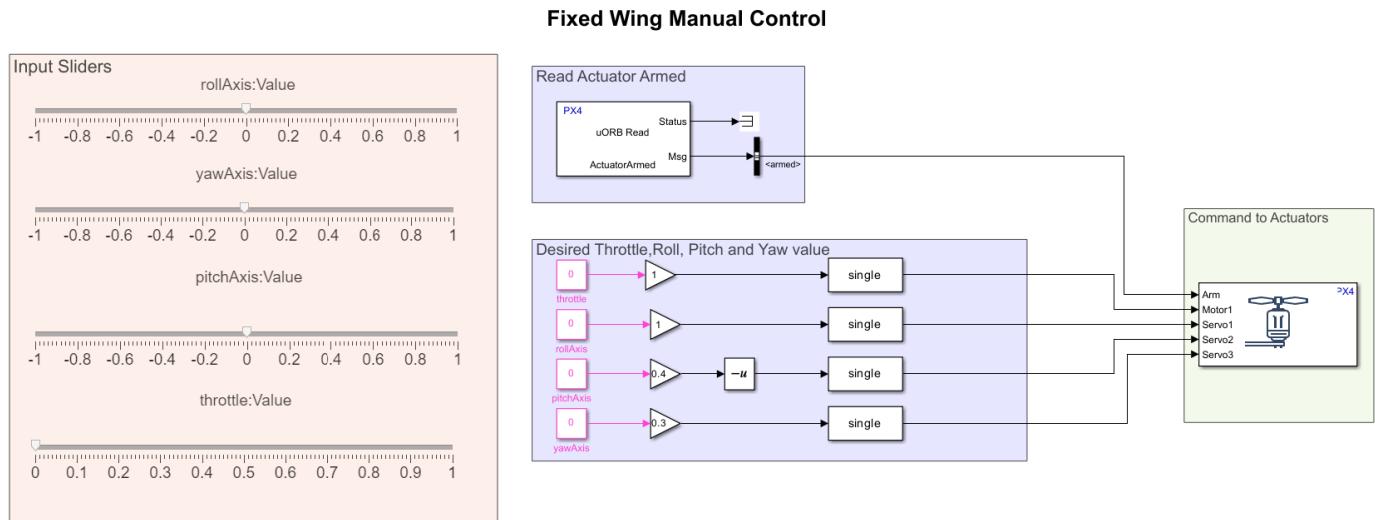
Required Third-Party Software

- QGroundControl (QGC)

Recommended: QGC version 4.3.0

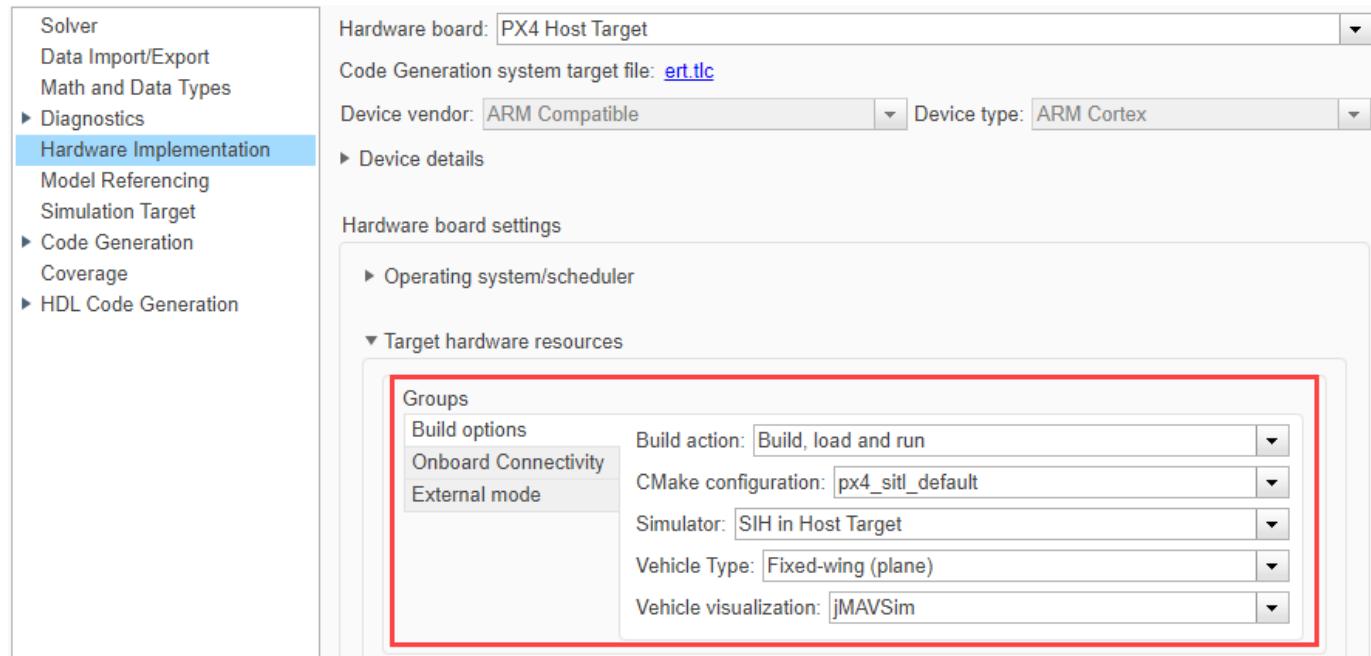
Model

Open the px4demo_ManualControl_FixedWing model.



This example model is configured to use **SIH in Host Target** as simulator, **Fixed-wing (plane)** as vehicle type, and **jMAVSim** as the visualizer. Perform these steps if you have changed the hardware or not using the pre-configured model.

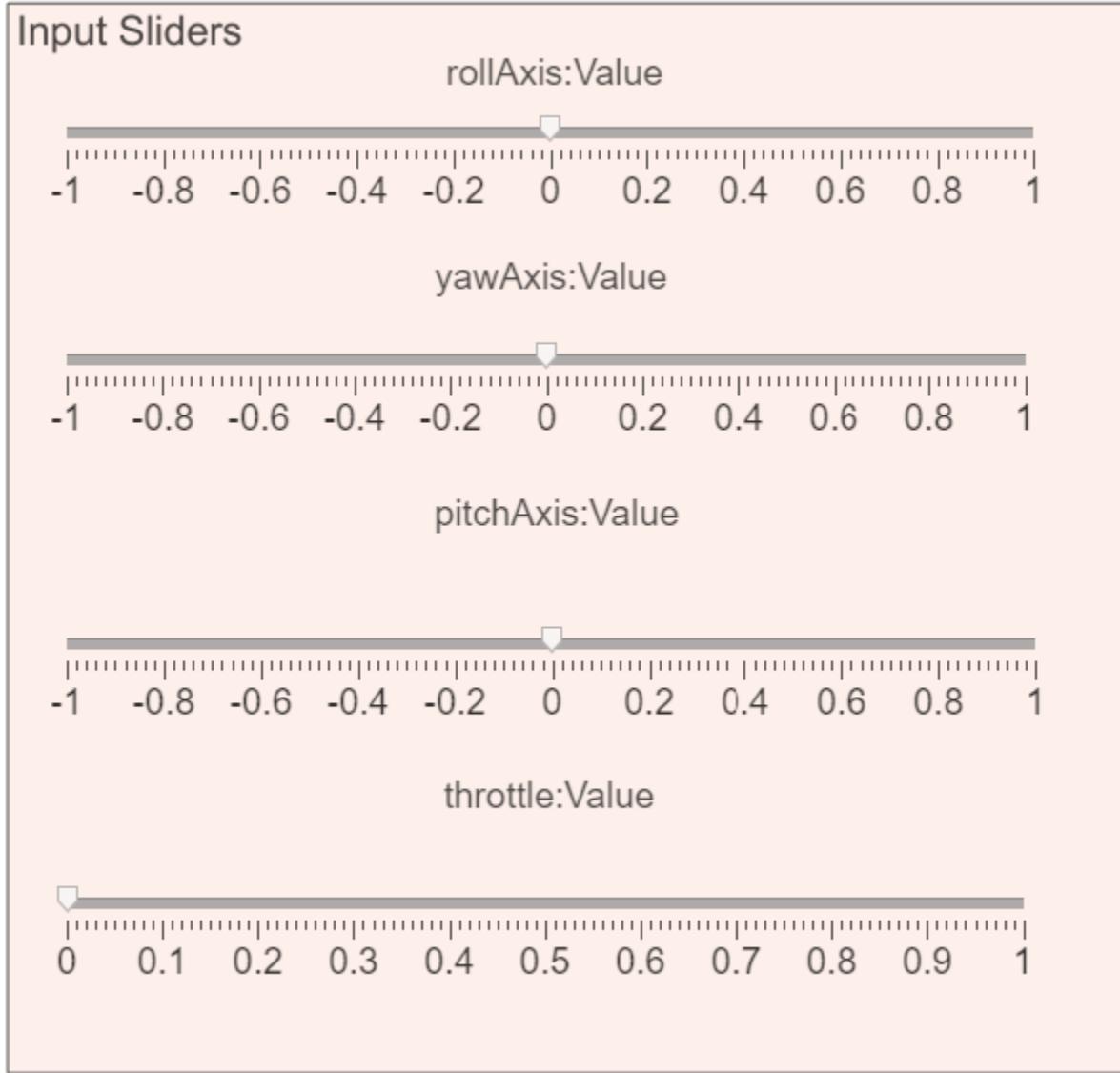
1. Open the model.
2. Go to **Modeling > Model Settings** to open the Configuration Parameters dialog box.
3. Open the **Hardware Implementation** pane, and select **PX4 Host Target**.
4. Expand Target hardware resources for that board and ensure the settings as shown in this image.



Read Desired Position and Current Position

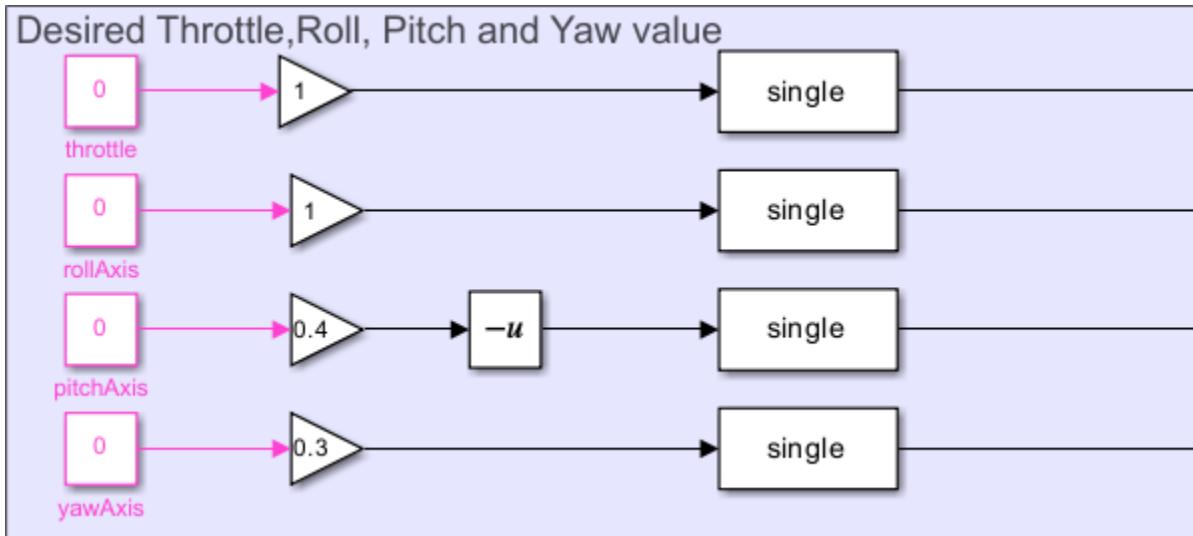
In this example, the quadcopter must try to maintain the desired position and altitude. According to the error in position, the pitch and roll commands are generated and the output to the actuator motors are modified.

The *Input Sliders* in the model can be used to provide the desired coordinates of the flight of the quadcopter.

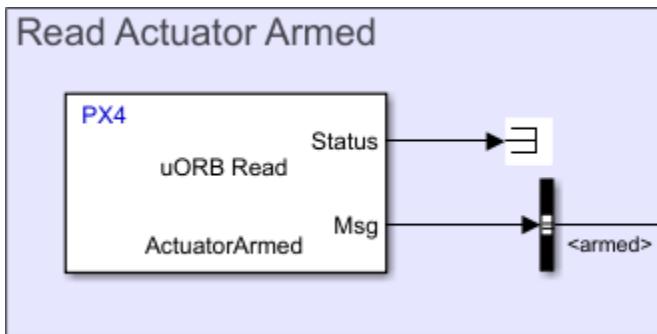


- The variable `des_alt` represents an altitude at which the vehicle hovers. The altitude value can be set using the respective slider or directly changing the value of constant `des_alt`.
- The desired position for X and Y can be set by assigning desired values to constants `des_x` and `des_y` or using respective sliders.

In the *Desired Position, Altitude and Yaw* area, you can also provide dynamic inputs (instead of only static inputs) to test the tracking capabilities of the position controller. In this example, the sine wave is used as the dynamic input and can be selected using respective manual switches.



The vehicle's position in the NED reference frame and its rates can be accessed using uORB Read block, which is configured to read the message 'vehicle_local_position'.



Run the Model on PX4 Host Target and Start jMAVSIM Simulator

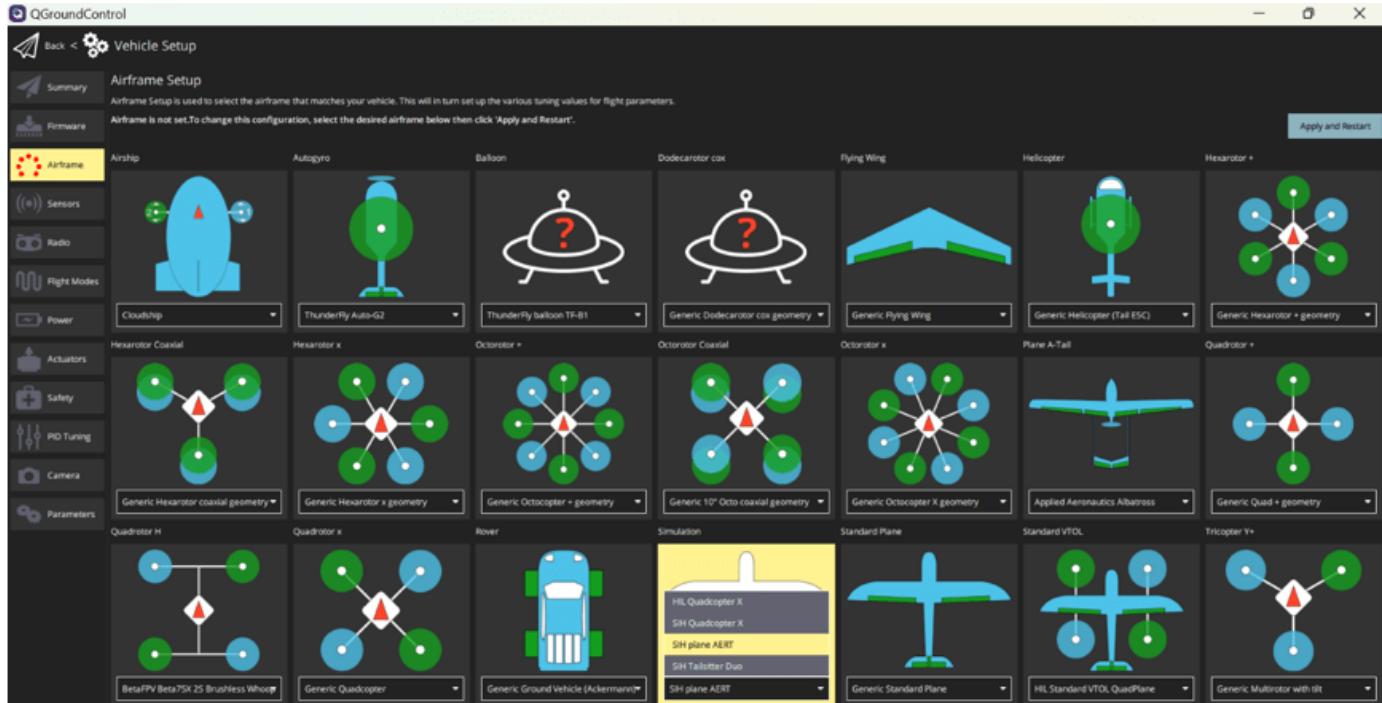
In this task, you select the Hardware Board as PX4 Host Target and start the jMAVSIM simulator.

