AlphaLink Engineering GmbH
Ackerstraße 76, ACK384
13355 Berlin

Unmanned Aircraft Experimental System

The Flying Lab for Applied Flight Control and Flight Mechanics

Manual

Version: 1.0
Date: March 31, 2020

Terms and Conditions

The AlphaLink Unmanned Aircraft Experimental System (UAXS) is a set that consists of software and hardware for experimental purposes. The software is provided on a USB stick. The MATLAB/Simulink model may not be copied or redistributed in whole or part. One copy per set is allowed for internal use only. Publications that refer to the supplied MATLAB/Simulink model must always be made with reference to this manual.

Note: The current version of the UAXS is designed for computers with MS Windows as operational system and is tested with MATLAB R2019b.

Delivery Scope

Included in the delivery of an AlphaLink UAXS set are:

1. Airborne System (ZOHD Nano Talon EVO PNP, incl. battery),
2. Flight Control Computer (Pixhawk 4 Mini),
3. GPS Module (UBLOX NEO M8N GPS),
4. Airspeed Sensor (Holybro Digital 4525DO Sensor),
5. Telemetry Set (Pixhawk 4 433 Mhz 100 mW), and
6. USB Stick with PX4 Software, QGroundControl, Python, Matulog, Pyulog, MATLAB/Simulink Model, Linear State-Space Models, Manual, and Readme file.

The following components are currently available as add-ons to the AlphaLink UAXS set:

1. Remote Control (FrSky Taranis Q X7 RC Transmitter 2.4 GHz with 16 channel, incl. battery) and RC Receiver (FrSky R-XSR EU LBT),
2. LiDAR Sensor (Benewake TFmini Plus Micro LiDAR) and
3. Hardware-in-the-Loop Simulator (AlphaLink hardware and software).

Contents

1 Setup and Commissioning of the Airborne System

The PX4 software is already implemented on the Pixhawk 4 Mini flight control computer (Pixhawk). For sets with the remote control add-on, the system is ready to use.[1]

1.1 5-Step Setup

1. To install the wing, take off the upper caps of the plane. Then, stick the wing spar through its mount in the fuselage (Fig. 1).

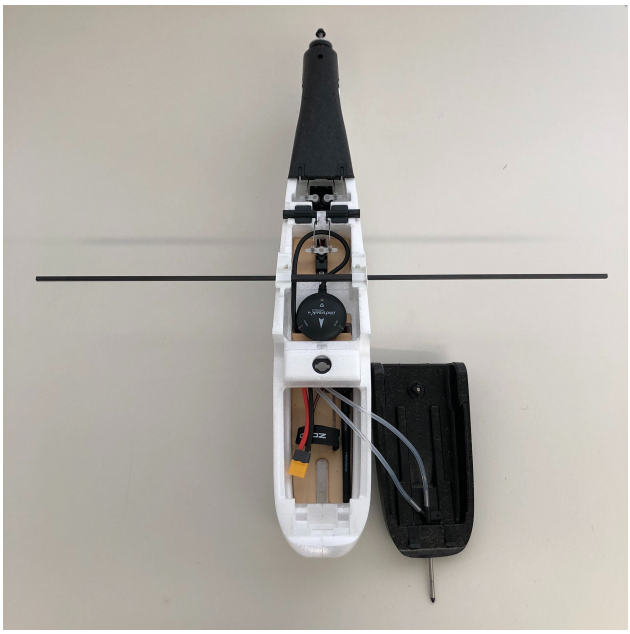


Figure 1: Installed Wing Spar.

2. Carefully push the wing halves onto the wing spar, until they lock into the hull. Be careful that the control mechanism for the aileron connects smoothly by tilting the ailerons for correct alignment (Fig. 2).

3. To install the V-tail, push in the tail halves with the magnets facing inwards and watch out again for correct alignment of the control mechanism (Fig. 3).

4. Stick the propeller onto the engine shaft with the markings facing forward. Then, tighten the nut with an 8 mm wrench (Fig. 4).

   Note: Only install the propeller shortly before the flying and take it off afterwards. Do not leave it on, when programming the Pixhawk. Do not touch the propeller, when the battery is connected!

5. Place the battery inside the front of the plane and secure it using the strap. Underneath the wing, there are two CG marks. Place a finger on each mark and balance the plane. The nose should slightly face downwards. If that is not the case, move the battery forward and check again. If the battery is already at the very front and the plane is still not balancing right, add some trim weight (not included).

[1] Otherwise, Sec. 1.2 describes the installation for a remote control and RC receiver.

