Appendices

Appendix A

UAV Toolbox Capabilities for Flight Control Law Development

The MathWorks® UAV Toolbox [17] with the Support Package for PX4 Autopilots [16] provides the tools needed to interface with the PX4 firmware architecture in Simulink®. The toolbox and support package provide blocks for interfacing with PX4 described in Section A.1 as well as modes for firmware testing and deployment described in Section A.2.

An overview of the UAV Toolbox custom PX4 firmware deployment workflow is shown in Figure A.1. The process begins by either cloning the PX4 source code (Section B.2.2) or starting from a locally modified version (Section B.3), after which the MATLAB® Hardware Setup screens (Section B.2.3) are used to configure the toolchain and build target. A Simulink® model is used to build flight control logic with proper input/output interface blocks available in the UAV Toolbox Support Package for PX4 Autopilots (Section A.1.) From this model, two execution paths are supported. The first is "Connected IO," which runs the logic in Simulink® and streams data to and from an autopilot running PX4 without embedding full firmware, suitable for bench testing (Section A.2.1.) The second option is "Build, Deploy, and Start," which generates and flashes custom PX4 firmware to the autopilot for flight testing (Section A.2.3.) Results from bench and flight tests are then used to iteratively refine the Simulink® model and parameters.

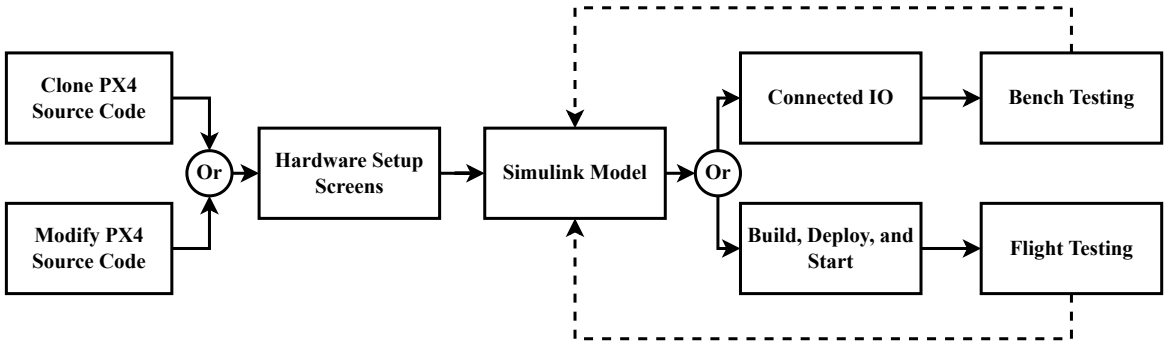

Figure A.1: Overview of the UAV Toolbox custom PX4 firmware deployment toolchain.

A.1 PX4 Interface Blocks

The UAV Toolbox Support Package for PX4 Autopilots includes blocks in Simulink® that can read and write to and from PX4 directly. This section describes which blocks are useful for flight control algorithm deployment. The best practices described are a culmination of lessons learned from the following experiences of the author, described in References [8, 10, 11, 12, 13, 20, 21]. Many of the blocks available for use are not described in this section. As a result, the full list of blocks available to interface with PX4 can be found in the support package documentation in Reference [15].

A.1.1 uORB Read, Write and Message

The PX4 firmware uses micro Object Request Broker (uORB) messaging to access data from different firmware modules. As a result, the uORB messages provide a plethora of information about the vehicle. The "PX4 uORB Read," "PX4 uORB Write," and "PX4 uORB Message" blocks are shown in Figure A.2.

Although the full list of topics can be found in Reference [19], useful topics to read into a

(a) PX4 uORB Read Block (b) PX4 uORB Message Block (c) PX4 uORB Write Block

Figure A.2: uORB Read, Message, and Write Simulink® blocks from the UAV Toolbox Support Package for PX4 Autopilots.

Simulink® diagram for algorithm development can be found in Table A.1.

Table A.1: Useful uORB topics for control algorithm development.

uORB Topic	Selected Msg(s)	Usage
`ActuatorArmed`	`armed`	Vehicle arming status flag
`Airspeed`	`true_airspeed_m_s`	True airspeed $[\frac{m}{s}]$
`RcChannels`	`channels`	Normalized pilot RC inputs for control mapping [-1,1]
`VehicleAttitude`	`q`	Quaternion rotation from the body frame (FRD) to the earth frame (NED)
`VehicleAngularVelocity`	`xyz`	Angular rates $[\frac{rad}{s}]$
`VehicleLocalPosition`	`x`, `y`, `z`, `vx`, `vy`, `vz`	Translational position (NED) [m] and Translational velocity (NED) $[\frac{m}{s}]$
`VehicleAirData`	`rho`	Air density $[\frac{kg}{m^3}]$
`DistanceSensor`	`current_distance`	LiDAR altitude [m]
`EscStatus`	`esc`	ESC feedback signals
`VehicleStatus`	`nav_state`	Navigator mode (i.e., flight mode, flight termination, etc.)

Writing to a uORB topic can be completed with the uORB Message, Bus Assignment, and uORB Write blocks as shown in Figure A.3.

A.1.2 Reading Parameters

The "Read Parameter" block shown in Figure A.4 allows the value of a parameter programmed from QGroundControl to be read into the algorithm coded in Simulink®. Custom

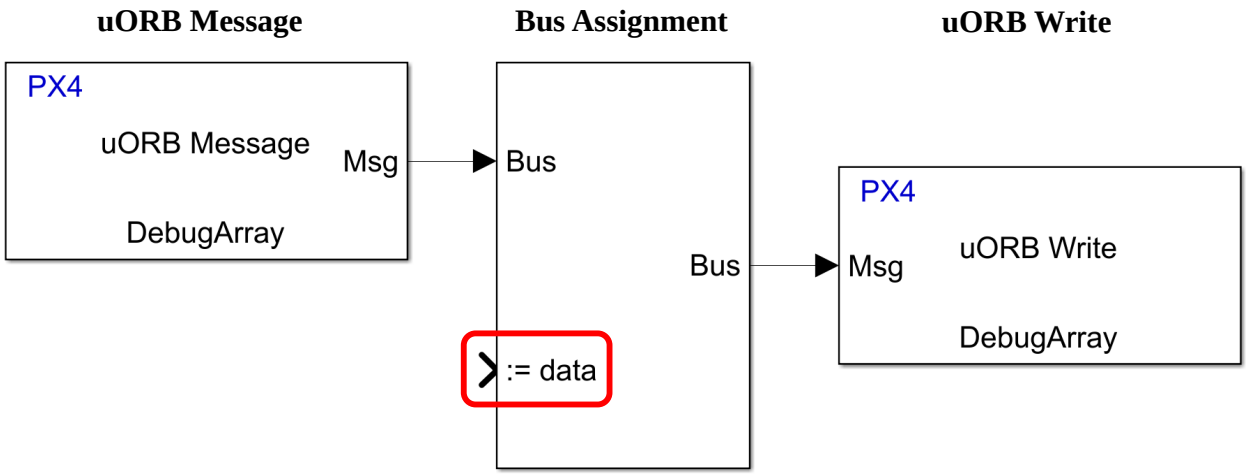

Figure A.3: Simulink® logic for writing to the uORB topic `DebugArray`.

parameters can be used to effect change in an algorithm in real-time. Some use cases used by the author include switching control modes, tuning control gains, and activating custom flight safety logic.

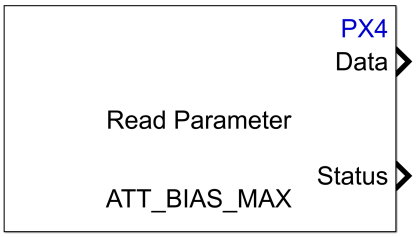

Figure A.4: Read Parameter block from the UAV Toolbox Support Package for PX4 Autopilots.

Instructions on how to create custom parameters can be found in Section B.3.3.

A.1.3 Writing to Control Effectors

The "PX4 PWM Output" block in the UAV Toolbox has been used by the author since R2022a to send commands to PWM-compatible servos and electronic speed controllers (ESCs). Beginning in R2024a, when support for PX4 v1.14.0 was introduced, the way this

block interfaces with PX4 changed. Before R2024a, the block wrote directly to the MAIN or AUX PWM channel selected in the block dialog. Since R2024a, the block instead writes to the actuator assignments defined in the "Actuators" tab of QGC. Users now assign each individual motor or servo to a channel or index depending on the communication protocol (for example, PWM, DroneCAN, etc.) within QGC. The firmware then enforces that the airframe configuration selected in the "Airframes" tab has the required number of motors and servos. This architecture gives users greater flexibility to interface control effectors that use various communication protocols. In practice, the PWM Output block MAIN channels 1-8 correspond to motors 1–8, while the PWM Output block AUX channels 1-8 correspond to servos 1–8 as configured in QGC.

Table A.2 compares the assignments for the control effectors present on the CZ-150 fixed wing aircraft. Each row lists the control effector, the physical servo rail port, the actuator (motor or servo) number in PX4, and the corresponding channel that would be used by the PWM Output block before and after the modification in R2024a.

Table A.2: Comparison of channels used by the PWM Output block before and after the R2024a modification.