1. In the **Modeling** tab, click **Model Settings**.
2. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane:
 - Set the **Hardware board** to *PX4 Host Target* (the same option that you selected during Hardware Setup screens).
 - In the **Target Hardware Resources** section, set the **Build options** to **Build, load and run**.
3. Once Simulink model is ready, there are two options to build the Simulink model and launch the simulator:
 - On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. Once the model is deployed in the target hardware (PX4 Host Target in this example), the jMAVSIM simulator is launched and the model executes independently in target hardware without having any dependencies and without communication with Simulink.

- On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune**. This launches QGC and jMAVSIM. In the jMAVSIM simulator, you can monitor and log the data while the model executes in the target hardware.

During Monitor and Tune operation, various parameters and variables can be monitored using display or scope blocks.

4. In the QGC, select SIH plane AERT as the airframe in the **Vehicle Setup -> Airframe** tab of QGC.



5. Click **Apply and Restart**. Restart QGC to apply the changes.

6. Click **Actuators** and setup the actuators.

The screenshot shows the QGroundControl software interface for vehicle setup. The left sidebar contains icons for Summary, Firmware, Airframe, Sensors, Radio, Flight Modes, Power, Actuators (which is selected), Safety, PID Tuning, Camera, and Parameters.

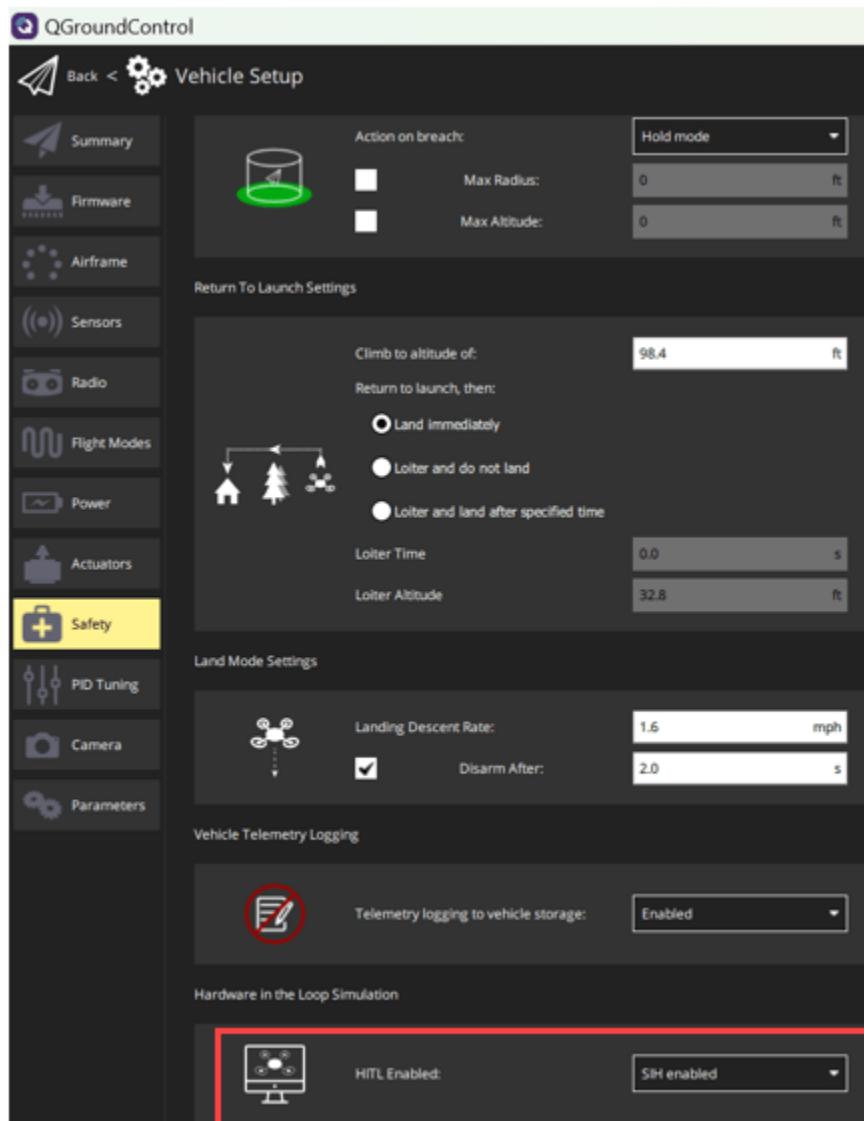
The main area is titled "Vehicle Setup" and "Actuators Setup". It specifies "Geometry: Fixed Wing". On the left, under "Actuators Setup", there is a section for "Motors" (set to 1) and "Control Surfaces" (set to 3). The "Control Surfaces" table includes columns for Type, Roll Torque, Pitch Torque, Yaw Torque, and Trim. The entries are:

Servo	Type	Roll Torque	Pitch Torque	Yaw Torque	Trim
Servo 1	Left Aileron	-0.50			0.00
Servo 2	Elevator		1.00		0.00
Servo 3	Rudder			1.00	0.00

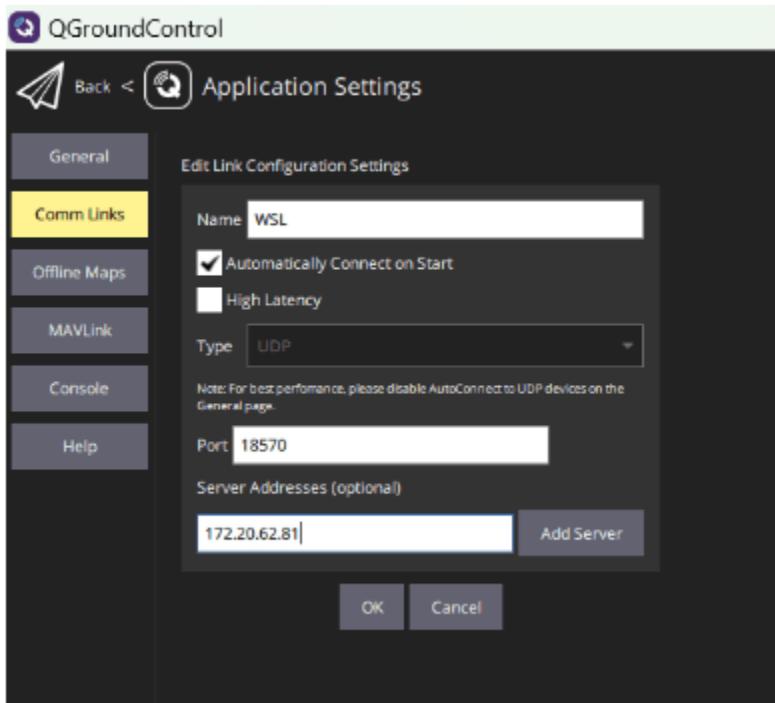
Below this is an "Actuator Testing" section with a note: "Propellers are removed - Enable sliders". It shows five sliders labeled "All Motors", "Motor 1", "Left Aileron", "Elevator", and "Rudder".

The right side of the screen shows the "Actuator Outputs" tab, which is currently set to "SIM". It lists 16 channels, each with a dropdown menu for "Function" and a switch for "Rev Range (for Servos)". All channels are currently set to "Disabled".

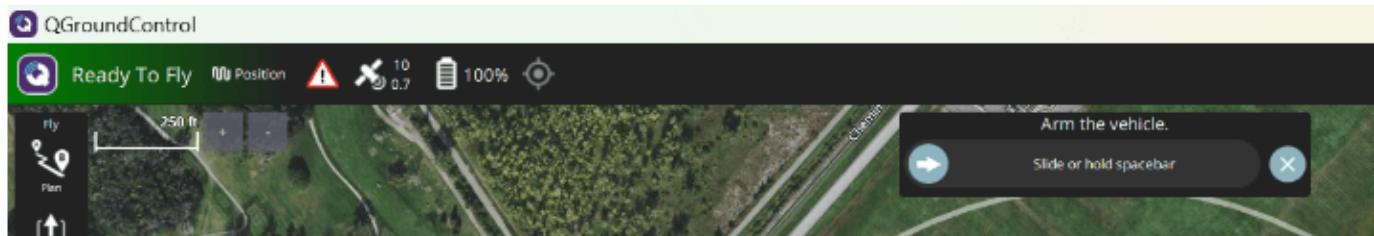
7. Click **Safety** and select SIH Enabled from **HITL Enabled** drop-down list.



8. Configure the communication links by clicking the gear icon in QGC and navigating to **Application Settings > Comm Links**. For more information see “Configuring QGC for Vehicle Visualization Without a Display”.



9. Arm the vehicle



10. Use the slider control to increase the throttle to its maximum. This gives full power to the fixed-wing plane.

11. Adjust Roll, Pitch, and Yaw:

- Roll: This controls the rotation of the plane around its front-to-back axis. Adjusting roll will tilt the plane to the left or right.
- Pitch: This controls the rotation around the side-to-side axis. Changing pitch will make the plane nose up or down.
- Yaw: This controls the rotation around the vertical axis. Altering yaw will turn the plane left or right.

Note: As this is manual control, flight might not be stable. Small adjustments can lead to significant changes in the plane's behavior. Make gradual inputs to better manage control.

By default, this example uses jMAVSIM for visualization. For more information, see "Deployment and Verification Using PX4 Host Target and jMAVSIM/Simulink".

However, you can opt to use Simulink for visualization instead. You have the option to use either:

- Simulation 3D UAV Vehicle Visualization (high fidelity)
- UAV Animation Visualization (low fidelity visualizer)

Additionally, you can select the fixed-wing option in the airframe settings. For more information, see “Visualize 3D Scenarios in Unreal Engine with PX4 Host Target Simulation” on page 1-390.

Related Topics

“Deployment and Verification Using PX4 Host Target and jMAVSim/Simulink”

Visualize 3D Scenarios in Unreal Engine with PX4 Host Target Simulation

This example shows how to use the UAV Toolbox Support Package for PX4® Autopilots to demonstrate 3D scenario Simulation with PX4 Host Target Simulation. For visualization, you can use this example model along with other models running in Simulation-In-Hardware (SIH) . This example is compatible with all existing SITL examples.

Prerequisites

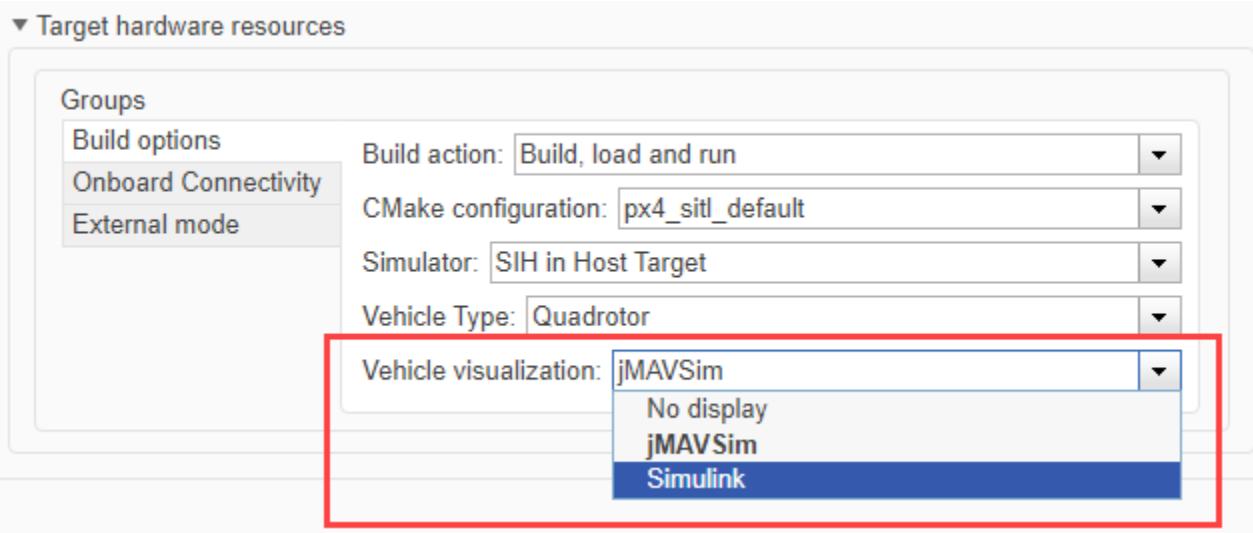
- If you are new to Simulink, watch the Simulink Quick Start video.

Model

For visualization, you can use this example model along with PX4 based flight controller models. It is recommended to open this model in a separate MATLAB instance to avoid slow performance and potential latency in visualization.

The controller model you use with this model must have Simulink selected as the vehicle visualization option. Verify the visualization option using these steps.

1. In the controller model, go to **Modeling > Model Settings** to open the Configuration Parameters dialog box.
2. Expand **Target Hardware resources** and in **Build Options**, ensure that you select **Simulink** for **Vehicle visualization** option.



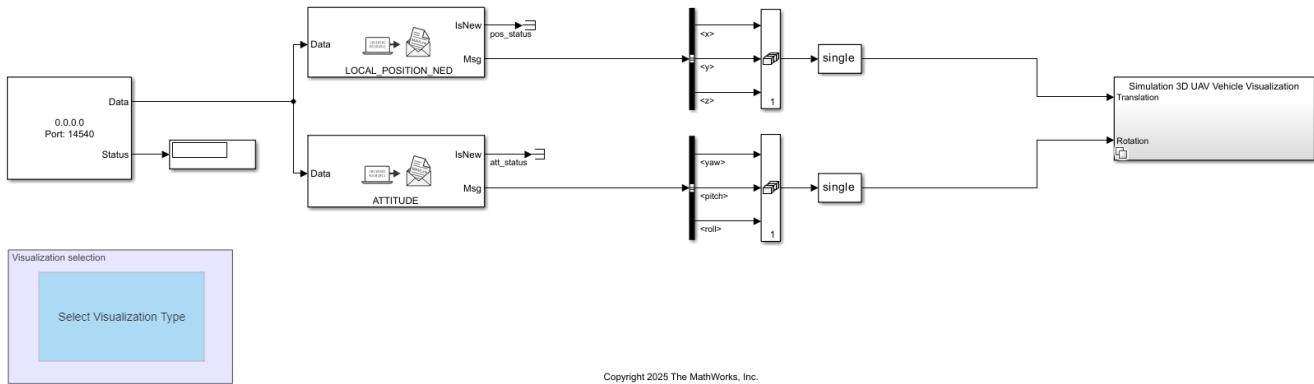
3. Click **Apply** and **OK**.

Open the `simulink_3DVisualization_HostTarget` model.