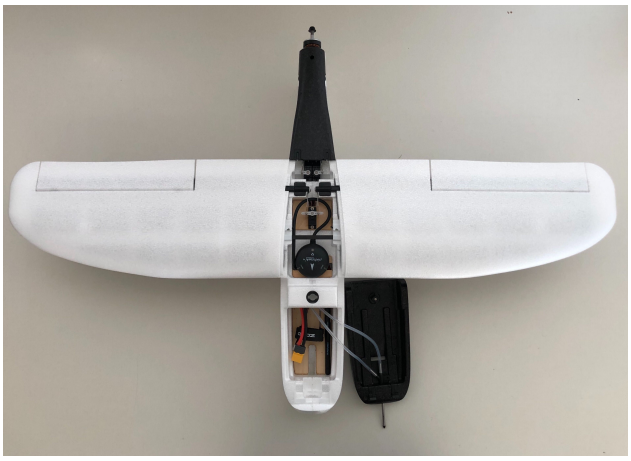


Figure 2: Installed Wing.

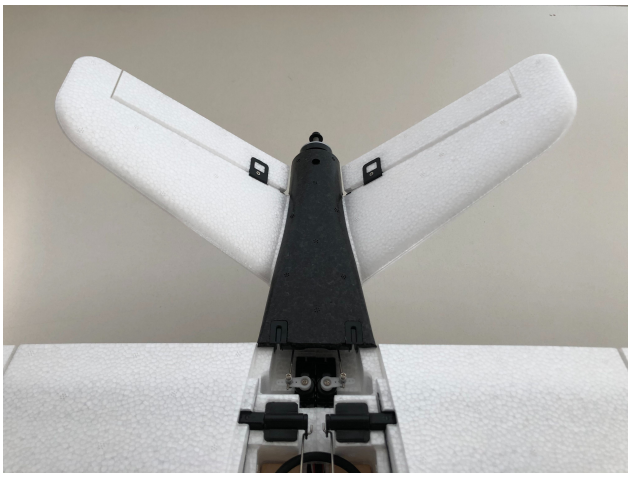


Figure 3: Installed V-Tail.

Figure 4: Installed Propeller.

1.2 Remote Control and RC Receiver

This procedure is only relevant for sets without the remote control add-on. Be aware, that only serial protocols like S.BUS, DSM or PPM are supported, but no PWM receivers. Further information about supported receivers can be found on the PX4 website.[2]
Installation of the remote control and the RC receiver requires 3 steps:

1. Connect the RC receiver to the Pixhawk. Figure 7 shows the hardware dependencies. From the delivered cables, choose one with a connector that fits into the `RC IN` port of the Pixhawk. Connect the other side of each cable to the receiver, according to the `RC IN port` table and the pin layout described in the manual of the receiver.

2. Set up the channels as described in the manual of the remote control. To use the setup with the pre-installed Direct Law model, choose the following channel layout:

   Channel 1: Thrust

   Channel 2: Elevator

   Channel 3: Aileron

   Channel 4: Rudder

   Channel 5: Switch 1

   Channel 6: Switch 2

   Channel 7: Switch 3

   Channel 8: Switch 4

   Be aware that the controls must be correct according to the flight mechanic algebraic sign. For this, typically the elevator channel has to be inverted. It may be necessary to adjust the channel range in the provided MATLAB/Simulink model (see Sec. 4.2).
   Figure 5 shows the configuration of the FrSky Taranis Q X7 remote control as an example.

3. Bind the receiver and the remote control according to their manuals. Supported receivers will be detected automatically by the Pixhawk.

[2] `https://docs.px4.io/v1.9.0/en/getting_started/rc_transmitter_receiver.html#compatible_receivers`.

Figure 5: Channel Layout of Exemplary Remote Control (Model: Taranis FrSky Q X7).

1.3 Flight Test Preparation and Take-Off

For a successful test flight, it is recommended to mind the following 7-point checklist before take-off:

- ☐ Set thrust on remote control to zero.
- ☐ Turn on remote control.
- ☐ Connect battery.
- ☐ Calibrate airspeed sensor (see Sec. 3.3).
- ☐ Restart Pixhawk. To do so, switch to the MAVLink NuttX console `NuttShell` (see Fig. 9) and enter `reboot`.
- ☐ Check that GPS signal is valid. This is done by enterin the `ekf2 status` command into the MAVLink NuttX console. Local Position needs to be `valid`.
- ☐ Test the correct operation of the control surfaces and the engine, while holding the plane safely.