Control Effector	Servo Rail Port	Actuator Number	PWM Output Block Assignment	
			Since R2022a	Since R2024a
Left aileron	MAIN 1	Servo 1	MAIN 1	AUX 1
Right aileron	MAIN 2	Servo 2	MAIN 2	AUX 2
Elevator	MAIN 3	Servo 3	MAIN 3	AUX 3
Rudder	MAIN 4	Servo 4	MAIN 4	AUX 4
Left flap	MAIN 5	Servo 5	MAIN 5	AUX 5
Right flap	MAIN 6	Servo 6	MAIN 6	AUX 6
Propeller	MAIN 7	Motor 1	MAIN 7	MAIN 1

Regardless of the UAV Toolbox release used, the PWM Output block overrides PX4 safety logic. Since the functional change made in the R2024a release, the PWM Output block writes to the `ActuatorTest` uORB topic. To maintain the safety of PX4 arming and disarming

logic, the PX4 arming status flag of the vehicle should be read in to the "Arm" port of the PWM Output block from the `ActuatorArmed` uORB topic. As a result, any safety or overrides from the PX4 firmware will change the arming status flag of the vehicle, and effectively the PWM Output block. An example workflow for this safety logic can be seen in Figure A.5

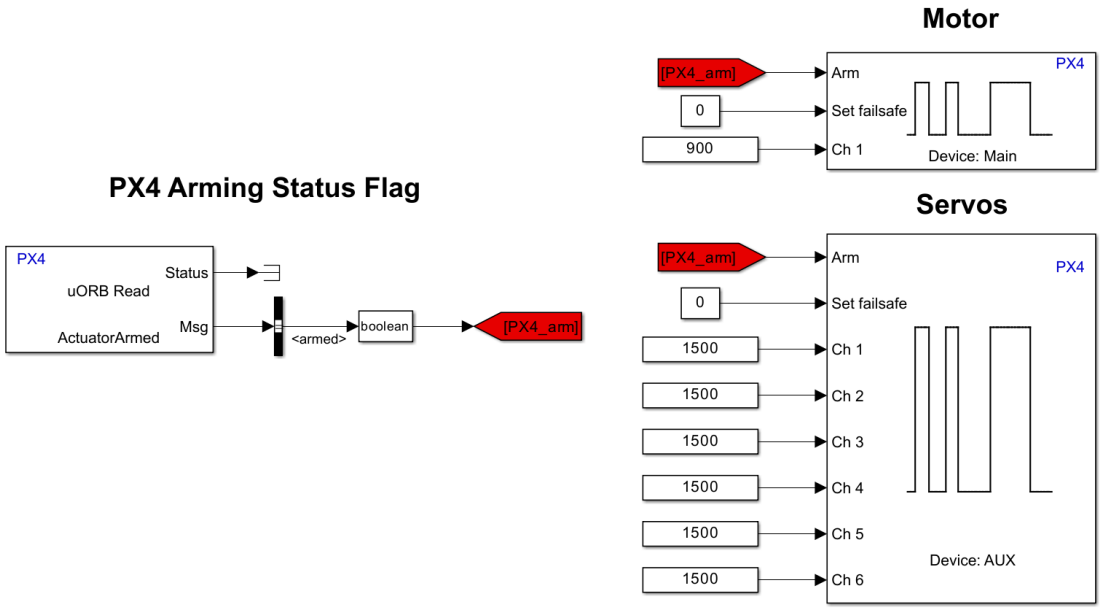

Figure A.5: Example logic to safely arm and disarm the PWM Output block using the PX4 arming status flag.

A.2 Firmware Testing and Deployment Modes

A.2.1 Connected IO

The "Connected IO" mode is the first of three modes provided under the Hardware tab in Simulink®. Connected IO generates the firmware and runs the code on a computer while sending and receiving signals using a hardwired connection to the autopilot. This mode is

helpful for efficient firmware checkouts without needing to load custom firmware onto the autopilot. Additionally, scopes can be used to monitor internal and external signals. The behavior of control surfaces and motors can be observed during bench testing by incorporating the hardware into the loop. Although the Connected IO mode does not run in real-time, the consistency and utility of Connected IO during bench testing proved useful for the initial testing of custom algorithms before embedding them as firmware onboard the autopilot.

A.2.2 Monitor and Tune

"Monitor and Tune" is an alternative connected mode run by embedding the firmware onboard the autopilot for real-time applications, while maintaining a hardwired USB connection. This allows the ability to tune parameters and change gains in the Simulink® diagram while running firmware on the autopilot. Although this is a promising capability, all checks and modifications were iterated in bench testing using the Connected IO mode due to its recompiling efficiency.

A.2.3 Build, Deploy, and Start

"Build, Deploy and Start" fully embeds the custom firmware onboard the Pixhawk and allows it to run independently from the computer. Build, Deploy and Start compiles the algorithm developed in Simulink® and embeds it as PX4 firmware on the board. This mode allows custom algorithms developed in a Simulink® model to be rapidly integrated with the autocoding feature and flight tested. Build, Deploy and Start was used to compile the algorithms flight tested in References [8, 10, 11, 12, 13, 20, 21].

Alternatively, a user can choose to just "Build" the firmware. This builds a ".px4" firmware file without uploading it to the board automatically. An advantage of just building the

firmware file includes the ability to send the firmware to other computers, and the ability to upload the firmware through QGC. The file path of all firmware builds is given in WSL as

```
<px4_repo_name>/PX4-Autopilot/build/<autopilot_and_build_target>
```

where `<autopilot_and_build_target>` in the case of the CubePilot Cube Orange with a `fixedwing` build target is `cubepilot_cubeorange_fixedwing`.

Appendix B

Setting Up The UAV Toolbox Firmware Deployment Toolchain

This appendix describes how to configure the MathWorks® UAV Toolbox to deploy custom algorithms to a PX4 autopilot from start to finish. A similar tutorial was given by the author in the appendices of [10] for the R2022a version. This appendix serves as updated and improved setup documentation for the R2025a version. The software, firmware, and hardware used are listed in Section B.1. Section B.2 provides a comprehensive guide for configuring the UAV Toolbox Support Package for PX4 Autopilots and its dependencies. Optional modifications to enhance the functionality of the source code for the research conducted in this thesis are described in Section B.3.

B.1 Materials and Software

This section describes the hardware, software, and firmware used to conduct the research described in this thesis.

B.1.1 Software

First install QGroundControl (QGC) (v4.4.2) before proceeding with the setup process.

Next, install MATLAB® R2025a. Either during the initial installation of MATLAB®, or using the "Manage Add-Ons," install the following software and utilities

- Simulink®
- UAV Toolbox
- UAV Toolbox Support Package for PX4 Autopilots [1]
- MATLAB Coder®
- Simulink Coder®
- Embedded Coder®

> **Note:** The UAV Toolbox Support Package for PX4 Autopilots requires a configuration process described in Section B.2.3. It is recommended to work through the configuration process outlined in Appendix B in the order presented.

The Windows Subsystem for Linux (WSL) 2 and Ubuntu 22.04.3 LTS are required for building PX4 on a Windows computer. The process to download and configure WSL 2 and Ubuntu is described in Section B.2.1.

B.1.2 Firmware

The R2025a release of the UAV Toolbox Support Package for PX4 Autopilots supports the use of PX4 v1.14.3. This is the version of the firmware cloned by the steps available from MathWorks® and in Section B.2.2.

[1] This package can only be installed using the "Manage Add-Ons."

B.1.3 Hardware

- CubePilot Cube Orange
- E-flite® Carbon-Z® Cessna® 150T with tricycle gear
- Laptop computer running Windows 10 or 11
- Here3 GPS
- Servo cables
- Spektrum SPM9745 Receiver
- Spektrum iX20
- SanDisk 32GB Extreme UHS-I microSDXC Memory Card
- 6s 5000mAh 22.2v 50C Lithium Polymer (LiPo) battery
- MAUCH PL 2-6s Battery Eliminator Circuit (BEC)
- MAUCH PL-200 Sensor Board
- RFD 900x v2 Modem
- Castle Phoenix Edge 100 Amp Electronic Speed Controller (ESC)
- ThunderFly TFRPM01 RPM Sensor
- Air Data Probe

The hardware assembly used for this research can be adapted to user needs.

B.2 PX4 Repository Setup

B.2.1 Setting Up the Windows Subsystem for Linux (WSL) Toolchain

The Windows Subsystem for Linux (WSL) is the software toolchain used by the R2025a version of the UAV Toolbox for compiling PX4 firmware. The WSL toolchain replaces the MinGW-w64 C/C++ Compiler used the R2022a version as described in [10].

In the command prompt as the administrator, run

```
wsl --install -d Ubuntu-22.04
```

If this does not successfully install the program, a series of steps to enable virtualization will need to be performed as the administrator in PowerShell.

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform -NoRestart
```

Then, restart the computer, and run the same three commands again as the administrator in PowerShell. Note that the fourth command will change the version of WSL to version 2.