PX4 Host Target Flight visualization

Steps to select the visualization type:

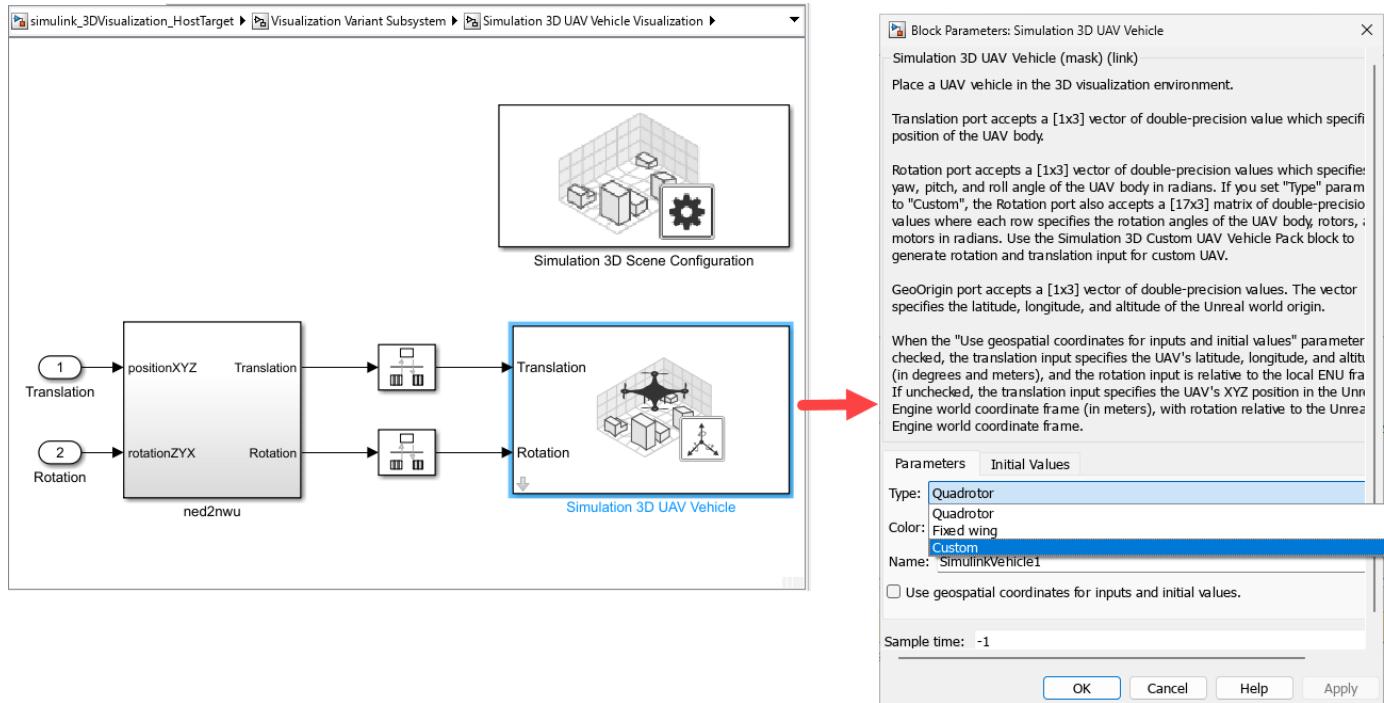
1. Click the [Select Visualization Type](#) button to select the visualization type.



Visualization for SIH in Host Target Workflow

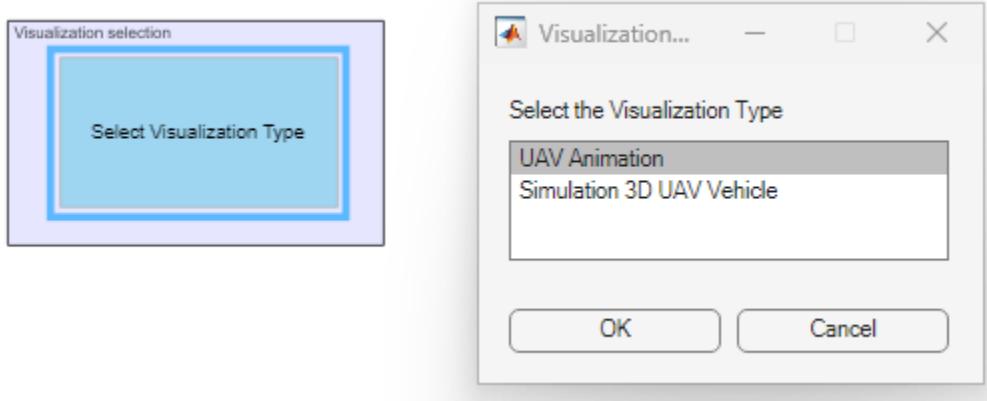
This section outlines the visualization options and configurations in the Host Target Workflow.

- Default Visualization: By default, the system uses **UAV Animation Visualization** with high fidelity.
- Switching Visualization: Use the **Visualization Selection** button to switch to the **UAV Animation Visualization**, which is a low fidelity visualizer.
- Model Configuration: The model utilizes the **MAVLink Deserializer** to obtain the vehicle's position in the NED (North-East-Down) frame and its attitude. This data is used solely for visualization purposes.
- Airframe Selection: The default airframe is set to a quadcopter. To change the airframe, in the Simulink model, go to `simulink_3DVisualization_HostTarget/Visualization Variant Subsystem/Simulation 3D UAV Vehicle Visualization` and then double-click the Simulation 3D UAV Vehicle block.

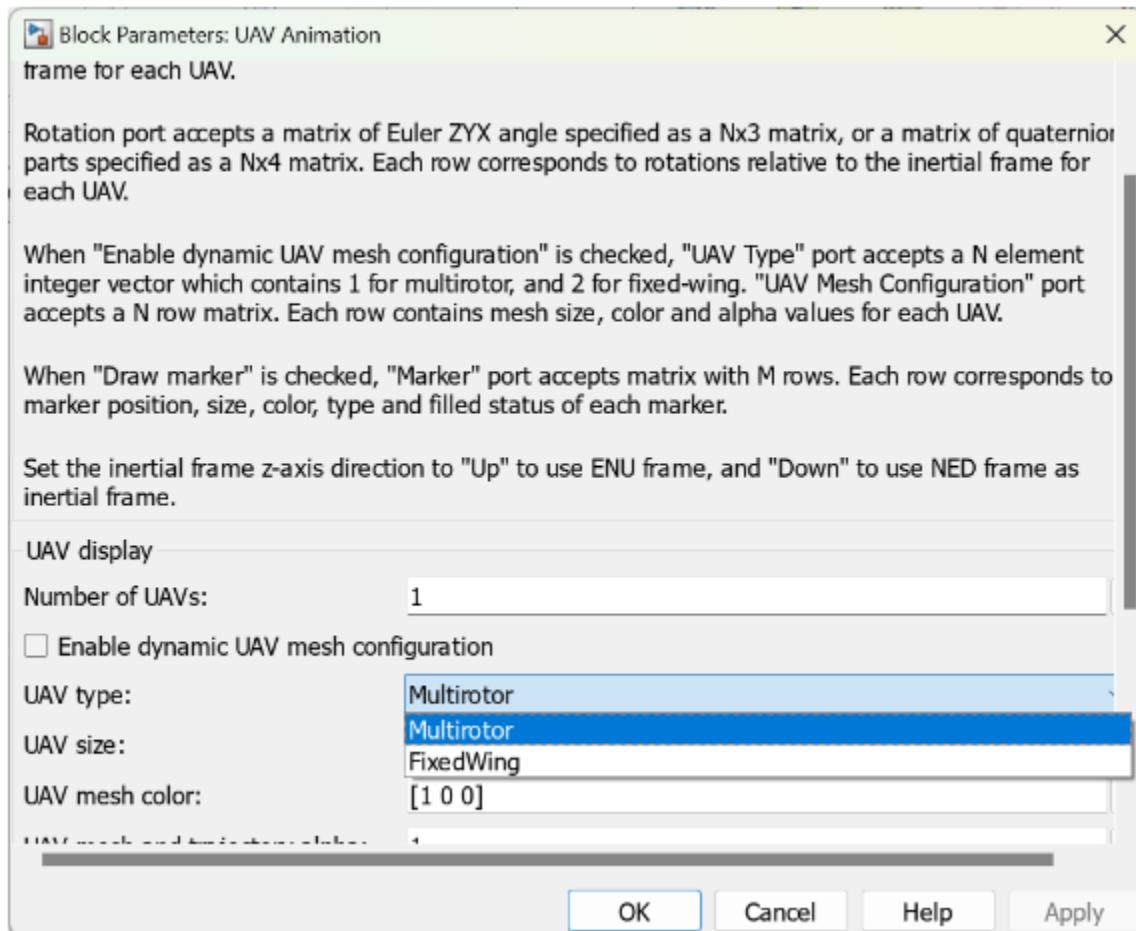


In the Simulation 3D UAV Vehicle block parameter dialogue, select the required vehicle.

Alternatively, use the **Visualization Selection** button to switch to UAV Animation Visualization, which is a low fidelity visualizer.



In the UAV Animation block parameter dialogue, select the required UAV type.



Monitor and Tune the Model

To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the Mode section, select **Run on board (External mode)** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

Related Topics

- “Deployment and Verification Using PX4 Host Target and jMAVSIM/Simulink”

Getting Started with PX4 Write Parameter Block for PX4 Autopilots

This example shows you how to use the PX4 Write Parameter block to write parameters values to a PX4-based flight controller. This allows you to dynamically change flight parameters during simulation and deployment, enabling real-time tuning and configuration.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens.

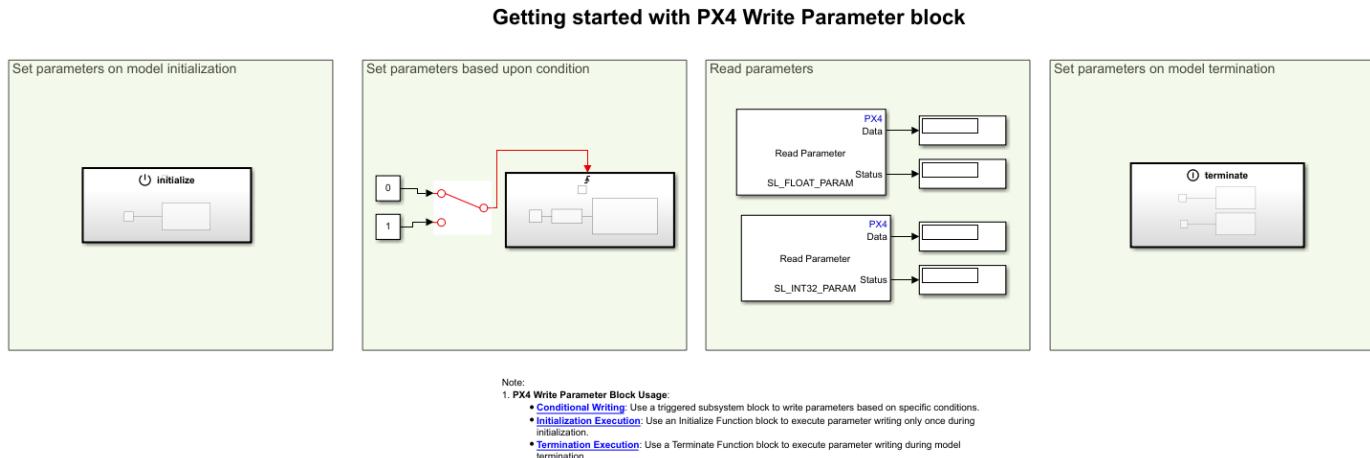
Required Third-Party Software

- QGroundControl (QGC)

Recommended: QGC version 4.3.0

Model

Open the px4WriteCustomParameter model.



In this example, the PX4 Write Parameter Block is used for:

Conditional Writing: To write parameters based on specific conditions, you can use a Triggered Subsystem block. This block allows you to execute parameter writing only when certain conditions are met.

- Inside the Triggered Subsystem, use a PX4 Write Parameter Block to write to SL_INT32_PARAM.

Initialization Execution: To execute parameter writing only once during initialization, use an Initialize Function block. This block ensures that the parameter writing occurs at the start of the model execution.

- Inside this block, use a PX4 Write Parameter Block to set SL_FLOAT_PARAM to 10.5.

Termination Execution: To execute parameter writing during model termination, use a Terminate Function block. This block allows you to perform cleanup tasks or final parameter writing.

- Inside this block, use PX4 Write Parameter Blocks to reset both SL_FLOAT_PARAM and SL_INT32_PARAM to their desired default values. For example, 0 or another specified reset value.

Note: By default, the parameters, SL_FLOAT_PARAM and SL_INT32_PARAM are added to the PX4 firmware during the hardware setup process. If they are not added, go through the hardware setup process again.

Monitor and Tune the Model

To view the changes done to the parameter in real-time, perform monitor and tune action. Perform these steps to perform signal monitoring and parameter tuning.

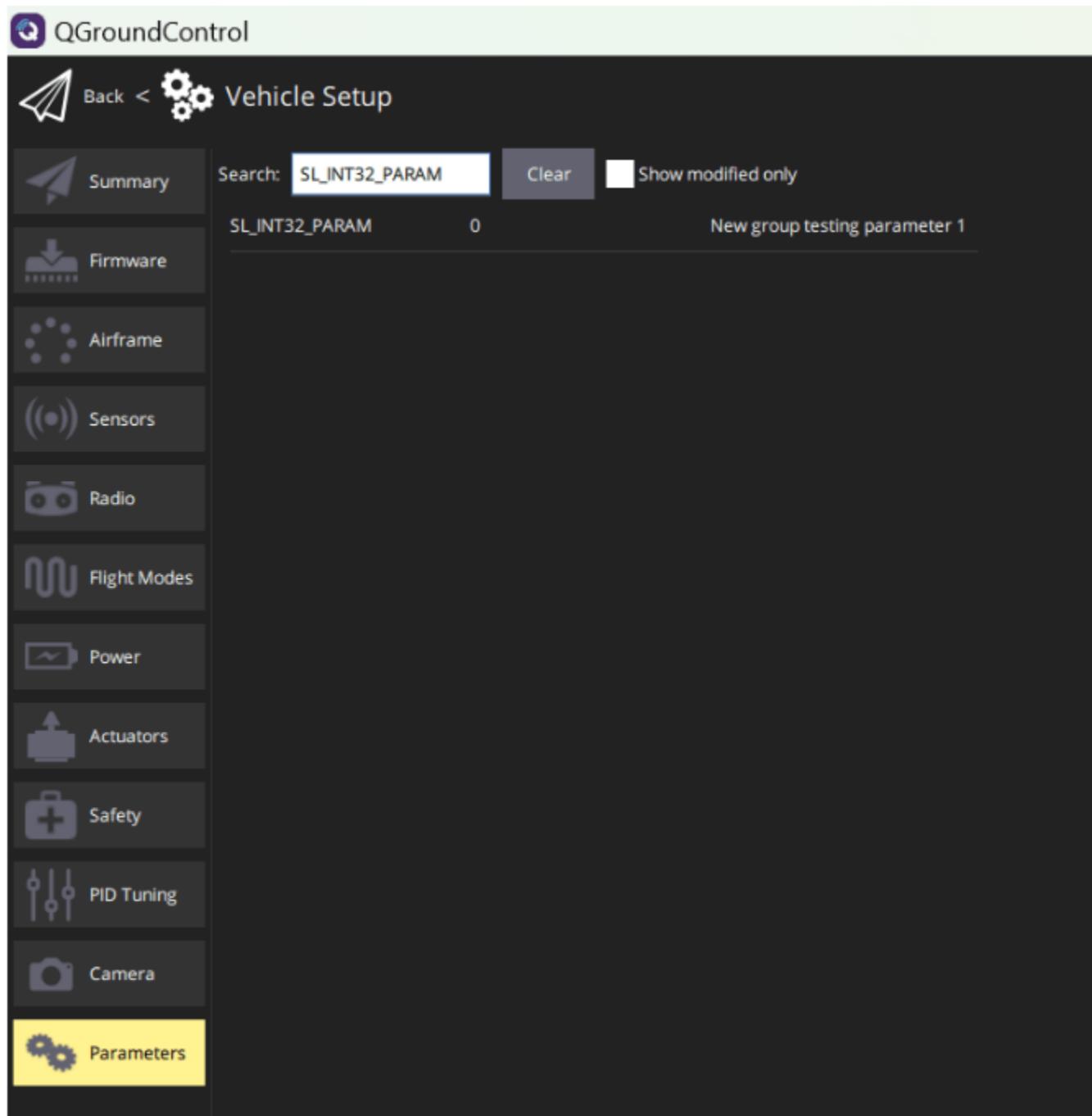
1. On the **Hardware** tab, in the Mode section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

Observe real-time parameter changes.

1. On initialization, the SL_FLOAT_PARAM is set to 10.5. Now use the Read Parameter block to continuously monitor the value of SL_FLOAT_PARAM.
2. Toggle the switch from 0 to 1 and observe the increment in SL_INT32_PARAM on each rising edge.
3. When stopping the model, verify that both parameters are reset to their specified values.

You can also verify the changes using the QGroundControl(QGC).

- Navigate to the **Parameters** section in QGC, and verify SL_FLOAT_PARAM and SL_INT32_PARAM parameters reflect the expected values during initialization, after switch operations, and after model termination.



See Also

[PX4 Write Parameter](#)

Actuator Control Using Write Thrust & Torque Blocks for PX4 Autopilots

This example shows how to use the PX4 Write Thrust & Torque block to command the actuators.

UAV Toolbox Support Package for PX4 Autopilots includes a Simulink blocks for writing thrust and torque setpoint value. In this example, you generate code for a Simulink model and deploy it to a PX4 Cube Orange, and thus spin the motors at 20% throttle.