For takeoff, an assistant should hold the plane with one hand from underneath. Then the pilot needs to give full throttle and the assistant throws the plane horizontally into a headwind (into the wind). When releasing the plane from the hand, the assistant should immediately move down the hand to avoid getting close to the spinning propeller.

After flying, disconnect the battery from the plane first, before turning off the remote control.

1.4 Safety Instructions

*Note: For the experimental purpose of the UAXS, the flight control system of the Talon Nano Evo model has been modified. Therefore it should only be controlled by **experienced pilots**; it may also require **higher-than-average operational speeds** to compensate for the increased mass of the whole airborne system.*

For safe aircraft operation, the instructions beneath shall be followed.

Battery: To charge the battery, use a LiPo suitable battery charger and set it to 3S (11.1 V). Generally use a charging current of 1.5 A. If the battery needs to be charged faster, you can set the charger to 3.1 A, but doing this regularly can reduce the battery life time.
Most chargers have a storage mode, where the battery is charged or discharged to about 40 to 50 percent of its capacity. This should be used, when not using the battery for a longer period.

Operation: Do not fly over people, close to power lines or buildings. Only fly, where it is allowed.
Be extra careful, whenever the propeller is installed and the battery is connected.
When the propeller is installed, always make sure that the remote control is turned on, before you connect the battery.
Only install the propeller shortly before the flying and take it off afterwards. Do not leave it on, when programming the Pixhawk. Do not touch the propeller, when the battery is connected.

Weather Conditions: Do not fly when it snows or rains. It is recommended not to fly, if wind gusts of more than 10 knots are forecasted. In cold conditions, the possible flight time decreases. Keep the battery warm before flying.

2 Aircraft and System Description

The UAXS set consists of the flight test vehicle, its linear flight dynamics models at four trim points, the flight control system, and associated software.

2.1 Aircraft Description

The Talon Nano Evo is a fixed-wing aircraft with V-tail. Table 1 lists the most important aircraft parameters. Three aerodynamic rudders and thrust are available as input variables (see Fig. 6). Left and right aileron are actuated by a common servo motor. Left and right V-tail rudder have different servo motors and, hence, can be deflected independently. A control allocation is made for consideration of conventional control surfaces andnis already implemented in the Direct Law model. It can be changed as required. Using one elevator input (η) and one rudder input (ζ), the following applies to the left (η_L) and right rudder (η_R) at the V-tail:

$$\begin{aligned} \eta_R &= \eta - \zeta \\ \eta_L &= \eta + \zeta \end{aligned} .$$

Table 1: Properties of Talon Nano Evo.

Property	**Dimension**	**Property**	**Dimension**
Wingspan	0.86 m	Wing Area	0.148 m^2
Aspect Ratio	5.1	Length	0.57 m
Take-Off Mass	0.65 kg	Battery Capacity	1550 mAh
Servo Motors	3 $\times$ 9 g metal gear	Motor	SunnySky 2204-1870KV
Propeller	6×3 (inch)	ESC	30 A, 5 V 2 A BEC