```
dism.exe /online /enable-feature  /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all   /norestart
Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform   -NoRestart
wsl --set-default-version 2
```

Then, close PowerShell and open the command prompt (not as the administrator) and run

```
wsl --update
wsl --install -d Ubuntu-22.04
```

Then, the user will be prompted to enter a username and password for use when performing

root operations with WSL. Then, search for WSL in the windows search bar. In WSL, send the following commands

```
sudo apt update && apt upgrade
```

WSL will then prompt the user for the password for root operations. Now the PX4 repository can be cloned.

B.2.2 Cloning the PX4 Repository Using WSL

> **Note:** The installation and configuration of WSL outlined in Section B.2.1 must be complete before continuing the process described in this section.

Open the WSL application and send the following commands (where `<px4_repo_name>` is the name of the folder your repository will be cloned under)

```
mkdir <px4_repo_name>
cd <px4_repo_name>
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
```

This step will take a while to clone the repository. Once it is done and while in the directory of the `<px4_repo_name>`, enter the following commands

```
cd PX4-Autopilot
git checkout v1.14.3 -f
git submodule update --init --recursive
cd Tools/setup
bash ./ubuntu.sh
```

The bash command will require root access (the password for `sudo` commands) to begin if it has not already been entered for this session of using the WSL terminal. After the setup process completes, restart the device. The next steps will take place in the hardware setup screens in MATLAB®.

B.2.3 Building the PX4 Base Code with the Hardware Setup Screens

The following instructions walk through the setup process for the UAV Toolbox Support Package for PX4 Autopilots. This process should be used to initialize a PX4 autopilot before using the UAV Toolbox functionality offered in Simulink®. Certain steps in the Hardware Setup screens appear only when setting up the PX4 support package for the first time. These initialization steps will be marked with the tag **[Initial Only]** .

1. In MATLAB®, under the home tab, navigate to the "Add-Ons" drop down and select "Manage Add-Ons."

2. Select the gear icon in the add-on manager next to the UAV Toolbox Support Package for PX4 Autopilots as shown in Figure B.1. This will open the Hardware Setup screens.

3. **[Initial Only]** The screens will prompt the download of Python as shown in Figure B.2. Select "Automatically download and install" and then proceed to the next screen.

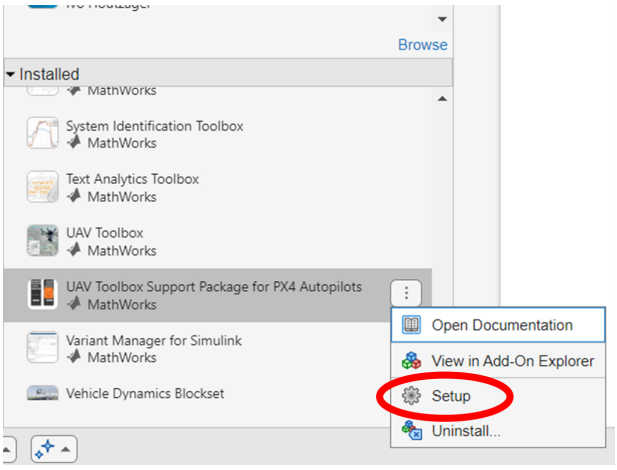

Figure B.1: Screenshot of the setup initiation from the manage add-ons menu.

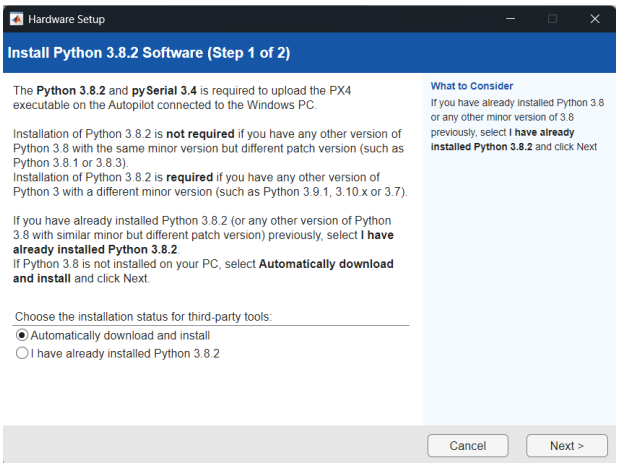

Figure B.2: Screenshot of step one of the python installation window.

4. [Initial Only] As shown in Figure B.3, click "Install" to download the proper Python and pySerial packages. Proceed to the next screen.

5. [Initial Only] Next, download the PX4 source code following the steps in Section B.2.2 if this step has not been done already. The cloning instructions correspond to the instruction documentation linked in the "Download PX4 Source Code" screen shown in Figure B.4. Once the source code has been cloned and initialized, proceed to the next screen.