Prerequisites

- If you are new to Simulink, watch the Simulink Quick Start video.
- For safety purpose, remove propellers from Cube Orange drone.
- Perform the initial “Setup and Configuration” of the support package using Hardware Setup screens. Select **PX4 Cube Orange** as the PX4 Autopilot board in Hardware Setup screens.
- Perform sensor calibration and radio setup.

Required Hardware

To run this example, you will need the following hardware:

- Cube Orange
- Micro USB (Universal Serial Bus) type-B cable

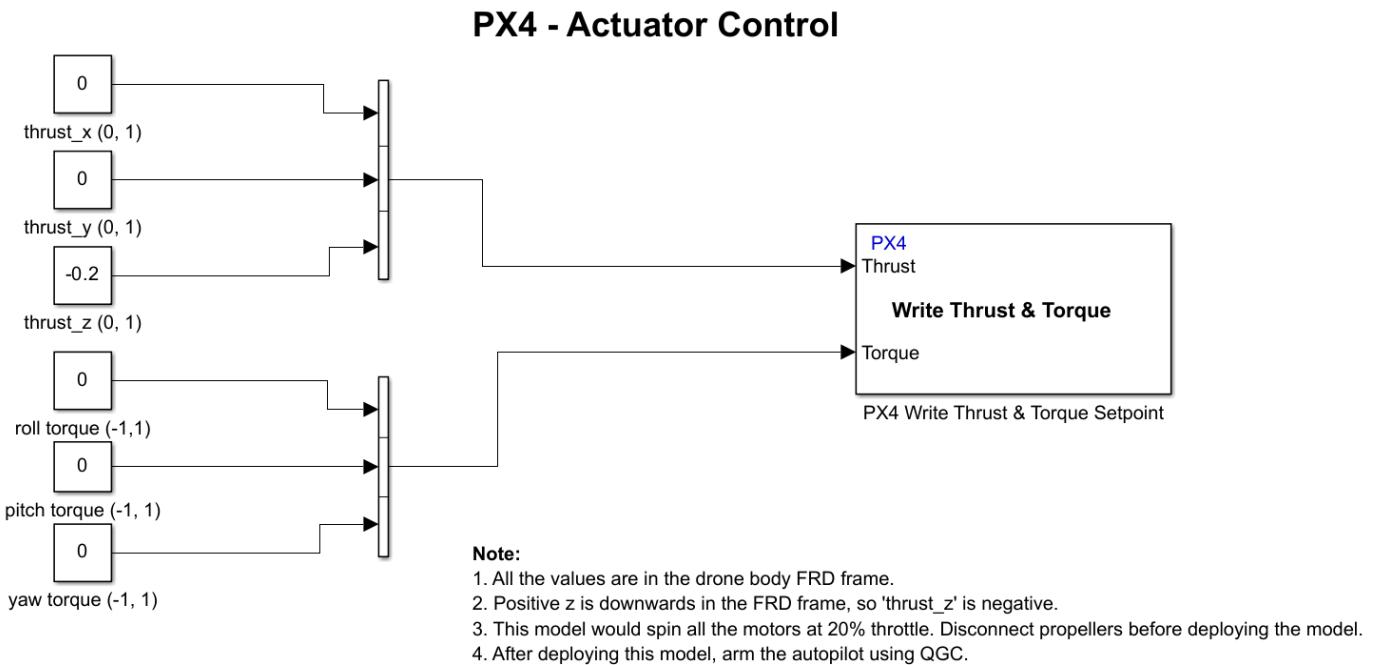
Required Third-Party Software

- QGroundControl (QGC)

Recommended: QGC version 4.3.0

Model

To get started, open px4demo_actuatorControl model.



Configure and Run the Model

In this task, you will configure and run the pre-defined Simulink model (*px4demo_actuatorControl*) and deploy it to the PX4 Cube Orange board.

1. In the **Modeling** tab, click **Model Settings**.
2. In the Configuration Parameters dialog box, navigate to the **Hardware Implementation** pane.
3. If you are using a hardware board other than PX4 Cube Orange, then select appropriate hardware board.
4. In the **Simulation** tab, set the *Stop Time* to **inf**.
5. Connect the USB cable from the Cube Orange to the host computer.
6. On the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Build, Deploy & Start**. The model is deployed to the PX4 hardware board.

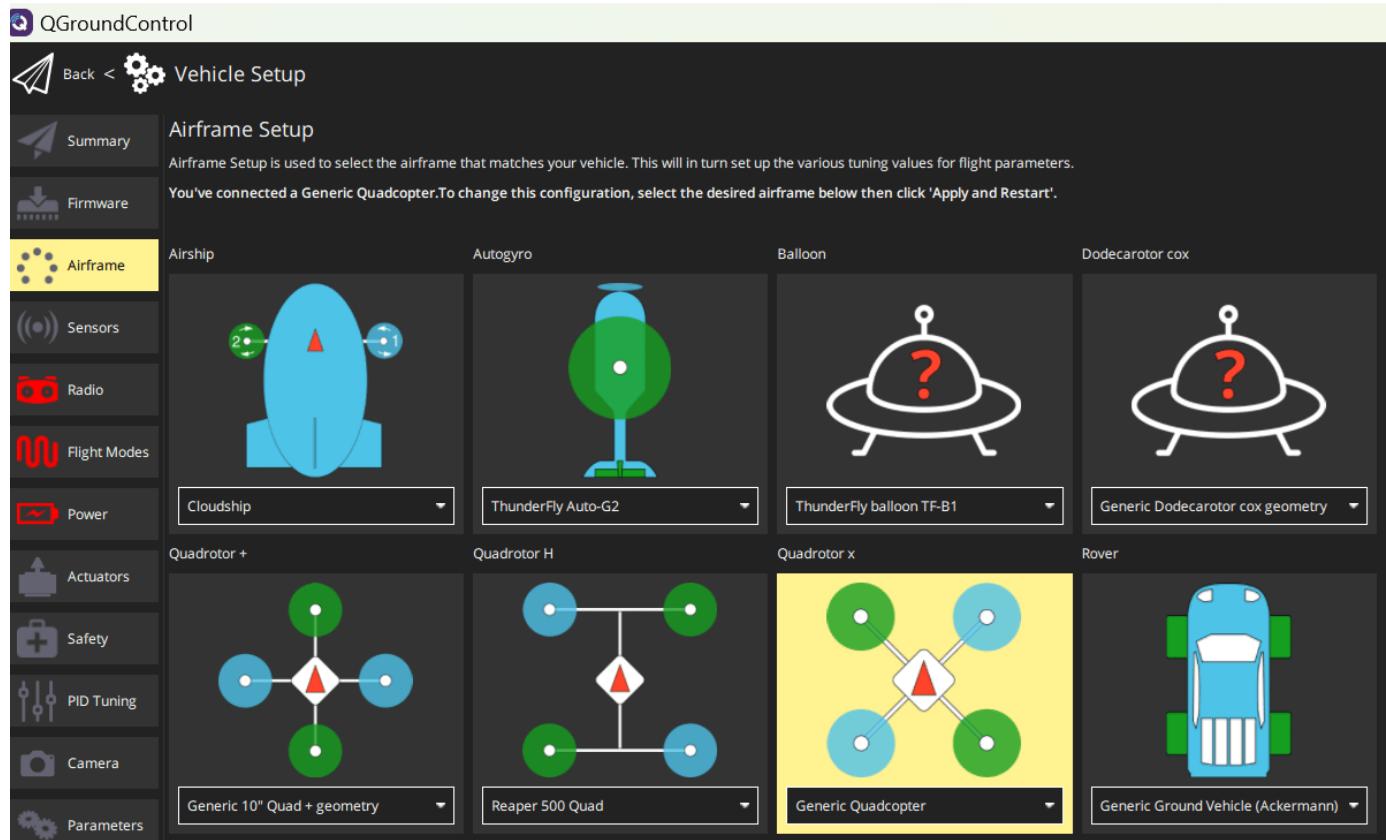
This deploys the controller code on the Cube Orange and launches the QGroundControl (QGC).

Note: If you are using Ubuntu, QGC might not launch automatically. To launch QGC, open Terminal and go to the location where QGC is downloaded and run the following command:

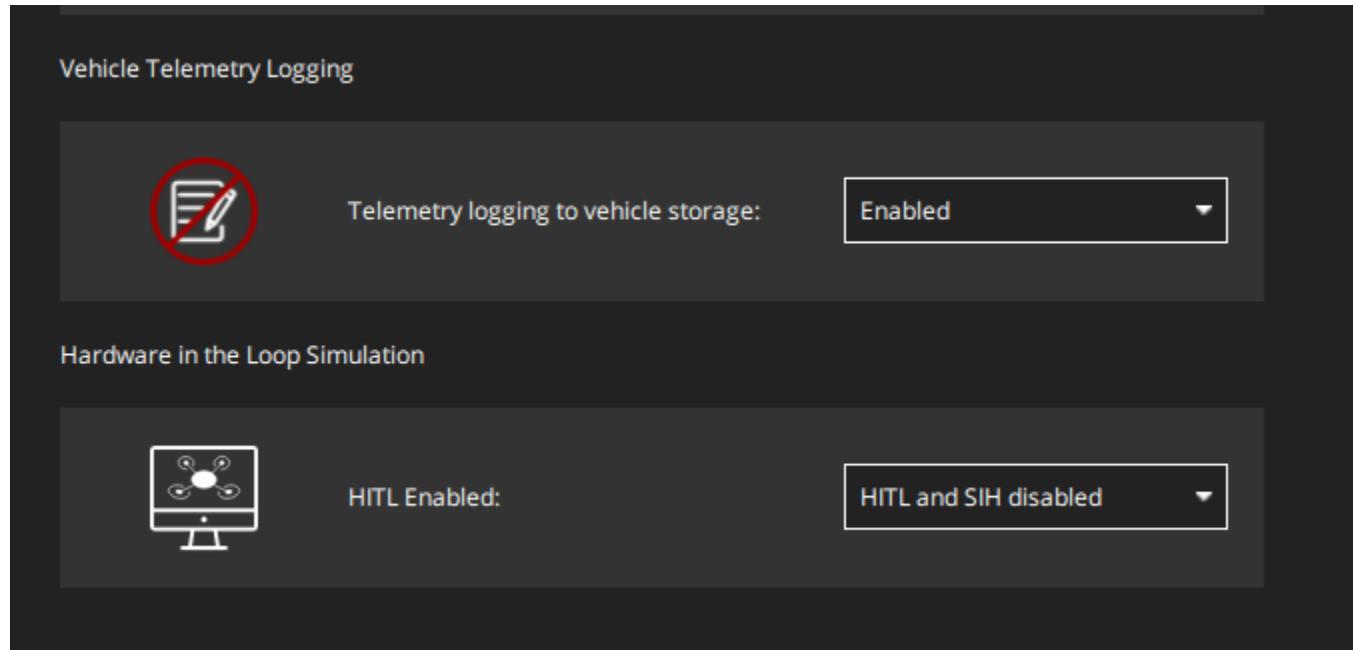
```
./QGroundControl.AppImage
```

Configure QGroundControl and Fly Cube Orange

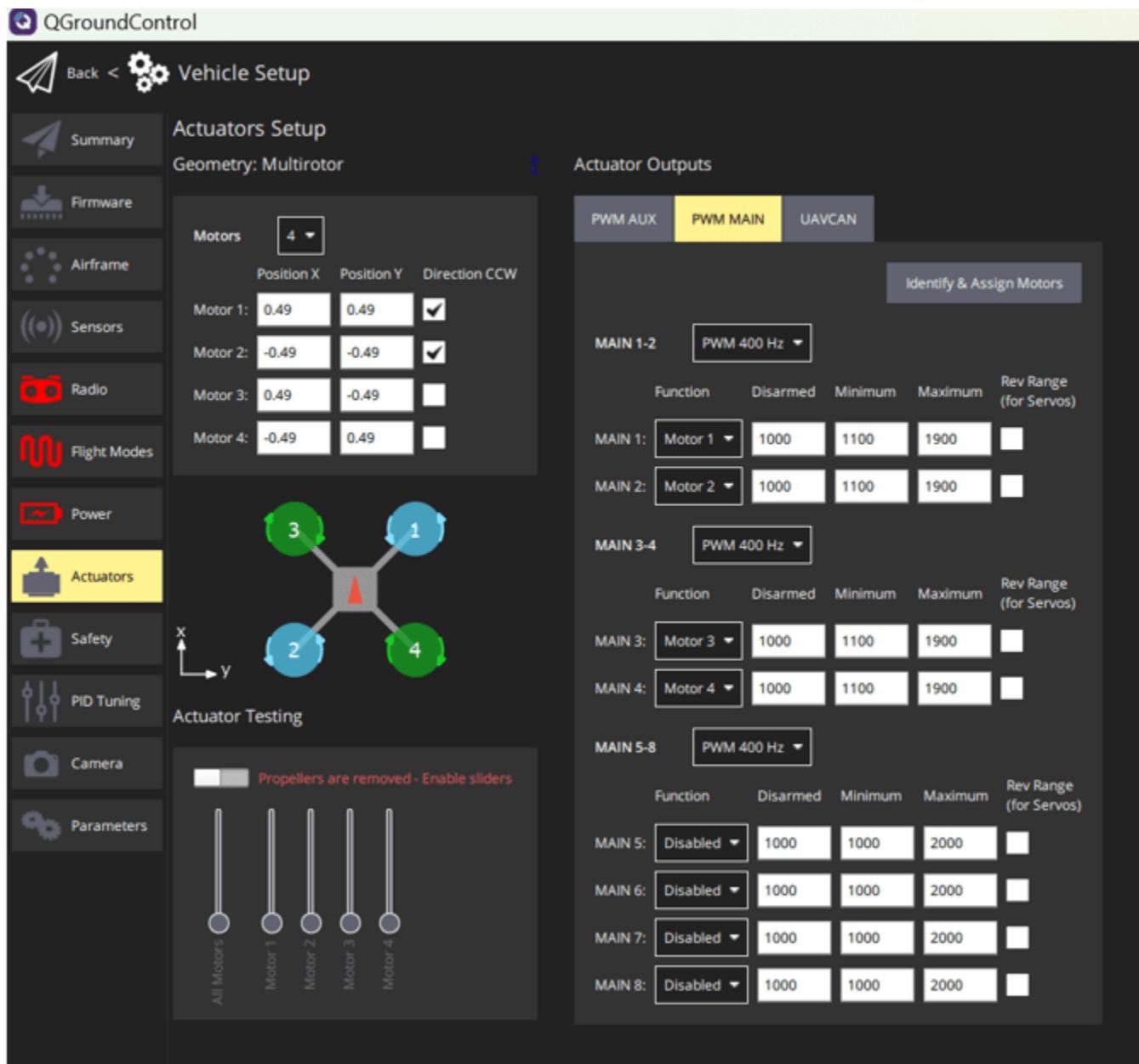
1. In the QGC, select Airframe.



- Navigate to **Setup > Airframes**.
 - Select Generic Standard Plane. Click **Apply and Restart** on top-right of the **Airframe Setup** page.
 - Restart QGC to allow it to connect to the board and apply the new settings.
2. Navigate to **Setup > Safety** section.