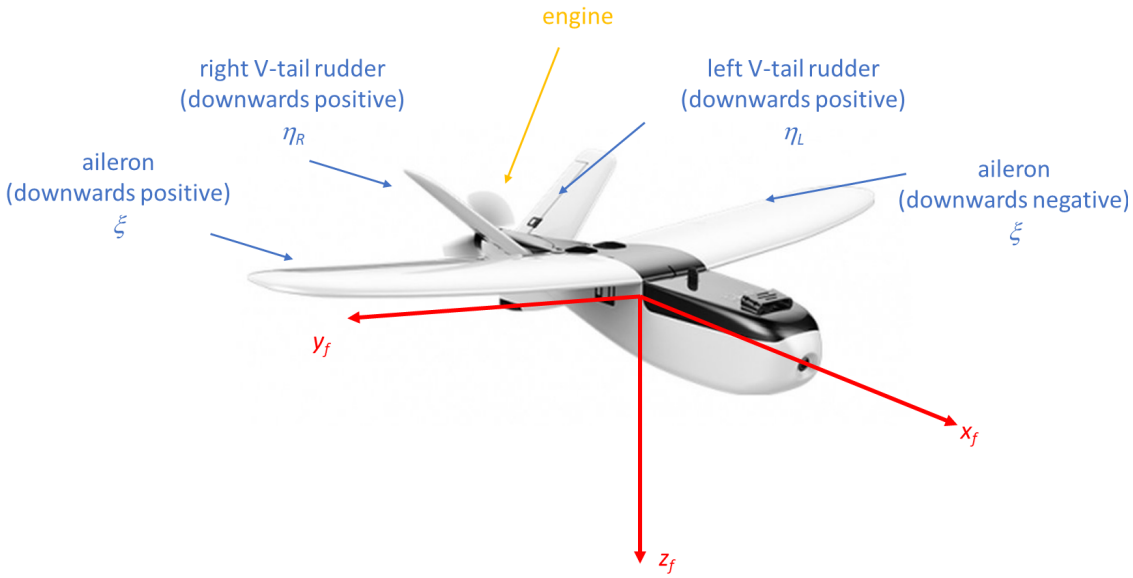


Figure 6: Illustration of Talon Nano Evo with Rudders and Engine in a Coordinate System.

The aerodynamic rudders are limited within $\pm 25^\circ$. The rotational speed of the engine is used as thrust command. The value is limited between 0 and 1. A value of 1 means maximum rotational speed while a value of 0 means full stop of the engine.

2.2 Linearised Flight Dynamics Models

A flight mechanical model for the aircraft was linearised at four trim points. The trim values are stored within MATLAB-Files. Table 2 provides the trim values as well as the corresponding file names. The trim data are available on the USB stick in the folder `/Matlab/Trimpoints`.

Table 2: Available Trim Points.

Nr.	Filename	Airspeed	Flight Path Angle	Angle of Attack	Thrust Lever	Elevator Deflection
1	`Trimm_V17_g0.mat`	$17\,\mathrm{m\,s^{-1}}$	0°	1.68°	0.91	-7.87°
2	`Trimm_V12_g0.mat`	$12\,\mathrm{m\,s^{-1}}$	0°	5.83°	0.66	-17.01°
3	`Trimm_V12_g10.mat`	$12\,\mathrm{m\,s^{-1}}$	10°	5.56°	0.77	-16.45°
4	`Trimm_V12_g-10.mat`	$12\,\mathrm{m\,s^{-1}}$	-10°	5.86°	0.52	-17.11°

The folder `/Matlab/Lin_Models` contains the linearised models for all trim points. For every trim point, three models are available:

1. one with consideration of longitudinal states only (MATLAB-Variable `G_long`),
2. one with consideration of lateral states only (MATLAB-Variable `G_lat`) and
3. one with consideration of both lateral and longitudinal states (MATLAB-Variable `G_full`).

In longitudinal motion, pitch rate (δq, unit: $1\,\mathrm{rad\,s^{-1}}$), angle of attack ($\delta\alpha$, unit: $1\,\mathrm{rad}$), indicated airspeed (δV_{IAS}, unit: $1\,\mathrm{m\,s^{-1}}$), and pitch angle ($\delta\Theta$, unit: $1\,\mathrm{rad}$) are used as state variables. In addition to these state variables, flight path angle ($\delta\gamma$, unit: $1\,\mathrm{rad}$) and ground speed (δV_{K}, unit: $1\,\mathrm{m\,s^{-1}}$) are available as output variables in longitudinal motion. Further, the elevator deflection ($\delta\eta$, unit: $1\,\mathrm{rad}$), thrust lever ($\delta\eta_{\mathrm{TL}}$, unit: 1), vertical wind speed (δw_{wg}, unit: $1\,\mathrm{m\,s^{-1}}$), and horizontal wind speed (δu_{wg}, unit: $1\,\mathrm{m\,s^{-1}}$) in the geodetic system as well as wind-induced pitch rate (δq_{wf}, unit: $1\,\mathrm{rad\,s^{-1}}$) in the body-fixed system are available as input and disturbance variables in longitudinal motion.

In lateral motion, yaw rate (δr, unit: $1\,\mathrm{rad\,s^{-1}}$), sideslip angle ($\delta\beta$, unit: $1\,\mathrm{rad}$), roll rate (δp, unit: $1\,\mathrm{rad\,s^{-1}}$) and bank angle ($\delta\Phi$, unit: $1\,\mathrm{rad}$) are used as state variables. In addition to these state variables, vertical acceleration (δa_{y}, unit: $1\,\mathrm{m\,s^{-2}}$) is available as an output variable in lateral motion. Further, aileron deflection ($\delta\xi$, unit $1\,\mathrm{rad}$), rudder deflection ($\delta\zeta$, unit $1\,\mathrm{rad}$) and east/west wind speed (δv_{wg}, unit: $1\,\mathrm{m\,s^{-1}}$) in the geodetic system as well as wind-induced roll rate (δp_{wf}, unit: $1\,\mathrm{rad\,s^{-1}}$) and wind-induced-yaw rate (δr_{wf}, unit: $1\,\mathrm{rad\,s^{-1}}$) in the body-fixed system are available as input and disturbance variables in lateral motion.