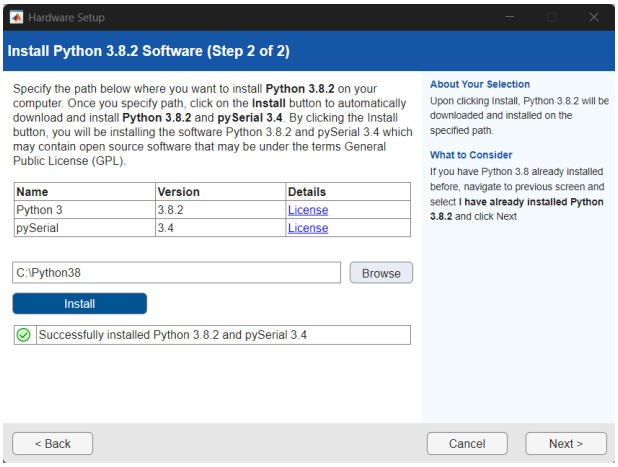

Figure B.3: Screenshot of step two of the python installation window.

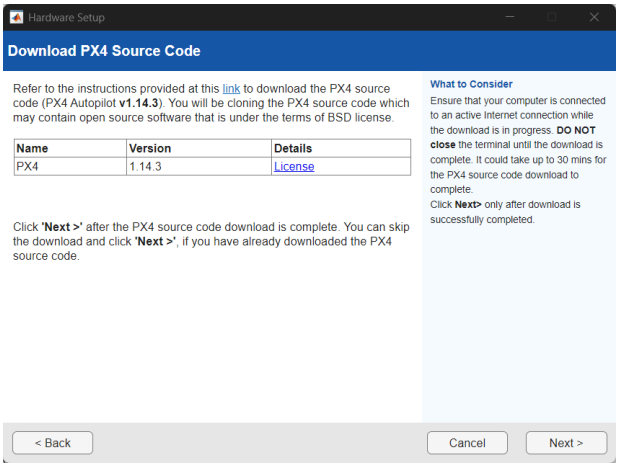

Figure B.4: Screenshot of the PX4 source code download instructions window.

6. As shown in Figure B.5, validation checks will be run on the PX4 source code installed using the instructions in Section B.2.2. Given that `<WSL_username>` is the username used during WSL setup, and `<px4_repo_name>` is the name of the folder that houses the cloned repository, the file path used in this screen must be the Linux path

   `/home/<WSL_username>/<px4_repo_name>/PX4-Autopilot`

7. Click "Validate" to verify the cloned repository will work with the UAV Toolbox. Proceed to the next screen.

8. The screen shown in Figure B.6 provides initialization instructions for the cloned PX4 source code. These steps are documented as a part of the cloning and setup instructions presented in Section B.2.2, so no further action is required. Proceed to the next screen

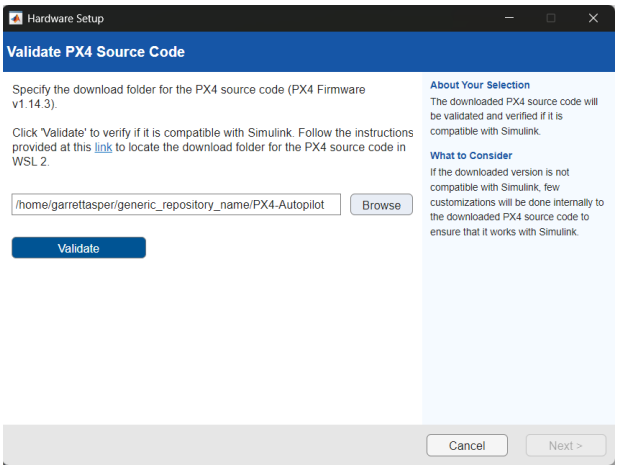

Figure B.5: Screenshot of the PX4 source code validation window.

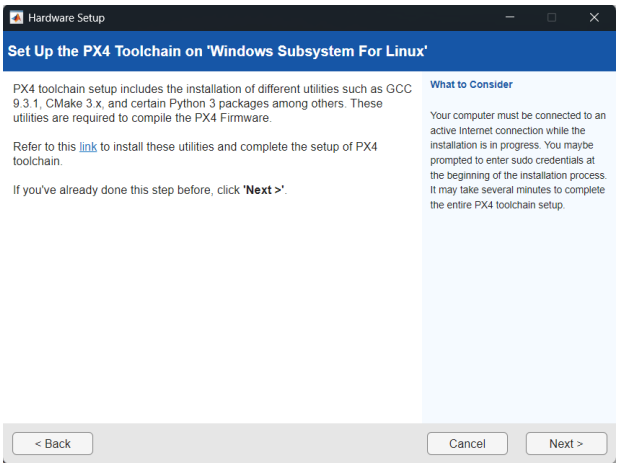

Figure B.6: Screenshot of the PX4 toolchain set up instructions window.

9. Select the appropriate PX4 Autopilot board and Build Target. For the experiments conducted in this thesis, the "PX4 Cube Orange" PX4 Autopilot Board was selected with the "`cubepilot_cubeorange_fixedwing`" build target as shown in Figure B.7. Make appropriate selections and proceed to the next screen.

10. Now, the user must decide whether to disable PX4 control, or leave the control logic enabled. For the research presented in this thesis, the PX4 control architecture was left enabled as seen in Figure B.8. Make a selection and proceed to the next screen.

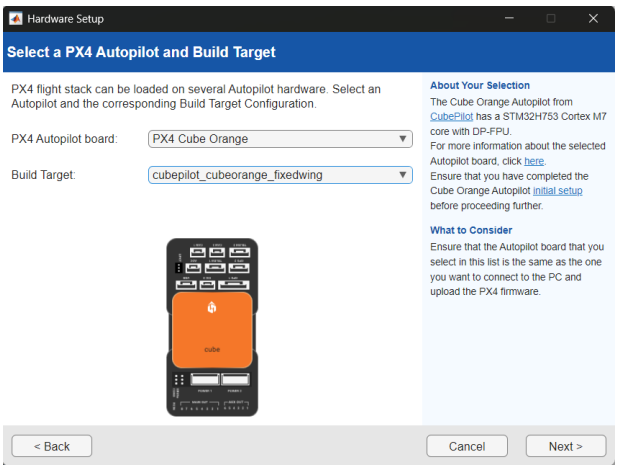

Figure B.7: Screenshot of the PX4 board and build selection window.

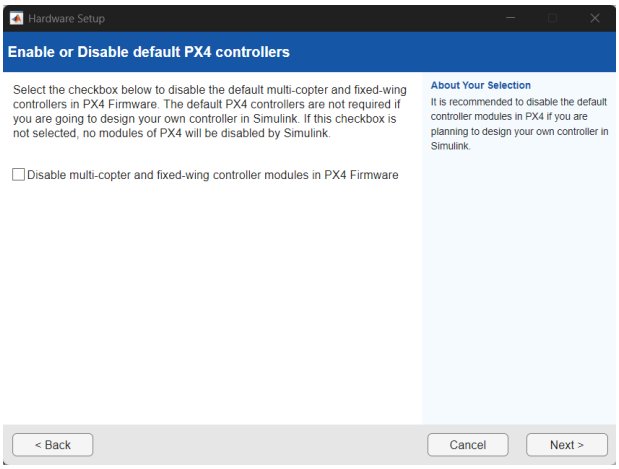

Figure B.8: Screenshot of the disable PX4 control module window.

11. The PX4 autopilot setup sequence can be modified. However, the default system startup script was used for this research as shown in Figure B.9. Make a selection, and proceed to the next screen

12. Next, verify the QGroundControl installation as shown in Figure B.10.

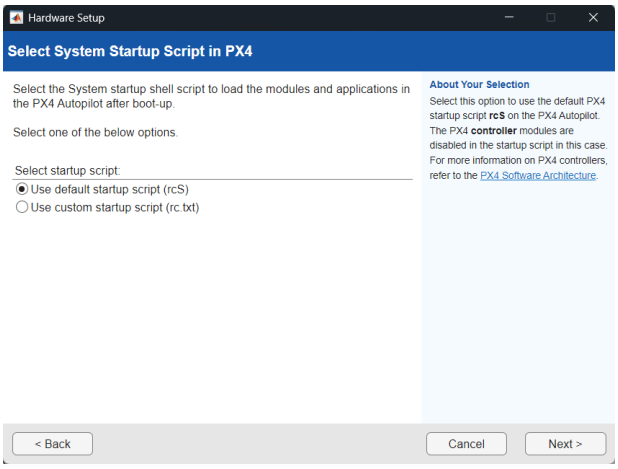

Figure B.9: Screenshot of the system startup script selection window.

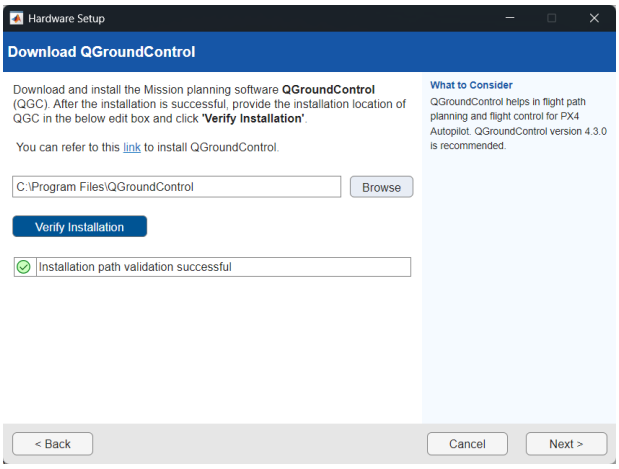

Figure B.10: Screenshot of the QGroundControl verification window.

13. The airframe selection screen shown in Figure B.11 serves as a notice. For the purposes of this step, no action needs to be taken.

14. The build PX4 firmware screen is where the user starts the build of PX4 base code compatible with the UAV Toolbox. Click "Build Firmware" as shown in Figure B.12 and monitor the Command Window in MATLAB® to view progress.

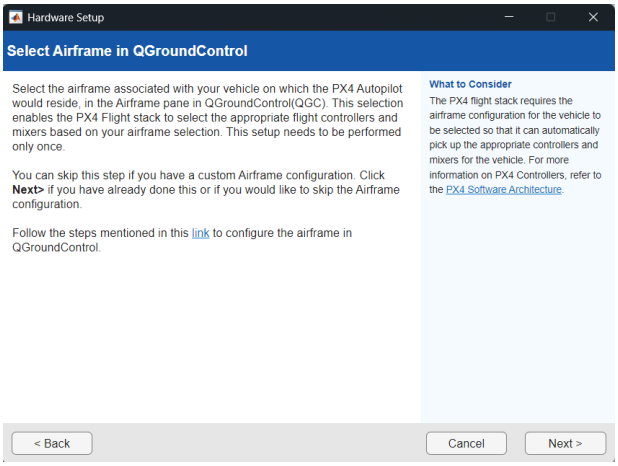

Figure B.11: Screenshot of the QGC airframe selection notice window.

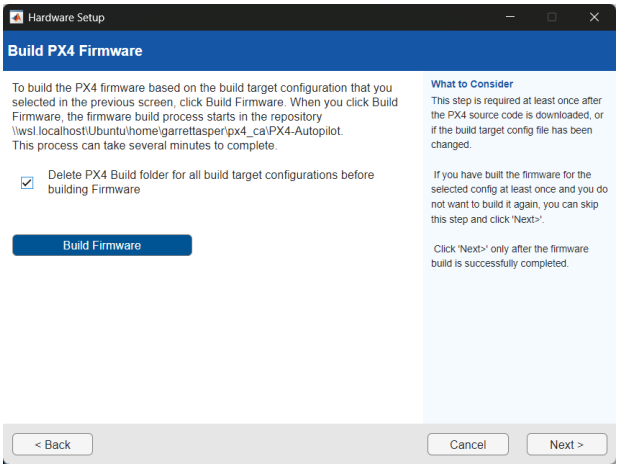

Figure B.12: Screenshot of the build PX4 firmware window.

Note: It is possible for the firmware build to fail if the automatic python installation shown in Figure B.3 did not properly install the `empy` python package. To see if this error caused the firmware build to fail, look for a message such as