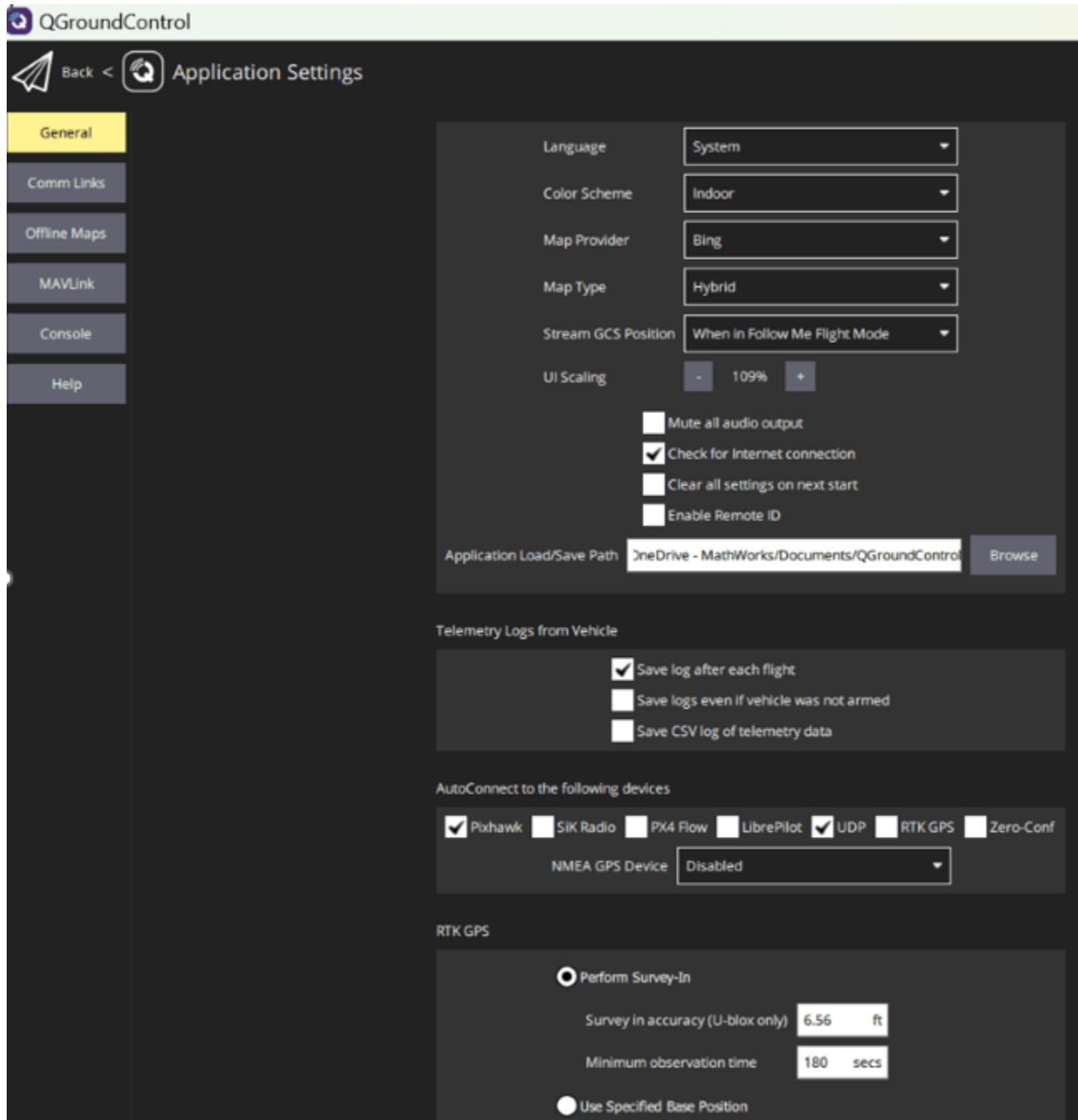


- Select **HITL and SIH disabled** from the **HITL Enabled** list.
3. Navigate to Actuators tab and identify and assign the main motors.

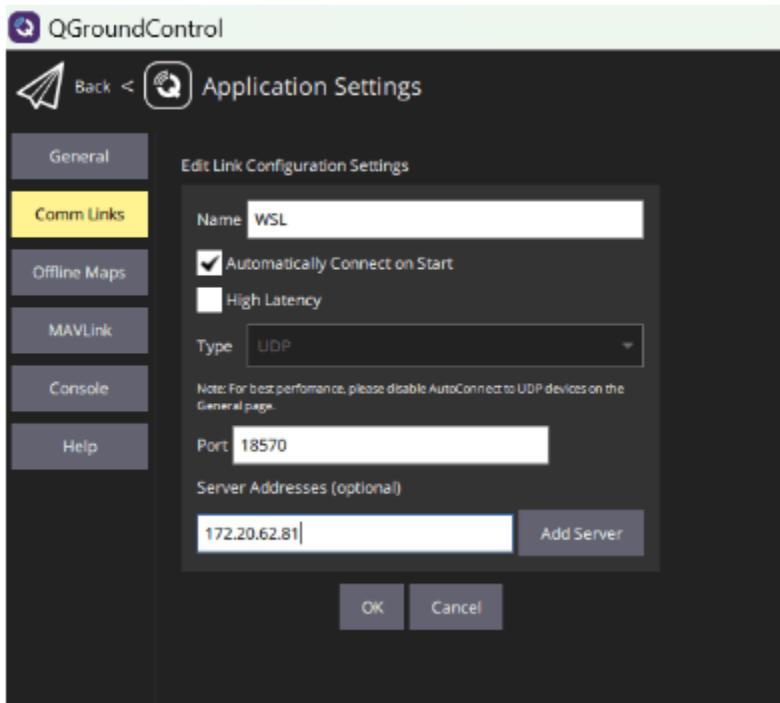


4. In **General** tab for AutoConnect options, select **Pixhawk** and **UDP**.

1 Blocks

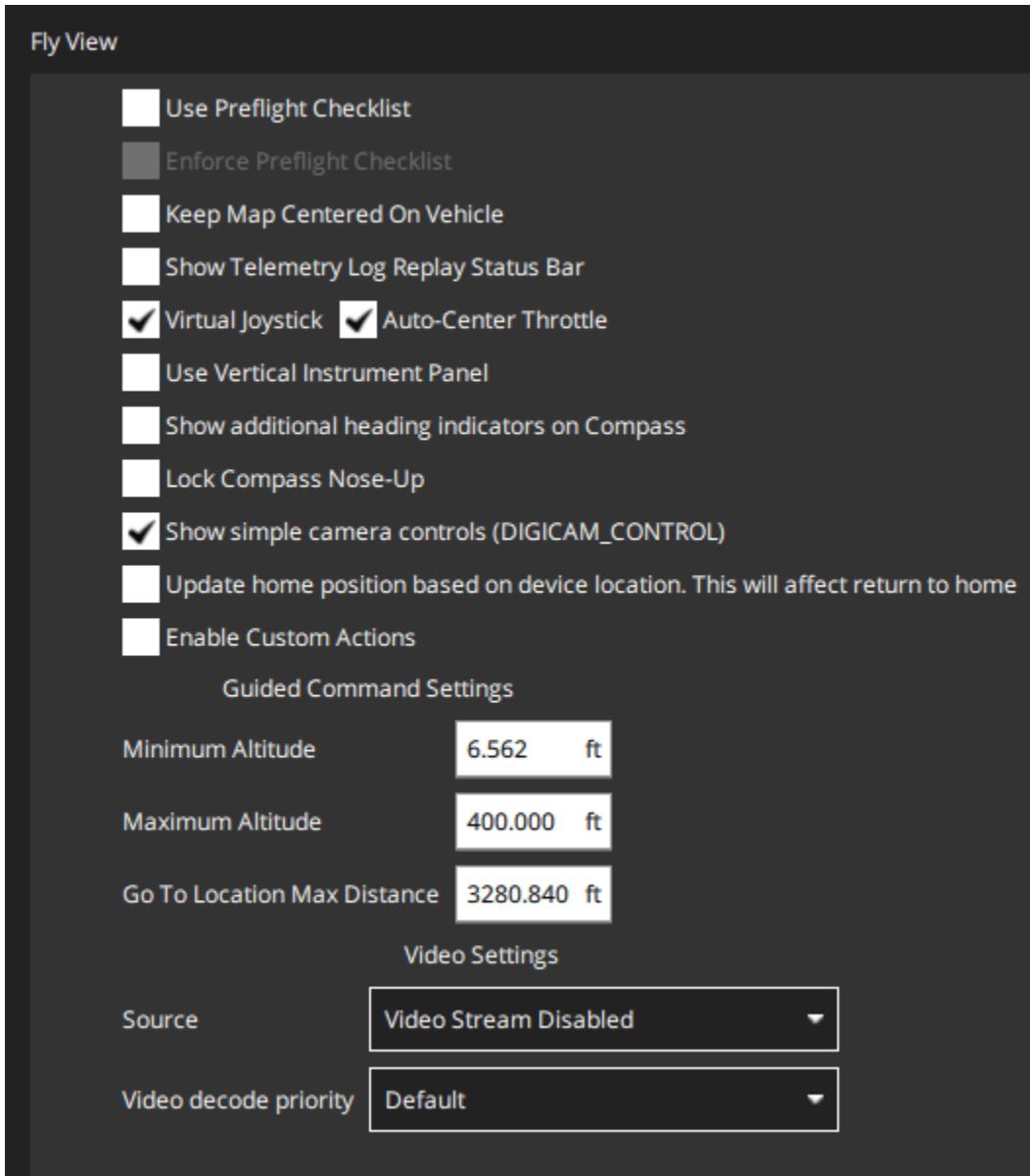


5. Click **Comm Links** and configure the communication links. For more information see “Configuring QGC for Vehicle Visualization Without a Display”.



6. Select Virtual Joystick.

- In the **General** tab, from **Fly View** of the settings menu, select **Virtual Joystick** option.

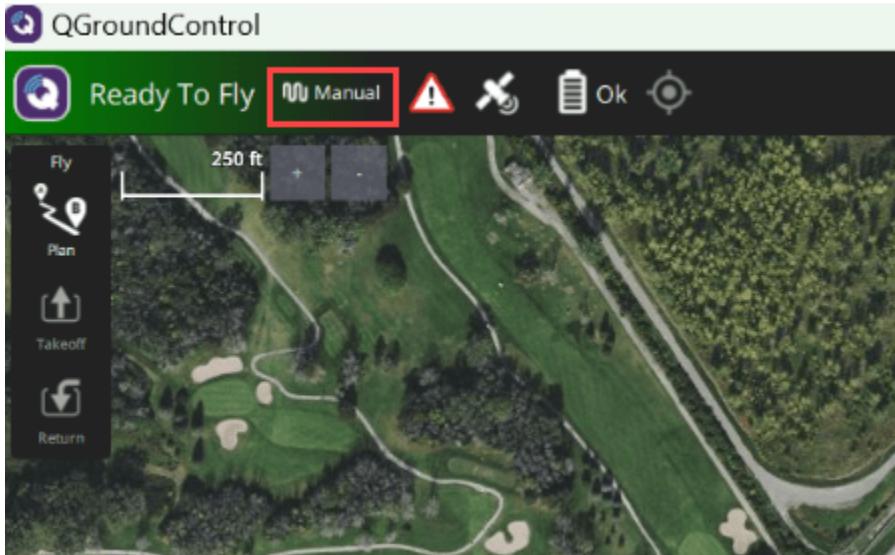


7. Set the following parameters.

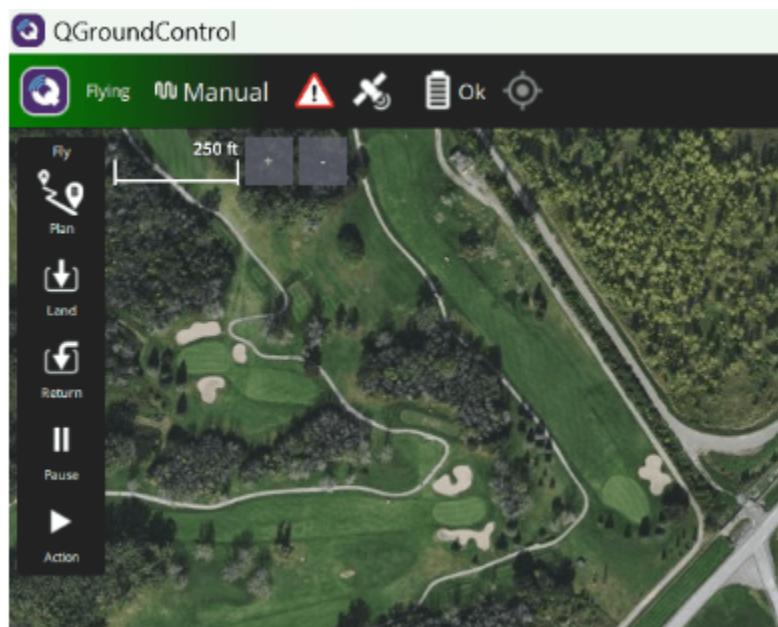
- Disable the parameter COM_ARM_SWISBTN for arming/disarming triggers on switch transition.
- Enable the parameter MAN_ARM_GESTURE for arm stick gesture.
- Disable the parameter COM_ARM_HFLT_CHK for FMU SD card hardfault detection check.

8. Restart the QGC to apply the changes.

- Change the flight mode to Manual in the QGC.



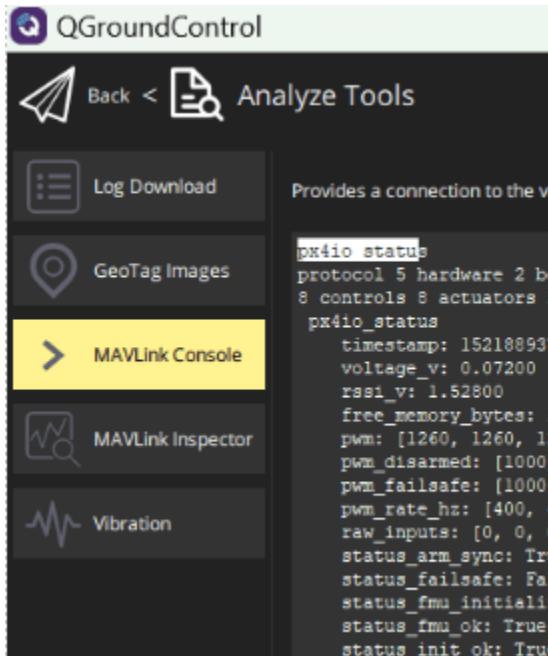
- Drag the left joystick to the bottom-right corner for a couple of seconds to force arm the UAV.



All four motors now spins at 20% throttle.

Validate Actuator Values

1. In the QGC, go to **Analyze Tools** and navigate to **MAVLink Console**.



2. Enter the px4 io status command in the console.

```
nsh > px4io status
```

This action updates the actuator values displayed from Simulink. Since the actuator value of 0.2 was written, observe the value updated to 1260.

```

IMU heater off
INFO [mixer_module] Param prefix: PWM_MAIN
control latency: 98368 events, 8722435145us elapsed, 88671.47us avg, min 271us max 195971us 61632.199us rms
INFO [mixer_module] Switched to rate_ctrl work queue
Channel Configuration:
Channel 0: func: 101, value: 1260, failsafe: 1000, disarmed: 1000, min: 1100, max: 1900
Channel 1: func: 102, value: 1260, failsafe: 1000, disarmed: 1000, min: 1100, max: 1900
Channel 2: func: 103, value: 1260, failsafe: 1000, disarmed: 1000, min: 1100, max: 1900
Channel 3: func: 104, value: 1260, failsafe: 1000, disarmed: 1000, min: 1100, max: 1900
Channel 4: func: 0, value: 1000, failsafe: 1000, disarmed: 1000, min: 1000, max: 2000
Channel 5: func: 0, value: 1000, failsafe: 1000, disarmed: 1000, min: 1000, max: 2000
Channel 6: func: 0, value: 1000, failsafe: 1000, disarmed: 1000, min: 1000, max: 2000
Channel 7: func: 0, value: 1000, failsafe: 1000, disarmed: 1000, min: 1000, max: 2000
nsh>
  
```

This value is obtained using a minimum value of 1100 and a maximum value of 1960. To find the updated value, multiply 0.2 by the difference between the maximum and minimum values (which is 800) and then add the result to the minimum value of 1100. Therefore, you calculate the updated value as ((0.2 \times 800) + 1100 = 1260).

See Also

"Getting Started with Actuator Control over PWM" on page 1-362

"Getting Started with Actuator Control over UAVCAN/DroneCAN" on page 1-374

PX4 Hardware-in-the-Loop (HITL) Simulation and Visualization with VTOL UAV

This example series demonstrates how to use the UAV Toolbox Support Package for PX4 Autopilots to verify the VTOL UAV controller that you developed using the VTOL UAV Controller Template through Hardware-in-the-Loop (HITL) simulation on PX4. Additionally, this series includes visualizing the VTOL UAV mission in an urban environment by using Unreal Engine®.