```
AttributeError: module 'em' has no attribute 'RAW_OPT'
```

The python package '`empy`' used by WSL must be downgraded to version 3.3.4. To do so, run the following commands in the WSL window:

```
pip uninstall em
pip uninstall empy
pip install empy==3.3.4
```

After running these lines of code, the error will no longer occur when building PX4 through the Hardware Setup screens. Re-launch the screens, and attempt another build.

15. A green checkmark will appear in the window when the firmware has built successfully as shown in Figure B.13.

16. At this time, ensure the autopilot is plugged into the computer so that the Hardware Setup screens recognize the appropriate COM port of the PX4 autopilot's bootloader. Alternatively, the user may manually enter the COM port. Once the COM port has been selected, click "Upload Firmware" as shown in Figure B.14.

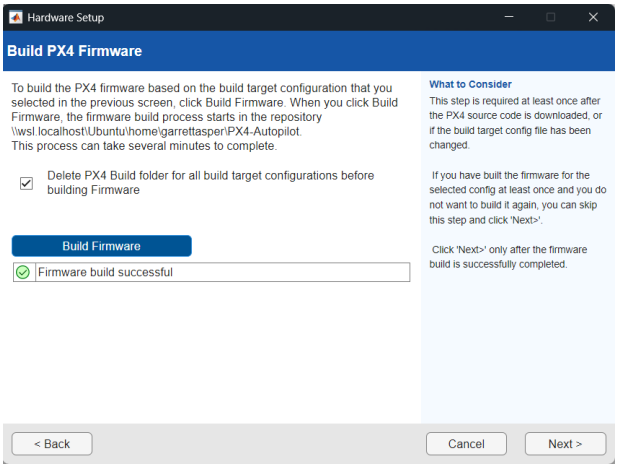

Figure B.13: Screenshot of the successful build PX4 firmware window.

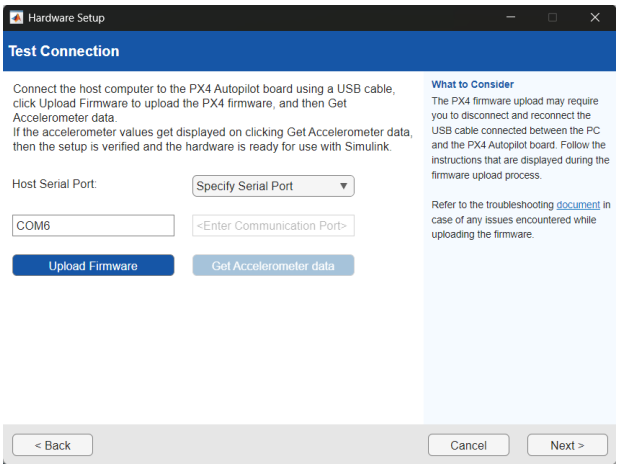

Figure B.14: Screenshot of the firmware upload window.

17. Next, a dialogue box will appear as shown in Figure B.15. Unplug the autopilot, click "OK," and reconnect the autopilot quickly to begin the firmware deployment process.

18. Monitor the progress of the firmware deployment in the MATLAB® command window as shown in Figure B.16 to ensure upload progress does not stall.

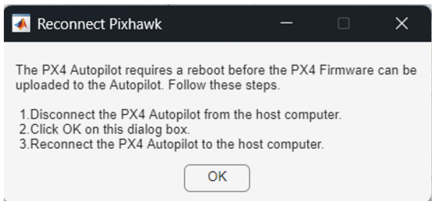

Figure B.15: Screenshot of the Pixhawk reboot prompt window.

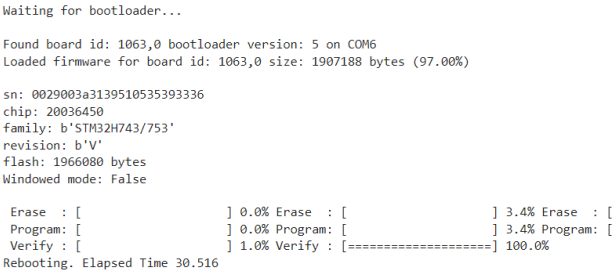

Figure B.16: Screenshot of the firmware deployment status in the command window.

19. Once the firmware has deployed successfully, a green checkmark will appear as shown in Figure B.17. Proceed to the final screen.

20. The hardware setup process is now complete. Optionally select "Show examples for support package" as desired, and click "Finish" as shown in Figure B.18.

At this point, the base PX4 firmware has been built and flashed to the autopilot. The user may proceed to Simulink® to explore the functionality of the UAV Toolbox as described in Section A.

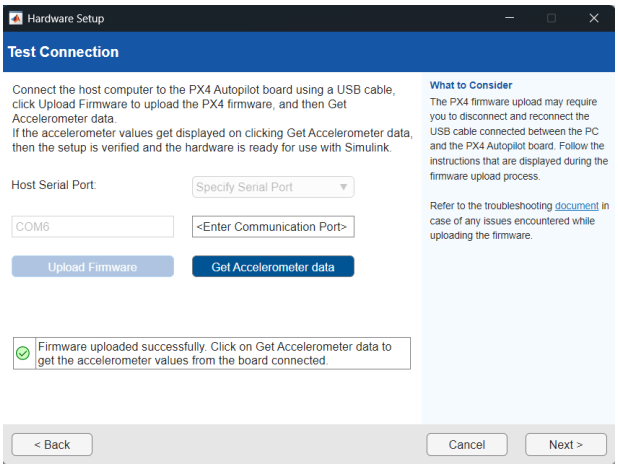

Figure B.17: Screenshot of the firmware upload success window.

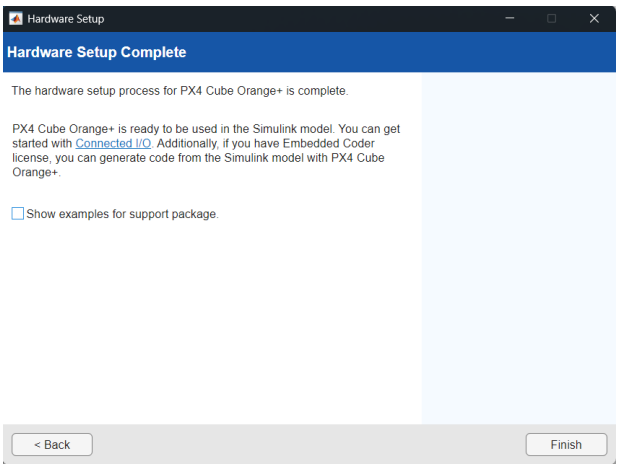

Figure B.18: Screenshot of the hardware setup complete window.

B.3 Optional Modifications to PX4

The research conducted in References [8, 10, 11, 12, 13, 20, 21]has required a variety of research-enabling modifications to the PX4 base code. This section presents the desired research functionality, an explanation of the enabling changes made to the source code, and open-source files available in Reference [9] to serve as example modifications suitable for the R2025a version of the UAV Toolbox. All source code modifications in this section are applied to the PX4 base-code repository cloned using the instructions in Section B.2.2 under the Linux path

```
/home/<WSL_username>/<px4_repo_name>/PX4-Autopilot
```

Note: Any change to the PX4 base code requires recompilation via the Hardware Setup screens (Section B.2.3) before re-testing or re-deploying the firmware from Simulink®.

Quick Start: Throughout this section, "Quick Start" boxes such as this one will summarize the exact files to use from Reference [9] and their destination file path within

the PX4 base code repository.

B.3.1 Custom Logging

To record internal Simulink® signals into the PX4 ULog file saved during flight, a user can define a custom uORB message and register it with the `msg` build system in PX4. For this tutorial, a topic to store example flight control system (FCS) signals, given by `FcsSignals`, will serve as the name of the custom uORB message added to the ULog file. The steps below outline this process.

1. Create a text file named `FcsSignals.msg` and save it under the Linux path `<px4_repo_name>/PX4-Autopilot/msg`. Each line in the file declares a field saved under the `FcsSignals` uORB topic. For example, adding the line

   ```
   float32[3] attitude
   ```

   allocates a 3×1 vector named `attitude` in the `FcsSignals` uORB topic.

2. Append "`FcsSignals.msg`" to `msg/CMakeLists.txt`, which serves as the list of uORB topics built during firmware compilation.

3. On the autopilot SD card, create the directory `etc/logging` and place a `logger_topics.txt` file that includes

   ```
   fcs_signals 0
   ```

   to ensure the custom topic is captured at the logger's default rate for that topic.[2]

[2] An example `logger_topics.txt` used on CZ-150 for system identification also enabled topics such as `actuator_motors`, `vehicle_attitude`, `sensor_combined`, etc.

Quick Start: Copy `FcsSignals.msg` into `<px4_repo_name>/PX4-Autopilot/msg`. Replace or update `<px4_repo_name>/PX4-Autopilot/msg/CMakeLists.txt` to register the new message. On the SD card used by the autopilot, create `etc/logging` (if needed) and add a file called "`logger_topics.txt`" containing at least the line `fcs_signals 0`.

Example Files in Reference [9]:

- `Custom Logging/ULog/FcsSignals.msg` - Example uORB message definition to store internal Simulink® data
- `Custom Logging/ULog/CMakeLists.txt` - CMake file that defines `FcsSignals` as a uORB message to be built
- `Custom Logging/SD Card/CZ150/logger_topics.txt` - Example definition file to specify logged uORB messages

4. Once the files are placed in the PX4 repository, complete the Hardware Setup screens process outlined in Section B.2.3 to successfully build the new uORB message into PX4.

5. Open the Simulink® diagram and place a "PX4 ULog" block shown in Figure B.19.

Figure B.19: Example "PX4 ULog" block used to write to `FcsSignals` in Simulink®.

6. Open the block parameters, and select the custom logger uORB topic `FcsSignals`.

7. Route any required signals to the block to ensure the topic contains the desired data.

B.3.2 Custom Telemetry Stream

A custom telemetry stream allows users to port signals internal to the Simulink® logic over an existing telemetry connection to QGroundControl. The telemetry stream uses the standardized MAVLink communication protocol to stream default data. The steps in this section outline how to alter the PX4 code to enable the `DEBUG_FLOAT_ARRAY` MAVLink message in the firmware. Additionally, this section describes how to publish to the `DebugArray` uORB message in Simulink to show the data in the `DEBUG_FLOAT_ARRAY` MAVLink message in QGC. Finally, the process of porting MAVLink data to a custom GCS layout in Simulink® is presented.

Enabling the `DEBUG_FLOAT_ARRAY` MAVLink Message

The `DEBUG_FLOAT_ARRAY` MAVLink message must be enabled in the MAVLink main file in PX4. The steps to follow outline the necessary changes.

1. Navigate to the `<px4_repo_name>/PX4-Autopilot/src/modules/mavlink/mavlink_main.cpp` file.

2. Under the "`case MAVLINK_MODE_NORMAL:`" line, make the following changes (shown in red)

```
configure_stream_local("ATTITUDE", 12.0f);
configure_stream_local("ATTITUDE_QUATERNION", 5.0f);
```

Insert the line below after "`configure_stream_local("DISTANCE_SENSOR", 0.5f);`"

```
configure_stream_local("DEBUG_FLOAT_ARRAY", 10.0f);
```

Under the "`#if !defined(CONSTRAINED_FLASH)`" statement delete the following line