- 1** “PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Tilt-Rotor Plant in Simulink” on page 1-334
- 2** “Visualize PX4 Hardware-in-the-Loop (HITL) Simulation with VTOL UAV Over Urban Environment” on page 1-344

Pin Mapping for Pixhawk Series Controller Boards

Index Numbers for Analog Channels on Pixhawk Series Controller Boards

The PX4 Analog Input block in UAV Toolbox Support Package for PX4 Autopilots supports reading of analog voltage values from the sensors connected to the Pixhawk Series controller board. This block outputs a 1-by-12 array that contains single values of analog voltage at the corresponding channel numbers. Because the analog pins on each Pixhawk Series controller board varies according to the FMU version that they support, it is necessary to understand the correct channel numbers represented in the output signal from the PX4 Analog Input block.

The supported FMU version defines all the available channels, even though the analog channels that are exposed on a PX4 flight controller depends on the manufacturer.

Available analog input channels based on FMU versions

FMU Version	Available Analog Input Channels
FMUv2	2, 3, 4, 10, 11, 12, 13, 14, 15
FMUv3	2, 3, 4, 10, 11, 12, 13, 14, 15
FMUv4	2, 3, 4, 10, 11, 12, 13, 14
FMUv5	0, 1, 2, 3, 4, 8, 10, 11, 12, 13, 14

The following table contains the index numbers that you can use to extract signal values, based on the channel to which the analog input device is connected. The table lists all the target hardware that is tested for this release of UAV Toolbox Support Package for PX4 Autopilots.

Index numbers for supported Pixhawk series boards

Target Hardware supported for R2018b	Analog Input Channels available on the corresponding FMU version	Analog Input Channels exposed on the board	Index of exposed channels in the 1-by-12 array (output signal from PX4 Analog Input block)
Pixhawk 1	FMUv2: 2, 3, 4, 10, 11, 12, 13, 14, 15	13, 14, 15	7, 8, 9
Cube (Pixhawk 2)	FMUv3: 2, 3, 4, 10, 11, 12, 13, 14, 15	15	9
Pixracer	FMUv4: 2, 3, 4, 10, 11, 12, 13, 14	No ADC channels are exposed on the board	--
Pixhawk 4	FMUv5: 0, 1, 2, 3, 4, 8, 10, 11, 12, 13, 14	4, 14	5, 11

In the Simulink model, the 1-by-12 array output from the PX4 Analog Input block can be connected to a Selector block, which extracts the signals from the array, based on index numbers that you specify.

Port Mapping for Serial Ports

Serial Port Names and Corresponding Labels on PX4 Flight Controller Boards

The PX4 Serial Receive and PX4 Serial Transmit block in UAV Toolbox Support Package for PX4 Autopilots support sending/receiving of data using the serial ports on the PX4 flight controller board.

This section lists the mapping of the different port labels on the PX4 flight controller board to the UART/USART port numbers that you can select during the configuration of the Simulink model.

Port Numbers for Cube Blue H7

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttys0
TELEM 2	/dev/ttys1
GPS1	/dev/ttys2
GPS2	/dev/ttys5

Port Numbers for Cube Orange

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttys0
TELEM 2	/dev/ttys1
GPS1	/dev/ttys2
GPS2	/dev/ttys5

Port Numbers for Cube Orange Plus

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttys0
TELEM 2	/dev/ttys1
GPS1	/dev/ttys2
CONSOLE/ADSB-IN	/dev/ttys4
GPS2	/dev/ttys5

Port Numbers for CUAV X7+

Labels	UART/USART port number
GPS1	/dev/ttys0
TELEM1	/dev/ttys1
GPS2	/dev/ttys2
TELEM2	/dev/ttys3

Port Numbers for Pixhawk 6c

Labels	UART/USART port number
USB	/dev/ttyACM0
GPS1	/dev/ttyS0
TELEM3	/dev/ttyS1
Debug Console	/dev/ttyS2
TELEM2	/dev/ttyS3
PX4IO	/dev/ttyS4
TELEM1	/dev/ttyS5
GPS2	/dev/ttyS6

Port Numbers for Pixhawk 6x

Labels	UART/USART port number
USB	/dev/ttyACM0
GPS1	/dev/ttyS0
TELEM3	/dev/ttyS1
Debug Console	/dev/ttyS2
UART4 & I2C	/dev/ttyS3
TELEM2	/dev/ttyS4
PX4IO/RC	/dev/ttyS5
TELEM1	/dev/ttyS6
GPS2	/dev/ttyS7

Port Numbers for Uvify IFO-S

Physical Label	QGC Port	UART/USART port number
USB	USB	/dev/ttyACM0
CONN1	TELEM 1	/dev/ttyS1
Internal (telemetry)	TELEM 2	/dev/ttyS2
GPS	GPS1	/dev/ttyS3
SERIAL4	TELEM 3	/dev/ttyS6

Port Numbers for Pixhawk 1

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttyS1
TELEM 2	/dev/ttyS2
GPS	/dev/ttyS3
Serial 4	/dev/ttyS6
Serial 5 (NSH port)	/dev/ttyS5

Port Numbers for Pixhawk 2.1 (Cube)

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttyS1
TELEM 2	/dev/ttyS2
GPS1	/dev/ttyS3
GPS2	/dev/ttyS6
Serial 5 (NSH port)	/dev/ttyS5

Port Numbers for Pixhawk 4

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttyS1
TELEM 2	/dev/ttyS2
GPS	/dev/ttyS0
UART	/dev/ttyS4

Port Numbers for Pixracer

Labels	UART/USART port number
USB	/dev/ttyACM0
TELEM 1	/dev/ttyS1
TELEM 2	/dev/ttyS2
Serial 5 (NSH port)	/dev/ttyS5

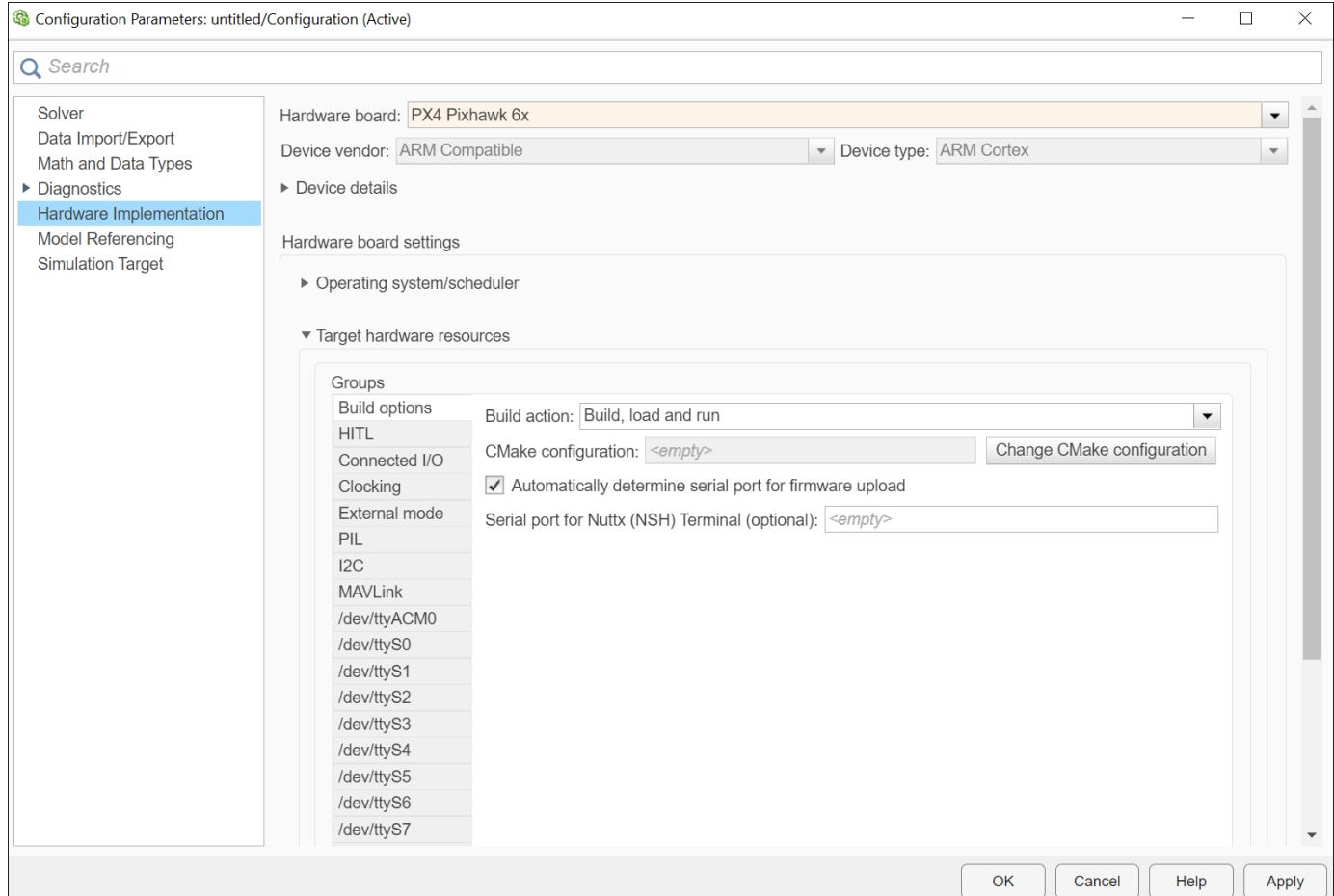
Coder Target Context Sensitive Help

Model Configuration Parameters for PX4 Flight Controller

In this section...

- "Hardware Implementation Pane Overview" on page 4-3
- "Build Options" on page 4-4
- "HITL" on page 4-5
- "Connected I/O" on page 4-5
- "External mode" on page 4-6
- "Onboard Connectivity" on page 4-7
- "Clocking" on page 4-7
- "PIL" on page 4-7
- "I2C" on page 4-8
- "CAN" on page 4-8
- "MAVLink" on page 4-9
- "/dev/tty" on page 4-9
- "Overrun Action" on page 4-10

Hardware Implementation Pane Overview



Default Hardware Implementation Pane

Configure PX4 flight controller to run Simulink models.

- 1 In the Simulink Editor, select **Simulation > Model Configuration Parameters**.
- 2 In the Configuration Parameter dialog box, click **Hardware Implementation**.
- 3 Set the **Hardware board** parameter to one of these, based on the board connected to the host computer:
 - PX4 Cube Blue H7
 - PX4 Cube Orange
 - PX4 Cube Orange Plus
 - PX4 Pixhawk 6c
 - PX4 CUAV X7+
 - PX4 Pixhawk 6x
 - PX4 Pixhawk 1
 - PX4 Pixhawk 2.1 (Cube)

- PX4 Pixhawk 4
- Pixhawk Series
- PX4 Pixracer
- PX4 Uvify IFO-S

For more information, see “Supported PX4 Autopilots”.

Additionally, you can also select **PX4 Host Target** as the Hardware board to perform only simulation using the jMAVSim simulator.

- 4 The parameter values under **Hardware board settings** are automatically populated to their default values.
You can optionally adjust these parameters for your particular use case.
- 5 To apply the changes, click **Apply**.

For more information on selecting a hardware support package and general configuration settings, see “Hardware Implementation Pane” (Simulink).

Build Options

Parameter	Description	Default Value
“Build action” on page 4-11	Option to specify whether you want only the build, or the build, load, and run actions during code generation	Build, load, and run
“CMake configuration” on page 4-11	Change the CMake configuration file that is used to build the firmware.	<empty>
“Automatically determine serial port for firmware upload” on page 4-11	Enables the automatic detection of the serial port for firmware upload, based on the hardware connections Note This parameter does not appear if you select the Hardware board as PX4 Host Target .	on
“Serial port for firmware upload” on page 4-11	Select the serial port of the host computer for firmware upload Note This parameter does not appear if you select the Hardware board as PX4 Host Target .	<empty>
“Serial port for NuttX (NSH) Terminal” on page 4-12	Select the serial port of the host computer for Nuttx (NSH) terminal	<empty>

Parameter	Description	Default Value
"Allow flashing FMUv3 CMake configuration on Pixhawk 1" on page 4-12	(Applicable only if you selected PX4 Pixhawk1 or PX4 Pixhawk Series as the Hardware Board) Determines if you need to allow flashing FMUv3 configuration on Pixhawk 1	off
"Simulator" on page 4-12	Select the simulator to be used Note This parameter appears only if you select the Hardware board as PX4 Host Target .	jMAVSim

HITL

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value
"Enable HITL Mode" on page 4-13	Enables the usage of HITL mode	off
"Simulator" on page 4-13	Simulation engine containing the physics for the UAV Dynamics	jMAVSim

Connected I/O

Note This tab does not appear if you select **PX4 Pixhawk Series** as the Hardware board.

Parameter	Description	Default Value
"Hardware board Serial Port" on page 4-14	Serial port number on the PX4 flight controller board for Connected I/O communication Note This parameter does not appear if you select PX4 Host Target as the Hardware board.	/dev/ttyACM0

Parameter	Description	Default Value
"Use the same host serial port for Connected I/O as used for firmware upload" on page 4-14	<p>Sets the same host serial port for Connected I/O as the one used for firmware upload</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	on
"Host Serial Port" on page 4-14	<p>Serial port number on the host computer for External mode communication</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	<empty>

External mode

Parameter	Description	Default Value
"Communication interface" on page 4-16	Communication interface that is used to exchange data between host computer and PX4 flight controller board.	XCP on TCP/IP — For PX4 Host Target XCP on Serial — For all other PX4 boards
"Use the same host serial port for External mode as used for firmware upload" on page 4-16	<p>Sets the same host serial port for External mode as the one used for firmware upload</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	on
"Host Serial Port" on page 4-16	<p>Serial port number on the host computer for External mode communication</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	<empty>
"Hardware board Serial Port" on page 4-16	<p>Serial port number on the PX4 flight controller board for External mode communication</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	/dev/tty/ACM0

Parameter	Description	Default Value
"Set logging buffer size automatically" on page 4-17	<p>Automatically set the number of bytes to preallocate for the buffer in the hardware during simulation.</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	on
"Logging buffer size (in bytes)" on page 4-17	<p>Specify the memory buffer size for XCP-based External mode simulation.</p> <p>Note This parameter does not appear if you select PX4 Host Target as the Hardware board.</p>	1024
"Verbose" on page 4-18	View External mode execution progress and updates	on

Onboard Connectivity

Note This tab appears only if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value
"Onboard Connectivity" on page 4-19	IP address of onboard computer	127.0.0.1

Clocking

Note This tab does not appear if you select the Hardware board as **PX4 Host Target**.

Parameter	Description	Default Value
"CPU Clock (MHz)" on page 4-15	The CPU clock frequency in MHz	168

PIL

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value
"Hardware board serial port" on page 4-20	Serial port number on the hardware board for PIL communication Note This parameter does not appear if you select PX4 Host Target as the Hardware board.	/dev/ttyACM0
"Use the same host serial port for PIL as used for firmware upload (mentioned under 'Build Options')" on page 4-20	Enables the usage of same host serial port for PIL as the one used for firmware upload Note This parameter does not appear if you select PX4 Host Target as the Hardware board.	on
"Host Serial Port" on page 4-20	Serial port number on the host computer for PIL communication	COM6

I2C

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value in KHz
"Bus 1 speed (KHz)" on page 4-21	Defines the rate of data communication between the peripherals connected by the I2C Bus 1.	100
"Bus 2 speed (KHz)" on page 4-21	Defines the rate of data communication between the peripherals connected by the I2C Bus 2.	100
"Bus 3 speed (KHz)" on page 4-21	Defines the rate of data communication between the peripherals connected by the I2C Bus 3.	100
"Bus 4 speed (KHz)" on page 4-21	Defines the rate of data communication between the peripherals connected by the I2C Bus 4.	100

CAN

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value
"CAN Port" on page 4-22	Defines the CAN port used on the host computer.	CAN
"Baud rate (in bits/s)" on page 4-22	Defines the rate at which data is transferred over CAN network. (in bits/s).	500000
"Test mode" on page 4-22	Specifies if the CAN test mode is enabled or not.	Off

MAVLink

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value
"Enable MAVLink on /dev/ttyACM0" on page 4-24	<p>Enables the usage of MAVLink protocol</p> <p>Note If you enable MAVLink, you must also change the Hardware board serial port value in External mode and PIL tabs. This is because these tabs use /dev/ttyACM0 as the default serial port.</p>	off

/dev/tty

Note All the /dev/tty tabs are available for configuration if you select any Pixhawk Series board as the Hardware board.

Note None of the /dev/tty tabs are displayed if you select **PX4 Host Target** as the Hardware board.