```
configure_stream_local("DEBUG_FLOAT_ARRAY", 1.0f);
```

3. Under the "`case MAVLINK_MODE_CONFIG: // USB`", make the following addition to enable `DEBUG_FLOAT_ARRAY` while connected over USB

```
configure_stream_local("DEBUG_FLOAT_ARRAY", 50.0f);
```

Under the "`#if !defined(CONSTRAINED_FLASH)`" statement delete the following line

```
configure_stream_local("DEBUG_FLOAT_ARRAY", 50.0f);
```

Quick Start: Replace `mavlink_main.cpp` found under the Linux path `<px4_repo_name>/PX4-Autopilot/src/modules/mavlink` with the version that enables `DEBUG_FLOAT_ARRAY` either created with the instructions, or with the example file found in Reference [9] under the path `Telemetry/mavlink_main.cpp`.

Writing to the `DebugArray` uORB Topic

To send internal Simulink® signals over the telemetry connection, the user must write to the 58 element `data` variable in the `DebugArray` uORB topic. Writing to this uORB topic through Simulink®, as shown in Figure B.20, ensures the `DEBUG_FLOAT_ARRAY` MAVLink message is streamed over Telemetry and USB connections. This MAVLink message can then be accessed through QGC and ported into a custom ground control station (GCS) built in Simulink® as described in Section B.3.2.

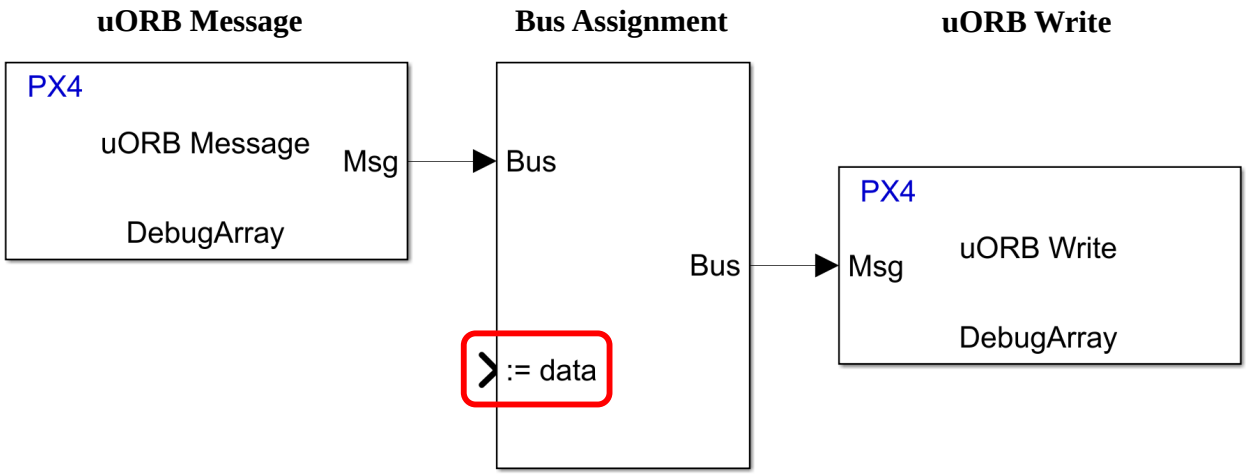

Figure B.20: Simulink® logic for sending a custom telemetry stream from the FCS.

Receiving Telemetry Data in QGroundControl and Simulink®

To receive the signals sent over the telemetry connection, the user must first open QGC. Then, navigate to "Application Settings," and then "MAVLink." Ensure that MAVLink forwarding is enabled and note the IP address (host name). Next, there must be a successful connection between the telemetry radio connected to the GCS computer and the telemetry radio onboard the vehicle. Alternatively, a hard-wired USB connection can be used.

With QGC open, create a blank diagram in Simulink®. Using a "UDP Receive" block and a "MAVLink Deserializer" block, replicate the logic shown in Figure B.21. In the "UDP Receive" block, set the "Local IP port" to match the IP address given by QGC. In the block parameters for the "MAVLink Deserializer" block, select the "`DEBUG_FLOAT_ARRAY`" MAVLink topic. Using a bus selector, select the "data" variable to access the vector of 58 signals sent from the vehicle.

The signals can then be used throughout Simulink® in gauges, scopes, and displays. To view live signals in the Simulink® GCS, run the diagram.

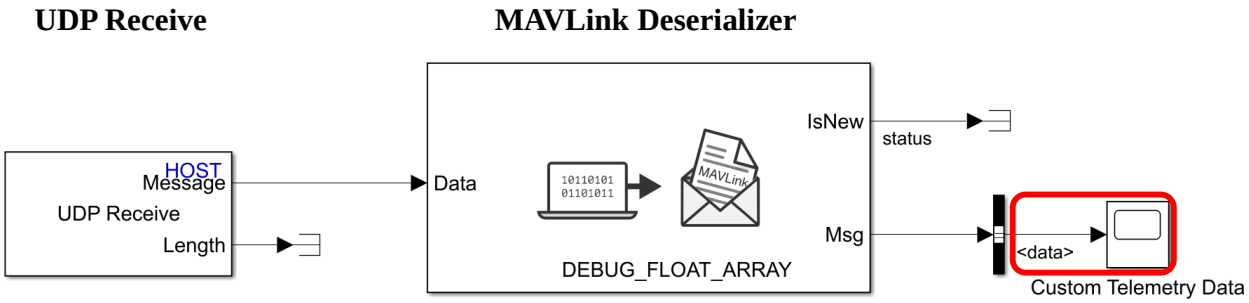

Figure B.21: Simulink® logic for receiving a custom telemetry stream from the FCS.

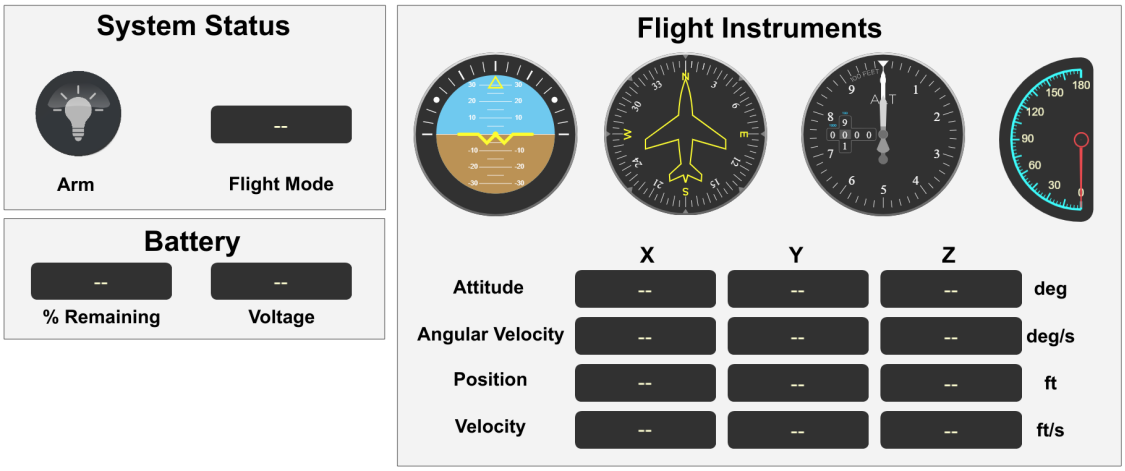

Figure B.22: Simulink® logic to view and analyze custom ground control signals in real time.

B.3.3 Custom Parameters

Custom PX4 parameters enable critical research functions, including the ability to tune control gains, switch between multiple programs, or activate custom flight safety logic, to name a few. This section explains how to manually add parameters such that they are built in the PX4 base code.

Follow the steps below to add custom parameters in the PX4 firmware

1. Navigate to the `module.yaml` file in the Linux path below

   ```
   <px4_repo_name>/PX4-Autopilot/src/modules/battery_status/module.yaml
   ```

2. Add desired parameters beneath the existing parameters (do not remove the existing battery parameters.) Refer to the example parameter code below that shows the definition of a parameter group `CUSTOM` under which the example `CUSTOM_PARAM_1` is defined as

   ```
   group: CUSTOM
     definitions:
       CUSTOM_PARAM_1:
         description:
           short: First Custom Param
           long: |
             Custom parameter that can be used as desired
         type: float
         decimal: 3
         default: 5
         min: 0
         max: 500
   ```

Quick Start: Place the `module.yaml` file in Reference [9] under the path `Parameters/module.yaml` into the Linux path

```
<px4_repo_name>/PX4-Autopilot/src/modules/battery_status/module.yaml
```

This example file should be modified to add all desired parameter functionality.

B.3.4 Flash Optimization

The flash memory on the CubePilot Cube Orange flight computer is limited, and complex Simulink® diagrams can require more flash memory than the space available when using the

unmodified PX4 source code. To reduce flash memory usage from the default PX4 modules, certain PX4 modules that are not required can be disabled. The following modifications were made to disable certain unnecessary modules for the `fixedwing` build target selected in the Hardware Setup screens. As the user has the option to choose between `default`, `fixedwing`, `multicopter`, and `VTOL` build targets, "`<build_target>`" will be used to represent the options generally.

1. Navigate to

   ```
   <px4_repo_name>/PX4-Autopilot/boards/cubepilot/<pixhawk_board_name> ...
     /<build_target>.px4board.original
   ```

2. Modify the following lines using "`n`" to disable the following modules

   ```
   CONFIG_DRIVERS_CAMERA_CAPTURE=n
   CONFIG_DRIVERS_CAMERA_TRIGGER=n
   CONFIG_COMMON_DISTANCE_SENSOR=n
   CONFIG_DRIVERS_DSHOT=n
   CONFIG_COMMON_OPTICAL_FLOW=n
   CONFIG_MODULES_CAMERA_FEEDBACK=n
   ```

Quick Start: Place the `fixedwing.px4board.original` file in Reference [9] under the path `Telemetry/mavlink_main.cpp` into the CubeOrange folder under the Linux path