Parameter	Description	Default Value
"Baud rate" on page 4-25	Defines the rate at which data is transferred over a serial line	3000000
"Parity" on page 4-25	Determines whether the UART or USART port generates and checks for even parity or odd parity	None
"Stop bits" on page 4-25	Sets the number of stop bits to indicate end of a packet	1

Parameter	Description	Default Value
"Enable hardware flow control" on page 4-25	Enables hardware flow control using RTS and CTS pins	<code>off</code>
"View port map" on page 4-25	View the port mapping of UART or USART port on the PX4 flight controller	

Overrun Action

Parameter	Description	Default Value
"Shutdown PX4 Autopilot upon overrun" on page 4-30	Enables shutdown of PX4 Autopilot if task overrun occurs	<code>off</code>

Build options

Defines how Simulink Coder software responds when you press **Ctrl+B** to build your model.

Build action

Default: Build, load, and run

Build, load, and run

With this option, pressing **Ctrl+B** or clicking Build Model:

- 1 Generates code from the model.
- 2 Compiles and links the code into a shared object called `librsedu.so`.
- 3 Detects if a PX4 flight controller is connected over Bluetooth.
- 4 Loads the `librsedu.so` into the connected PX4 flight controller.
- 5 Runs the `librsedu.so` in the PX4 flight controller.

Build

With this option, pressing **Ctrl+B** or clicking Build Model:

- 1 Generates code from the model.
- 2 Compiles and links the code into a shared object called `librsedu.so`.

This option does not load and run the `librsedu.so` on the PX4 flight controller.

CMake configuration

This parameter is used to select the CMake configuration file that will be used for building the PX4 firmware.

If you have performed the Hardware Setup process, this field contains the name of the CMake file that you selected during that process. Click **Change CMake Configuration** to launch the Hardware Setup screens again and change the CMake file.

Automatically determine serial port for firmware upload

Use this parameter to automatically determine the serial port for firmware upload, based on the hardware connections.

Note This parameter does not appear if you select the Hardware board as **PX4 Host Target**.

Serial port for firmware upload

Use this parameter to enter the name of the serial port of the host computer to which the PX4 flight controller is connected over USB, for firmware upload.

Note This parameter does not appear if you select the Hardware board as **PX4 Host Target**.

Serial port for NuttX (NSH) Terminal

Use this parameter to enter the name of the serial port of the host computer used for NuttX connection (to launch the NSH shell on the hardware).

Allow flashing FMUv3 CMake configuration on Pixhawk 1

This option appears only if you select PX4Pixhawk 1 or PX4 Pixhawk Series as the Hardware Board. Select this parameter if you need to allow flashing FMUv3 configuration on Pixhawk 1.

Note This option should be carefully used because enabling this option will result in the following:

- There will not be any check either for the selected CMake compatibility on the connected Pixhawk controller or for any silicon errata on the Pixhawk board.
- If the CMake is not compatible with the Pixhawk controller, the entire board can get into a faulty state.

Hence, only use this option if you have connected Pixhawk 1 and you want to flash FMUv3 CMake on Pixhawk 1.

Simulator

Use this parameter to select the simulator that is used as part of PX4 Software-in-the-loop (PX4 SITL).

Note This parameter appears only if you select the Hardware board as **PX4 Host Target**.

Settings

Default: jMAVSim

See Also

“Model Configuration Parameters for Parrot Minidrone” (Simulink)

HITL

Enable HITL Mode

Select this check box to enable HITL Mode and allow the Simulink generated code integrated with PX4 firmware to run on real flight controller hardware.

Settings

Default: off

Simulator

Use this parameter to select the simulator that is used as part of PX4 Hardware-in-the-loop (HITL).

Settings

Default: jMAVSim

Connected I/O

Note This tab does not appear if you select **PX4 Pixhawk Series** as the Hardware board.

Hardware board Serial Port

Use this parameter to define the serial port for Connected I/O on the PX4 flight controller. Connected I/O uses this port for communication between the PX4 flight controller and host computer.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default: /dev/ttyACM0

Use the same host serial port for Connected I/O as used for firmware upload

Use this parameter to set the same host serial port for Connected I/O as the one used for “Serial port for firmware upload” on page 4-11.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default:on

Host Serial Port

Use this parameter to explicitly define the serial port for External mode on the host computer, if it is different from the one that you have used for firmware upload. External mode uses this port for communication between the PX4 flight controller and host computer.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default: COM6

Clocking

CPU Clock (MHz)

The frequency of the primary CPU core clock of the PX4 flight controller connected to the host computer.

Note The value of the **CPU Clock (MHz)** parameter is read-only if you select any specific hardware from the Hardware board list. However, you can enter a specific clock frequency value for this parameter if you select **PX4 Pixhawk Series** as the Hardware board.

Note This parameter does not appear if you select the Hardware board as **PX4 Host Target**.

Settings

Default:168

External Mode

External mode enables Simulink on the host computer to communicate with the deployed model on the PX4 flight controller during runtime. This feature helps you to tune the parameters and perform real-time monitoring of the model running on the PX4 Autopilot.

Communication interface

Select the transport layer that the External mode uses to exchange data between the host computer and the PX4 flight controller.

Settings (except for PX4 Host Target)

Default:Serial

Settings for PX4 Host Target

Default:TCP/IP

Use the same host serial port for External mode as used for firmware upload

Use this parameter to set the same host serial port for External mode as the one used for “Serial port for firmware upload” on page 4-11.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default:on

Host Serial Port

Use this parameter to explicitly define the serial port for External mode on the host computer, if it is different from the one that you have used for firmware upload. External mode uses this port for communication between the PX4 flight controller and host computer.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default: COM6

Hardware board Serial Port

Use this parameter to define the serial port for External mode on the PX4 flight controller. If you are using the micro-USB port for External mode, this corresponds to the default value `/dev/ttyACM0`. External mode uses this port for communication between the PX4 flight controller and host computer.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default: COM6

Set logging buffer size automatically

Use this parameter to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation.

Note This parameter does not appear if you select **PX4 Host Target** as the Hardware board.

Settings

Default: on

Logging buffer size (in bytes)

Specify the number of bytes to allocate for the buffer in the hardware during simulation. Specify **Logging buffer size (in bytes)** to a value large enough to accommodate the logged signals.

When specifying the buffer size, ensure that:

- The maximum value of this parameter does not exceed the available PX4 memory, which the Simulink Real-Time™ also uses to store other items. For example, in addition to signal logging data, the software also uses the target computer memory for the Simulink Real-Time kernel, real-time application, and scopes.

Assume that your model has six data items (time, two states, two outputs, and task execution time). If you enter a buffer size of 100000, the target object property `tg.MaxLogSamples` is calculated as `floor(100000 / 6) = 16666`. After the buffer saves 16666 sample points, it wraps and further samples overwrite the older ones.

- You enter a logging buffer size larger than the available RAM on PX4. When you download and initialize the real-time application, the hardware displays a message, `ERROR: allocation of logging memory failed`. To avoid this error, either install more RAM or reduce the buffer size for logging, and then restart the hardware. To calculate the maximum buffer size available for your real-time application logs, divide the amount of available RAM by `sizeof(double)`, or 8. Specify that value in the **Logging buffer size (in bytes)** value.

Settings

Default: 1024

Verbose

Select this check box to view the External mode execution progress and updates in the Diagnostic Viewer or in the MATLAB Command Window.

See Also

["Model Configuration Parameters for Parrot Minidrone" \(Simulink\)](#)

Onboard Connectivity

Onboard Computer IP Address

Specify the IP address of the onboard computer. This is required to establish communication between the onboard computer and PX4 hardware.

Settings

Default:127.0.0.1

PIL

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Hardware board serial port

Use this parameter to explicitly define the serial port for PIL on the host computer, if it is different from the one that you have used for External mode. PIL uses this port for communication between the PX4 flight controller and host computer.

Settings

Default:/dev/ttyACM0

Use the same host serial port for PIL as used for firmware upload (mentioned under 'Build Options')

Use this parameter to set the same host serial port for PIL as the one used for “Serial port for firmware upload” on page 4-11.

Settings

Default:on

Host Serial Port

Use this parameter to explicitly define the serial port for PIL on the host computer, if it is different from the one that you have used for firmware upload. PIL uses this port for communication between the PX4 flight controller and host computer.

Settings

Default: COM6

I2C

To set the I2C communication parameters, use the I2C options. The bus speed determines the rate of data communication between the peripherals that are connected together by the I2C bus.

Bus 1 speed (KHz)

Use the I2C option to set the Bus 1 speed parameter.

Settings

Default:100

Bus 2 speed (KHz)

Use the I2C option to set the Bus 2 speed parameter.

Settings

Default:100

Bus 3 speed (KHz)

Use the I2C option to set the Bus 3 speed parameter.

Settings

Default:100

Bus 4 speed (KHz)

Use the I2C option to set the Bus 4 speed parameter.

Settings

Default:100

CAN

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

CAN Port

This field defines the CAN port used on the host computer.

The value of the **CAN port** parameter is read-only. For any selected PX4 hardware, only one CAN port is supported. If the PX4 hardware has only one port, the parameter value is set to CAN. If the PX4 hardware has two ports, then CAN1 is supported.

Supported CAN ports for different boards:

PX4 boards	CAN port
PX4 Pixhawk 1	CAN
PX4 Pixhawk 2.1 (Cube)	CAN2
PX4 Pixhawk 4	CAN1
PX4 Pixhawk Series	CAN1
PX4 Pixracer	CAN

Settings

Default:CAN

Baud rate (in bits/s)

This field defines the rate at which data is transferred over CAN network. (in bits/s).

Settings

Default:500000

Test mode

Use this parameter to enable CAN test mode. Select one of these options.

- **Off:** This option enables normal transmission and receiving of CAN messages in the network.
- **Loopback:** This option enables loopback testing of CAN messages. In this option, the CAN network treats its own transmitted messages as received messages.
- **Silent:** This option enables only receiving of CAN messages in the network. Transmission of CAN messages is disabled.

Settings**Default:Off**

MAVLink

Note This tab does not appear if you select **PX4 Host Target** as the Hardware board.

Enable MAVLink on /dev/ttyACM0

Select this check box to enable MAVLink protocol on /dev/ttyACM0 port for establishing connection between the PX4 flight controller and QGroundControl.

Settings

Default: off

/dev/tty

Note All the /dev/tty tabs are available for configuration if you select any Pixhawk Series board as the Hardware board.

Note None of the /dev/tty tabs are displayed if you select **PX4 Host Target** as the Hardware board.

Baud rate

This field defines the rate at which data is transferred over a serial line (in bits/s).

Settings

Default: 3000000

Parity

Use this parameter to determine whether the UART or USART port generates and checks for even parity or odd parity.

Settings

Default: None

Stop bits

This parameter is used to set the number of stop bits to indicate end of a packet.

Settings

Default: 1

Enable hardware flow control

Select this parameter to enable hardware flow control using RTS and CTS pins.

View port map

Click this to view the port mapping of UART or USART port on the PX4 flight controller

PWM Frequency

Enable Oneshot125 protocol for Main PWM channels

Select this parameter to enable the Oneshot125 protocol for Main PWM channels

Configure Frequency for Main PWM channels (in Hz)

Use this parameter to enter the frequency of Main channels, if Oneshot125 protocol is not used.

Settings

Default: 400

Enable Oneshot125 protocol for AUX PWM channels

Select this parameter to enable the Oneshot125 protocol for AUX PWM channels

Configure Frequency for AUX PWM channels (in Hz)

Use this parameter to enter the frequency of AUX channels, if Oneshot125 protocol is not used.

Settings

Default: 200

Main PWM

Main PWM channel x failsafe ON time value (in us)

Use this parameter to enter the time value (in microseconds) that represents the ON time value of PWM signal for the particular Main channel, during failsafe condition.

Settings

Default: 900

Main PWM channel x disarmed ON time value (in us)

Use this parameter to enter the time value (in microseconds) that represents the ON time value of PWM signal for the particular Main channel, during the disarmed condition.

Settings

Default: 900

AUX PWM

AUX PWM channel x failsafe ON time value (in us)

Use this parameter to enter the time value (in microseconds) that represents the ON time value of PWM signal for the particular AUX channel, during failsafe condition.

Settings

Default: 900

AUX PWM channel x disarmed ON time value (in us)

Use this parameter to enter the time value (in microseconds) that represents the ON time value of PWM signal for the particular AUX channel, during the disarmed condition.

Settings

Default: 900

PWM

Note This tab appears only if you select **PX4 Host Target** as the Hardware board.

PWM channel x disarmed ON time value (in us)

Use this parameter to enter the time value (in microseconds) that represents the ON time value of PWM signal for the particular PWM channel, during the disarmed condition.

Settings

Default: 900

Overrun Action

Shutdown PX4 Autopilot upon overrun

Select this parameter to enable the shutdown of PX4 Autopilot upon task overrun.

Bus Mapping for I2C Blocks

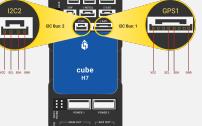
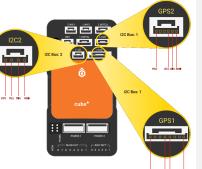
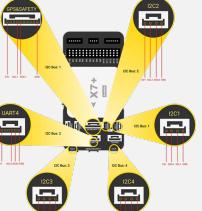
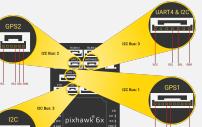
I2C Bus Port Numbers for Labels on PX4 Autopilots

The I2C Controller Read and I2C Controller Write blocks in the UAV Toolbox Support Package for PX4 Autopilots support sending and receiving data using the bus on the PX4 flight controller board.

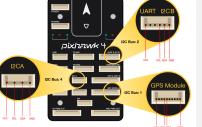
This section lists the mapping of the different port labels on the PX4 flight controller board to the bus port numbers that you can select when configuring the blocks in your Simulink model.

Note For Pixhawk 6C Autopilots with serial numbers in the format XXXX 001 XXXXXX (located on the packaging), the TELEM3 port can also function as an I2C device. Do not connect non-I2C devices to the TELEM3 port if you have this version.

Bus Mapping for Supported Target Hardware Boards

Target Hardware Supported	Bus Mapping Diagram
Cube Blue H7	
Cube Orange	
Cube Orange Plus	
CUAV X7+	
Pixhawk 6c	
Pixhawk 6x	
Pixhawk 1	
Pixhawk 2.1 (Cube)	

5 Bus Mapping for I2C Blocks

Target Hardware Supported	Bus Mapping Diagram
Pixhawk 4	
Pixracer	

Functions

getMATFilesFromPixhawk

Retrieve MAT-files from SD card inserted on Pixhawk hardware board

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.

Syntax

```
getMATFilesFromPixhawk(modelname)
getMATFilesFromPixhawk(modelname,Name,Value)
```

Description

`getMATFilesFromPixhawk(modelname)` retrieves MAT-files corresponding to the model name, specified as an argument, from the SD card inserted on the Pixhawk hardware board.

Note To work with this function, you need to enable MAVLink in the Simulink model for which the MAT-files are generated. For details, see “Enable MAVLink”.

`getMATFilesFromPixhawk(modelname,Name,Value)` retrieves MAT-files from the SD card based on the options specified by one or more `Name,Value` pair arguments.

Examples

Retrieve MAT-files

Retrieve all MAT-files that are generated as part of the latest run (the MAT-files that are generated after you last clicked **Build, Deploy & Start** in Simulink).

Consider that the file name of the Simulink model that is configured for MAT-file logging is `px4_squarepath_sdcard`. This Simulink model is deployed to the Pixhawk hardware three times, and you want to retrieve files only from the latest run.

To do this, enter the following command at the MATLAB command prompt after the model has successfully run on the hardware:

```
getMATFilesFromPixhawk(px4_squarepath_sdcard)
```

After the above function is run, only the files from the latest (third) run are copied to the current folder in MATLAB. For example, the file names of the retrieved files in the current folder look like this:

```
px4_squarepath_sdcard_3_1.mat
px4_squarepath_sdcard_3_2.mat
px4_squarepath_sdcard_3_3.mat
```

The suffix 3 just after the model name represents the run number.

Specify Options and Retrieve MAT-files

Retrieve MAT-files by specifying options as name-value pairs.