```
<px4_repo_name>/PX4-Autopilot/boards/cubepilot/cubeorange
```

This example file is valid for the CubePilot Cube Orange autopilot board when using the `fixedwing` target.

Note: The user should ensure any PX4 module disabled is not required for their hardware or attached peripherals.

Appendix C

DroneCAN Setup Guide for Use with PX4

The term "UAVCAN" has historically referred to a modular, decentralized communication protocol designed for aerospace and robotic applications. However, to clarify and address some of the confusion around its versions, the protocol has been forked into two distinct projects: DroneCAN and Cyphal. DroneCAN refers to the protocol previously known as UAVCAN v0, while Cyphal was previously known as UAVCAN v1. The choice of hardware and firmware for a given vehicle depends significantly on which of these two protocols is implemented.

DroneCAN is compatible with PX4 v1.14 and is currently supported within the PX4 developer community. Cyphal is seeing a gradual increase in adoption within both ArduPilot and PX4 ecosystems, with expectations of wider support in future releases. However, PX4 v1.14 does not support Cyphal [18]. For a detailed comparison of DroneCAN and Cyphal, see Reference [14].

This appendix describes the setup process for the Zubax Myxa ESC, detailing its integration with PX4 v1.14 and the MathWorks® UAV Toolbox for direct RPM command execution on uncrewed aerial vehicles (UAVs). Throughout this tutorial, the Rotorcraft for Advanced Controls and Estimation Research (RACER) quadcopter, shown in Figure C.1, was used as the integration testbed.

Figure C.1: The RACER vehicle at the Nonlinear Systems Laboratory (NSL).

C.1 Hardware and Software

The hardware used in this tutorial is outlined in Table C.1.

Table C.1: Summary of hardware components.

Manufacturer	Component
CubePilot	Cube Orange/Cube Orange+
Zubax	Myxa A2 ESC
Zubax	CANFace CF2 “Babel-Babel”
UCANPHY	Micro Patch Cable
UCANPHY	Micro Termination Plug
KDE Direct	2315XF 965Kv Motor

Table C.2 presents the software and version used for the setup process, along with consider-

ations by use case.

Table C.2: Summary of software and versions used.

Software	Version	Purpose and Notes
Kucher	v1.1.0	Used for motor parameter identification. Interfaces with the ESC over USB [2].
DroneCAN GUI Tool	v1.2.25	Enables communication with multiple ESCs by accessing all network nodes [1].
QGroundControl	v4.2.8	Interfaces with PX4 to configure DroneCAN node IDs with the PX4 airframe [4].
UAV Toolbox	R2025a	Facilitates the building of a custom flight controller and direct RPM command sending. Optional unless developing a custom flight controller [10, 17].

Finally, the component, firmware, and version used are listed in Table C.3.

Table C.3: Summary of firmware and versions used.

Component	Version
CubePilot Cube Orange	PX4 v1.14.3
Zubax Myxa ESC	Telega v0.6

C.2 Zubax ESC Configuration

This section provides an overview of the sequential setup process to configure the proper parameters on the Zubax Myxa ESCs.

C.2.1 Rolling Back Firmware from Telega v1 to v0

As of March 2024, all Zubax Myxa ESCs ship with Telega v1 (Cyphal-compatible) firmware unless a special request is sent to have them ship with Telega v0 (DroneCAN-compatible) firmware. Rolling the firmware back from Telega v1 to Telega v0 is relatively simple and requires the installation of Yukon v2023.3.45 [6].

The ESC and Pixhawk must be powered to create the DroneCAN network. Next, attach the Babel-Babel to the network, launch Yukon, and navigate to Transports > CAN > SLCAN. Select the COM port of the Babel-Babel, and wait for the ESC node to appear under the "Monitor" menu. Next, download the binary file of the desired Telega v0 (DroneCAN-compatible) release. An application and compound binary files are present on the Zubax firmware repository [5]. For the RACER, the application version of Telega v0.6 was selected. In Yukon, select the "Firmware" button, and navigate to the `com.zubax.telega-1-0.6.cd758ab7.application.bin` file. This update will take several minutes to complete. If successful, the ESC will now appear under the DroneCAN tab of Yukon and can be accessed through Kucher or the DroneCAN GUI Tool.

Note: Firmware updates can only be conducted over the CAN1 port of the ESC.

C.2.2 Motor Identification

The motor identification process must be completed for each ESC and motor pair before integrating the hardware with PX4. The manufacturer's given parameters should be considered if the motor has parameters identified on the sheet shared in Reference [3]. In the case of the KDE Direct 2315XF motor, motor identification was completed in the lab using the Kucher software [2]. Kucher is only compatible with Telega v0. If the Zubax Myxa shipped

with Telega v1, the COM port will not be accessible. Refer to Section C.2.1 for instructions on rolling the firmware back to Telega v0.

> **Note:** Although Zubax recommends that Motor ID is performed with a completely unloaded motor, other users have found that a reduction in the `error_count` seen in the "`EscStatus`" uORB message from flight testing can be reduced if Motor ID and tuning are done with the propeller installed. The author found that it is best to conduct motor identification with a completely unloaded motor unless the `error_count` persistently increases during bench and flight testing. Then, motor identification should be redone with the propeller load attached.

The motor identification procedure below was derived from Reference [7].

1. Connect the motor to a power supply or the vehicle to a battery if the ESCs are already installed.

2. Connect the ESC to a computer using a direct USB connection.

3. Open Kucher

   > **Note:** Kucher is more stable when run on a Linux computer. Turning Bluetooth® off can alleviate some issues on Windows. If issues persist, use a Linux system.

4. Select the COM port of the ESC and click "Connect."

5. Define the number of motor poles using the `m.num_poles` parameter.

   > **Note:** The Kucher program expects the number of magnetic poles, not stator poles. The number of stator poles refers to the count of magnetic coils embedded in the motor, whereas the number of magnetic poles corresponds to the distinct magnetic field orientations. This is found by counting the number of detents (clicks)

felt when rotating the motor through one full revolution.

6. Define the maximum current using the `m.max_current` parameter.

7. Navigate to the "Motor Identification" subpanel and select "Resistance, Inductance, and Flux Linkage."

8. Launch the motor identification process. This step takes between three to five minutes and should be completed on an unloaded motor.

9. After identification is complete, highlight all parameters under the "`m`" list in the registers panel as shown in Figure C.2.

10. Right-click and select the "Read Selected" option.

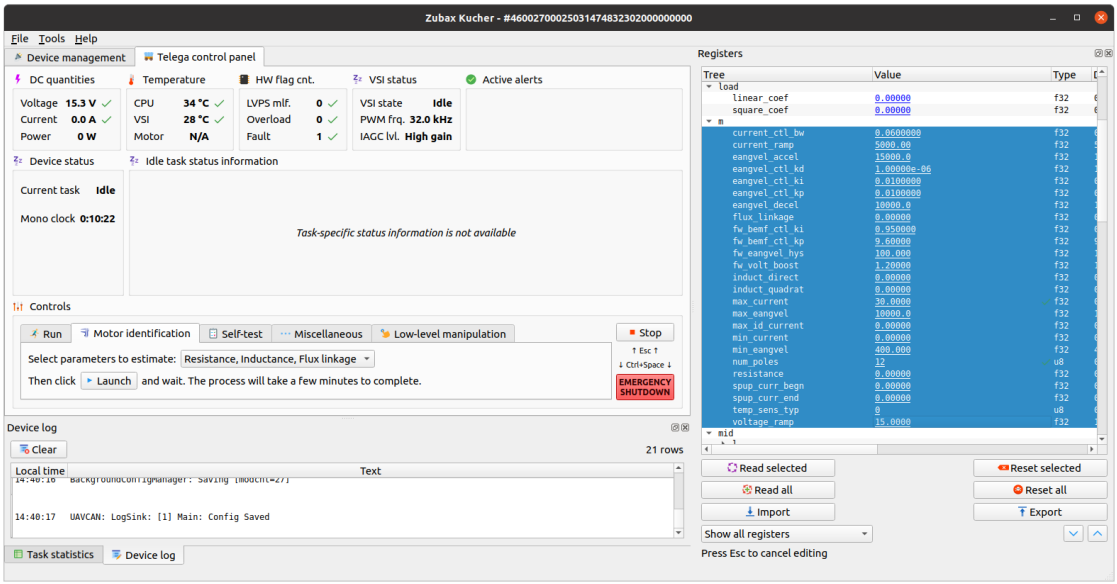

Figure C.2: Writing the motor ID parameters to the ESC using the Kucher interface.

The user has the option to use the same parameters for each motor on the vehicle or to run this procedure for each motor separately. Zubax recommends the user completes the Motor ID procedure on each motor unless motor manufacturer data is provided on the parameters.

C.2.3 ESC Index Enumeration

With all the ESCs connected together properly on the CAN network, as discussed in Reference [18], connect the network to the CAN1 port of the PX4 autopilot. The PX4 parameter `UAVCAN_ENABLE` must be set to "3," enabling all CAN actuators and sensors. It is important that the autopilot and ESCs are powered to access the CAN network for ESC index enumeration.

The Zubax Babel-Babel can be used to connect to a pre-existing DroneCAN network by connecting a CAN cable from the Babel 1 port to an available node, or CAN bus splitter. For the RACER, this was done using an I2C breakout board.

The following steps were completed to enumerate the ESC index for compatibility with PX4:

1. Power the vehicle including the autopilot and the motors.
2. Connect the Babel-Babel to the DroneCAN network via a CAN cable and to a computer via USB.
3. Launch the DroneCAN GUI Tool.
4. Select the USB Serial device listed in the drop-down menu, and initialize connection.
5. Click the checkmark next to "Local node configuration" to view information about the devices on the network.
6. Double click the first node and click "Fetch Parameters" to view the list of parameters.
7. Define the parameter for `uavcan.esc_index` as $n - 1$ for each ESC on the network where n corresponds to the ESC number in PX4.

After each ESC has been enumerated using the DroneCAN GUI Tool, the "Actuator" panel in QGroundControl can be used to define which ESC, n corresponds to the PX4 motor number for the airframe.

C.2.4 Parameters

Table C.4 shows the key parameters defined on the Zubax Myxa ESCs before flight.

Table C.4: The non-default parameters set for motor 1 using the DroneCAN GUI Tool for DroneCAN v0.6.

Parameter	Description	Defined by	Motor 1 Value
`uavcan.node_id`	UAVCAN node ID	Default	111
`uavcan.esc_ttl`	Deadman timeout switch	User	.7
`uavcan.esc_index`	Index of ESC on vehicle	User	0
`uavcan.esc_sint`	Status interval when running	User	0.01
`uavcan.esc_sintp`	Status interval when passive	User	0.01
`uavcan.esc_rcm`	Ratiometric Control Mode	User	1
`ctl.spinup_durat`	Spinup duration in seconds	User	1.5
`ctl.num_attempts`	Number of Attempts to Failure	User	1e7
`m.num_poles`	Number of magnetic poles on the rotor	User	14
`m.max_current`	Rated phase current	User	30.0
`m.max_id_current`	Maximum ID current	User	9.0
`m.min_current`	Minimum stable operating current	User	0.75
`m.spup_curr_begn`	Initial current setpoint for ramp	Motor ID	6.00
`m.spup_curr_end`	Final current setpoint for ramp	Motor ID	0.75
`m.flux_linkage`	Magnetic flux linkage	Motor ID	8.6783e-4
`m.resistance`	Phase resistance	Motor ID	7.0946e-2
`m.induct_direct`	Direct axis phase inductance	Motor ID	7.1384e-06
`m.induct_quad`	Quadrature axis phase inductance	Motor ID	7.1384e-06
`m.min_eangvel`	Minimum electrical angular velocity	User	732.58
`m.max_eangvel`	Maximum electrical angular velocity	User	7990.12
`m.eangvel_accel`	RPM control acceleration	User	20000
`load.square_coef`	Coefficient for load square	User	9.28e-10

C.2.5 Configuring RPM Tracking

Configuring the RPM tracking is a crucial step in ensuring that your UAV operates efficiently and responds accurately to control inputs. This section will outline the determination of minimum and maximum motor RPMs, which are essential for properly configuring the ESCs to track RPM setpoints throughout the flight envelope with its internal control loop. The process involves a series of steps, from conducting initial tests to programming the ESC with the derived parameters.

Initial Testing for Maximum RPM

The first step involves conducting a test to determine the maximum RPM the loaded motor can achieve at the minimum voltage per cell at which the UAV will be operated. In this example, the testing was conducted at 3.6 volts per cell. All RPM measurement information was taken from saving the `EscStatus` uORB topic to a ULog file. Once a maximum RPM is established, select a desired minimum RPM for your system. It is necessary to choose a non-zero minimum RPM, around 500, for example, to avoid stalling the motor and to ensure smooth operation across the entire range of speeds.

Script Use and Parameter Programming

Let RPM_{max} and RPM_{min} denote the maximum and minimum mechanical RPM of the motor, respectively. The number of magnetic poles is defined as p. The actuator output value $a \in [0, A_{\text{max}}]$, where $A_{\text{max}} = 8191$ is the standard maximum actuator output for a CAN-based electronic speed controller (ESC). The linear relationship between actuator

output and mechanical RPM is given by

$$\mathrm{RPM}(a) = \frac{a}{A_{\max}} \mathrm{RPM}_{\max} \tag{C.1}$$

The user specifies a proposed minimum RPM, $\mathrm{RPM}_{\min}^{\mathrm{prop}}$, from which an initial minimum actuator output is calculated as

$$a_{\min}^{\mathrm{init}} = \frac{\mathrm{RPM}_{\min}^{\mathrm{prop}}}{\mathrm{RPM}_{\max}} A_{\max} \tag{C.2}$$

The value (C.2) is rounded to the nearest integer to obtain the programmed minimum actuator output,

$$a_{\min} = \mathrm{round}\big(a_{\min}^{\mathrm{init}}\big) \tag{C.3}$$

Due to rounding, the actual minimum mechanical RPM that corresponds to $a_{\min}$ differs slightly from the proposed value. The actual value is computed as

$$\mathrm{RPM}_{\min}^{\mathrm{act}} = \frac{a_{\min}}{A_{\max}} \mathrm{RPM}_{\max} \tag{C.4}$$

The corresponding electrical angular velocities (eRPM) required for programming the Zubax Myxa ESC parameters are given by

$$\omega_{e,\max} = \frac{\pi p}{60} \mathrm{RPM}_{\max} \tag{C.5a}$$

$$\omega_{e,\min} = \frac{\pi p}{60} \mathrm{RPM}_{\min}^{\mathrm{act}} \tag{C.5b}$$

Equations (C.3)–(C.5) define the values that are programmed into QGC, Simulink® (via the UAV Toolbox), and the Zubax Myxa ESCs. Specifically, $a_{\min}$ and $A_{\max}$ are assigned to the

minimum and maximum actuator command parameters defined in the "Actuators" tab in QGC. $\text{RPM}_{\text{min}}^{\text{act}}$ and RPM_{max} define the constrained RPM range that can be commanded as a control input in the control logic created in Simulink®. The electrical angular velocities $\omega_{e,\text{min}}$ and $\omega_{e,\text{max}}$ are programmed to the ESC using the DroneCan GUI Tool as the `m.min_eangvel` and `m.max_eangvel` parameters, respectively.

> **Note:** The $\text{RPM}_{\text{min}}^{\text{act}}$ and RPM_{max} parameters are only relevant for mapping control inputs from a custom control logic in Simulink® if using the UAV Toolbox for firmware deployment. Users with stock PX4 implementations do not need to use these parameters, as the RPM is effectively constrained in the PX4 control architecture using the a_{min} and A_{max} assigned in QGC.

A MATLAB® script used to calculate the QGC, Zubax Myxa, and mechanical RPM range is given below.

```
%%%%%%%%%%%%%%%%%%%%%%%% USER DEFINED PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%
m_RPM_max = 10900; % maximum mechanical RPM from testing at 3.6 v/cell
m_RPM_min_proposed = 1000; % initial/proposed minimum RPM (this will change)
p = 14; % number of poles in the motor

%%%%%%%%%%%%%%%%%%%%%%%%%%%% CALCULATIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%
max_act_output = 8191; % standard maximum actuator_output for CAN ESC

% Calculate initial min_act_output based on proposed minimum RPM
min_act_output_initial = (m_RPM_min_proposed * max_act_output) / m_RPM_max;

% Round min_act_output to nearest integer
min_act_output_rounded = round(min_act_output_initial);

% Calculate the actual minimum RPM based on the rounded min_act_output
actual_m_RPM_min = (min_act_output_rounded * m_RPM_max) / max_act_output;

% Calculate the eRPM for the zubax parameters
eRPM_max = vpa((1/60)*pi*p*m_RPM_max);
eRPM_min = vpa((1/60)*pi*p*actual_m_RPM_min);

%%%%%%%%%%%%%%%%%%%%%%%%%%%% DISPLAY RESULTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp(' ')
disp(' ************************************************')
disp('Program this into QGC: (where # is ESC 1,2,3, etc.)')
fprintf('UAVCAN_EC_MIN#: %d\n', min_act_output_rounded);
fprintf('UAVCAN_EC_MAX#: %d\n', max_act_output);
disp(' ')
disp(' ************************************************')
disp(['Program this into the Simulink diagram as the new minimum RPM ' ...
'that is being sent: '])
fprintf('Actual Minimum RPM: %.10f\n', actual_m_RPM_min);
fprintf('Actual Maximum RPM: %.10f\n', m_RPM_max);
```

```
34  disp(' ')
35  disp(' ************************************************')
36  disp('Program these parameters on the Zubax Myxa ESCs: ')
37  fprintf('m.max_eangvel: %.10f\n', eRPM_max);
38  fprintf('m.min_eangvel: %.10f\n', eRPM_min);
```