Consider that the file name of the Simulink model that is configured for MAT-file logging is px4_squarepath_sdcard. This Simulink model is deployed to the Pixhawk hardware three times, and you want to retrieve all files from all the successful runs, and also delete all the files from SD card after retrieving the files.

To do this, enter the following command at the MATLAB command prompt:

```
getMATFilesFromPixhawk(px4_squarepath_sdcard, 'ExtractFilesFromAllRuns', true, 'DeleteAfterRetrieval')
```

After the above function is run, all the files from all the successful runs are copied to the current folder in MATLAB, and they are also deleted from the SD card. For example, the file names of the retrieved files in the current folder look like this:

```
px4_squarepath_sdcard_1_1.mat
px4_squarepath_sdcard_1_2.mat
px4_squarepath_sdcard_1_3.mat
px4_squarepath_sdcard_2_1.mat
px4_squarepath_sdcard_2_2.mat
px4_squarepath_sdcard_2_3.mat
px4_squarepath_sdcard_3_1.mat
px4_squarepath_sdcard_3_2.mat
px4_squarepath_sdcard_3_3.mat
```

Input Arguments

modelName — File name of the Simulink model
character vector

File name of the Simulink model that is configured for MAT-file logging and deployed on the Pixhawk hardware board with SD card inserted.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1, ..., NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name in quotes.

Example:

ExtractFilesFromAllRuns — Extract MAT-files from SD card for all runs
false (default) | true

Extract all the MAT-files from the SD card inserted on the Pixhawk hardware, for all runs (for all successful **Build, Deploy & Start** actions)

Example:

```
getMATFilesFromPixhawk(px4_squarepath_sdcard, 'ExtractFilesFromAllRuns', true)
```

Data Types: logical

DeleteAfterRetrieval — Delete the MAT-files that are retrieved from SD card

false (default) | true

Delete the MAT-files that are retrieved from the SD card inserted on the Pixhawk hardware, after successful retrieval of the files.

Tip Set the value of `DeleteAfterRetrieval` to `true` always because this will reduce the time taken to retrieve the MAT-Files from SD card next time.

Example:

```
getMATFilesFromPixhawk(px4_squarepath_sdcard, 'DeleteAfterRetrieval', true)
```

Data Types: logical

COMPort — The COM Port of the host computer to which the Pixhawk hardware board is connected

character vector | string scalar

The COM Port of the host computer to which the Pixhawk hardware board is connected to retrieve the MAT-files, specified as a character vector or string.

Note This name-value pair is generally used if you select **PX4 Pixhawk Series** as the Hardware board. For all other boards such as **PX4 Pixhawk 1**, the COM port is determined automatically unless you specify the COM port as part of this name-value pair.

Example:

```
getMATFilesFromPixhawk(px4_squarepath_sdcard, 'DeleteAfterRetrieval', true, 'COMPort', 'COM15')
```

Data Types: char | string

Version History

Introduced in R2020b

px4MATFilestitcher

Combine multiple MAT-files retrieved from SD card into a single MAT-file

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.

Syntax

```
px4MATFilestitcher()
px4MATFilestitcher(folderpath)
```

Description

`px4MATFilestitcher()` combines all MAT-files, starting with the same file name in the current folder in MATLAB, into a single MAT-file. The function identifies the first part of the file name that is common for a specific run, and then combine those files in the order determined by the numeric characters present at the end of their file names.

`px4MATFilestitcher(folderpath)` combines all MAT-files, starting with the same file name in the folder specified using the full folder path. The function identifies the first part of the file name that is common for a specific run, and then combine those files in the order determined by the numeric characters present at the end of their file names.

Examples

Combine All MAT-files Starting with the Same File Name in Current Folder

Combine all MAT-files that start with the same file name (part of the same run) in the current folder in MATLAB.

Consider that the file name of the MAT-files start with `px4_squarepath_sdcard` and that there are nine MAT-files in the current folder in MATLAB, which correspond to three runs, as listed below.

```
px4_squarepath_sdcard_1_1.mat
px4_squarepath_sdcard_1_2.mat
px4_squarepath_sdcard_1_3.mat
px4_squarepath_sdcard_2_1.mat
px4_squarepath_sdcard_2_2.mat
px4_squarepath_sdcard_2_3.mat
px4_squarepath_sdcard_3_1.mat
px4_squarepath_sdcard_3_2.mat
px4_squarepath_sdcard_3_3.mat
```

To combine the files, enter the following command at the MATLAB command prompt:

```
px4MATFilestitcher()
```

The files are combined based on the run (the suffix after the file name) to generate three stitched files:

```
px4_squarepath_sdcard_1_stitched.mat  
px4_squarepath_sdcard_2_stitched.mat  
px4_squarepath_sdcard_3_stitched.mat
```

For example, all data points logged in `px4_squarepath_sdcard_1_1.mat`, `px4_squarepath_sdcard_1_2.mat`, and `px4_squarepath_sdcard_1_3.mat` are copied to the combined file - `px4_squarepath_sdcard_1_stitched.mat`.

Combine All MAT-files Starting with the Same File Name in Specified Folder

Combine all MAT-files that start with the same file name (part of the same run) in any folder that you specify.

Consider that there is a folder named `Today_Runs` inside the current folder in MATLAB. The file name of the MAT-files in `Today_Runs` start with `px4_squarepath_sdcard` and that there are nine MAT-files in the current folder in MATLAB, which correspond to three runs, as listed below.

```
px4_squarepath_sdcard_23_1.mat  
px4_squarepath_sdcard_23_2.mat  
px4_squarepath_sdcard_23_3.mat  
px4_squarepath_sdcard_24_1.mat  
px4_squarepath_sdcard_24_2.mat  
px4_squarepath_sdcard_24_3.mat  
px4_squarepath_sdcard_25_1.mat  
px4_squarepath_sdcard_25_2.mat  
px4_squarepath_sdcard_25_3.mat
```

To combine the files, enter the following command at the MATLAB command prompt:

```
px4MATFilestitcher(fullfile(pwd, 'Today_Runs'))
```

The files are combined based on the run (the suffix after the file name) to generate three stitched files:

```
px4_squarepath_sdcard_23_stitched.mat  
px4_squarepath_sdcard_24_stitched.mat  
px4_squarepath_sdcard_25_stitched.mat
```

If the folder `Today_Runs` is present in a different path other than inside the current folder in MATLAB, specify the full path to combine the files. For example:

```
px4MATFilestitcher('C:\User\Autopilot\matfiles\Today_Runs')
```

Note However, in this case (the folder present in a different path other than inside the current folder in MATLAB), the stitched files are always generated in the current folder in MATLAB.

Input Arguments

folderpath — Absolute path of the folder that contains the MAT-files that needs to be combined

character vector

Absolute path of the folder in the host computer, which contains the MAT-files (which were retrieved from the SD card) that need to be combined.

Data Types: char | string

Version History

Introduced in R2020b

createPX4uORBMessage

Create custom uORB topic for logging in a Simulink model

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.

Syntax

```
createPX4uORBMessage(messageName, uORBFields)
```

Description

`createPX4uORBMessage(messageName, uORBFields)` creates a custom uORB topic for logging in a Simulink model running on a Pixhawk hardware board.

A custom uORB message refers to a user-defined message format in the uORB (micro Object Request Broker) messaging system. uORB is a publish-subscribe messaging system used in the PX4 autopilot software for communication between different modules and components.

Examples

Create a Custom uORB message

Create a new custom uORB message `MyCustomMessage`, with the required datatype and corresponding signal names

This example shows the functioning of the API with the changes that are being made in the Firmware.

To do this, enter the following command at the MATLAB command prompt after the model has successfully run on the hardware:

```
createPX4uORBMessage('MyCustomMessage', 'float32 x', 'float64 y')
```

This creates a new message `MyCustomMessage.msg` with the mentioned fields `x` and `y`, with their datatype. Then, the message is included in the specified directory, `C:\PX4\home\Firmware\msg\`

```

Current Folder
Name
LedControl.msg
LoggerStatus.msg
LogMessage.msg
MagnetometerBiasEstimate.msg
MagWorkerData.msg
ManualControlSetpoint.msg
ManualControlSwitches.msg
MavlinkLog.msg
MavlinkTunnel.msg
Mission.msg
MissionResult.msg
ModeCompleted.msg
MountOrientation.msg
MyCustomMessage.msg
NavigatorMissionItem.msg
NormalizedUnsignedSetpoint.m..
NpfStatus.msg

```

Editor - \\wsl\localhost\ubuntu\home\px4\mypx4\ PX4-Autopilot\msg\MyCustomMessage.msg

```

1 uint64 timestamp    #time since system start (microseconds)
2 float32 x
3 float64 y
4

```

It also updates the CMakeLists.txt with the newly created message as shown below.

```

CMakeLists.txt
201 VehicleCommandAck.msg
202 VehicleConstraints.msg
203 VehicleControlMode.msg
204 VehicleGlobalPosition.msg
205 VehicleImu.msg
206 VehicleImuStatus.msg
207 VehicleLandDetected.msg
208 VehicleLocalPosition.msg
209 VehicleLocalPositionSetpoint.msg
210 VehicleMagnetometer.msg
211 VehicleOdometry.msg
212 VehicleOpticalFlow.msg
213 VehicleOpticalFlowVel.msg
214 VehicleRatesSetpoint.msg
215 VehicleRoi.msg
216 VehicleStatus.msg
217 VehicleThrustSetpoint.msg
218 VehicleTorqueSetpoint.msg
219 VehicleTrajectoryBezier.msg
220 VehicleTrajectoryWaypoint.msg
221 VtolVehicleStatus.msg
222 Wind.msg
223 YawEstimatorStatus.msg
224 MyCustomMessage.msg
225 SimulinkCustomMessage.msg
226 )
227 list(SORT msg_files)
228
229 px4_list_make_absolute(msg_files ${CMAKE_CURRENT_SOURCE_DIR} ${msg_files})
230
231 if(NOT EXTERNAL_MODULES_LOCATION STREQUAL "")
232     # Check that the msg directory and the CMakeLists.txt file exists
233     if(EXISTS ${EXTERNAL_MODULES_LOCATION}/msg/CMakeLists.txt)
234         add_subdirectory(${EXTERNAL_MODULES_LOCATION}/msg external_msg)
235
236

```

To use this custom uORB message, build the PX4 firmware using the Hardware Setup Screen.

Input Arguments

messageName — Name of the message that you want to create
character vector

Name of the message that you want to create. Message name must be a unique name that has not been used before in your project.

Data Types: char | string

uORBFIELDS — Varargin input
character vector

Varargin input containing information about the required signals with their corresponding data types and signal names as mandatory inputs.

Data Types: char | string

Version History

Introduced in R2023b

createCustomPX4Parameter

Create custom PX4 parameter for writing to the PX4 system parameter

Note Add-On Required: This feature requires the UAV Toolbox Support Package for PX4 Autopilots add-on.

Syntax

```
createCustomPX4Parameter()
```

Description

`createCustomPX4Parameter()` creates a custom PX4 parameter and also allows easy parameter modification.

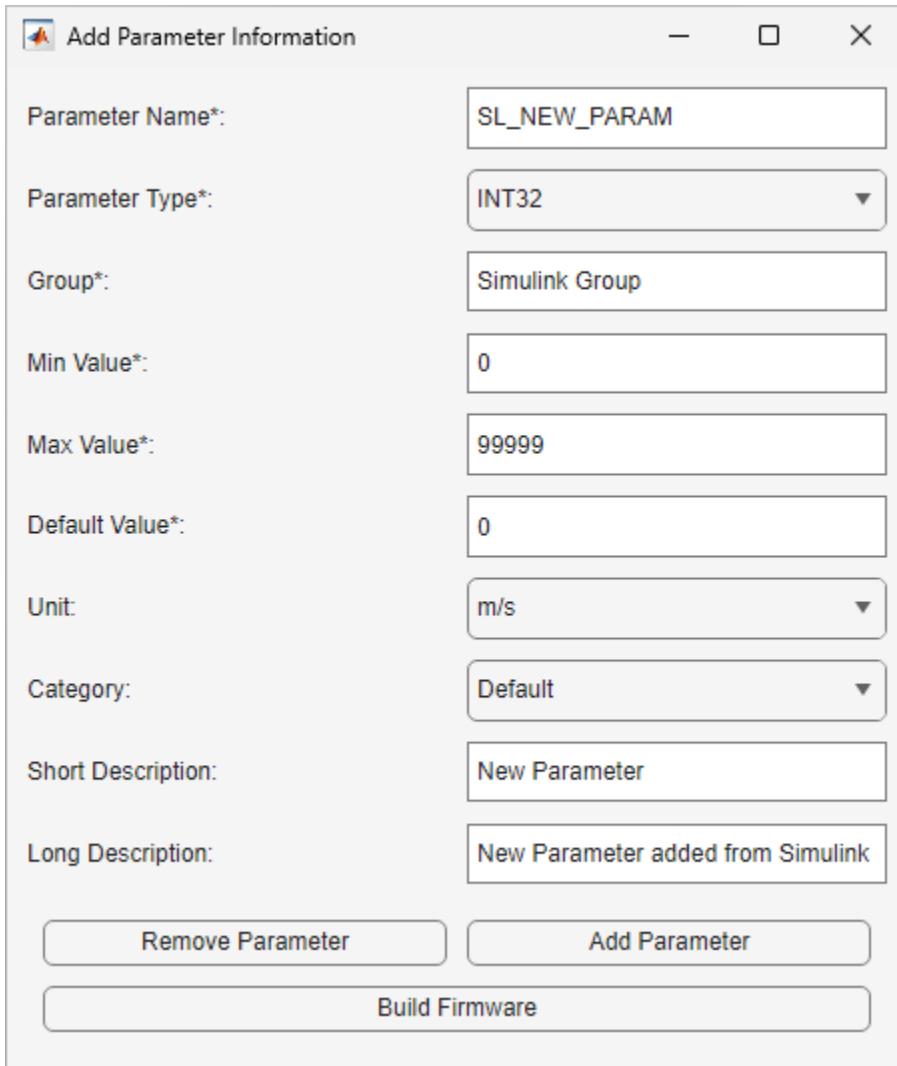
Examples

Create a Custom PX4 Parameter

Create a new custom PX4 Parameter, with the required parameter type and corresponding values.

This example shows the functioning of the API with the changes that are being made in the Firmware.

```
createCustomPX4Parameter()
```



In the **Add Parameter Information** dialog box, add the information for the parameter.

- 1 Specify a name for the parameter and select the parameter type.

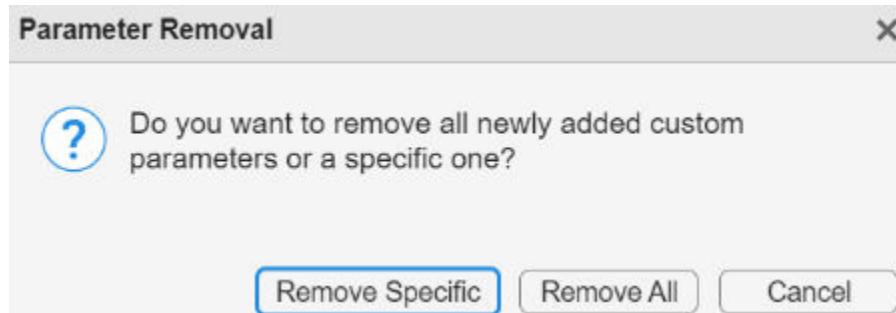
Note Parameter name must be less than 16 characters.

- 2 Specify the group for the parameter.
- 3 Specify minimum and maximum values.
- 4 Specify default value, which must be within minimum and maximum value specified.
- 5 Select the unit and category type for the parameter.
- 6 Specify short and long description for the parameter.

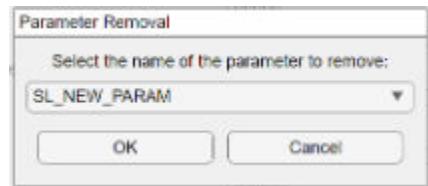
Note Short description must be less than 70 characters.

- 7 Click **Add Parameter** to add the parameter.

To remove a parameter, click **Remove Parameter**. This provides an option to remove any specific parameter or to remove all parameters.



Click **Remove All** to remove all the parameters. Clicking **Remove Specific** provides an option to select and remove specific parameter.



Note After adding or deleting a parameter, you must build the PX4 firmware by clicking **Build Firmware**.

Version History

Introduced in R2025a