In the full model, all state, input, disturbance and output variables of both the longitudinal and lateral motion are available.

2.3 Flight Control System

The flight control system consists of the flight control computer, sensors and actuators. The flight control computer is the Pixhawk 4 Mini (Pixhawk). The flight control system is already configured and installed in the aircraft. The Pixhawk is already connected to the GPS module, the telemetry modem, the dynamic pressure sensor (I2C B) and the power distribution board. For sets with the LiDAR sensor add-on, the sensor is connected to the UART interface. If this device is not used, any other device that has a UART interface can be connected to the UART interface of the Pixhawk. Additionally, a CAN bus interface is available. For sets with the hardware-in-the-loop simulator add-on, this CAN bus can be used to interact with the provided hardware (additional, external box). Alternatively, the UAVCAN can be used with the CAN interface. Furthermore, an RC receiver (S.BUS, DSM or PPM) can be connected (see Sec. 1.2). For sets with remote control add-on, an S.BUS RC receiver is already connected to this interface. Figure 7 shows the interfaces on the Pixhawk and the pin assignment for the previously mentioned connectors.

The PWM signals for the motor controller and servomotors as well as other interfaces are located on the opposite side of the Pixhawk. In total, 8 PWM servo outputs are available (main out). In addition to the main out pins, there is a PPM RC input port for PPM RC receivers, an analog input (ADC) and four dedicated PWM/capture (CAP) inputs. However, there are no drivers for the PWM/CAP inputs in the PX4 software. Figure 8 shows the interfaces on the Pixhawk and the pin assignment for PWM out, PPM, ADC, and CAP.

One side of the Pixhawk contains a USB port and an FMU debug port. The USB port is already connected with an extension cable, which ends in the front of the aircraft. With this USB cable, the software can be uploaded to the flight control computer. Furthermore, a power supply is provided by this cable. For simple system tests no battery needs to be connected to the power board.

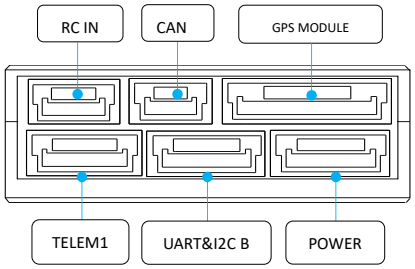
The other side of the Pixhawk contains the slot for the SD card. An SD card with a storage capacity of 8 GB is already included in the set. However, other SD cards can also be used.[3]

Three servo motors and the motor are used as actuators. The actuators are already connected to the Pixhawk.

In addition to the standard on-board sensors of the PixHawk, a dynamic pressure sensor and GPS are included in the set. The dynamic pressure sensor is of model MS4525DO. A pitot tube is already built in the airframe and connected to the sensor. The sensor is connected to the Pixhawk via I2C. The maximum measuring range is $6894.76\,\mathrm{Pa}$. The resolution is $0.74\,\mathrm{Pa}$. The accuracy is in the range of $\pm\ 17.23\,\mathrm{Pa}$ (at $25\,^{\circ}\mathrm{C}$). The dynamic pressure sensor is pre-calibrated by the original manufacturer. It is recommended to perform a new calibration. The GPS module is of type UBLOX M8N. It is connected to the Pixhawk via the UART interface. The geographical position is obtained when the blue LED turns on and off.[4]

[3] For an SD card other than the provided one, it is recommended to copy the folder `/etc/` from the provided SD card to the new one.

[4] For initial connections of the battery to the airborne system, a median time of $\approx 30\,\mathrm{s}$ was observed until GPS connection was successfully established (cold start). Due to the built-in battery of the GPS module, this time may be significantly reduced for system restarts.



RC IN port

Pin	Signal	Volt
1(red)	VDD_5V_SBUS_RC	+5V
2(black)	SBUS*	+3.3V
3(black)	RSSI**	+3.3V
4(black)	VDD_3V3_SPEKTRUM	+3.3V
5(black)	GND	GND

*Connect SBUS or DSM/Spektrum receivers signal wire connect here.

**Sends the RC signal strength info to autopilot.

UART & I2C B * ports

Pin	Signal	Volt
1(red)	VCC	+5V
2(black)	TX (out)	+3.3V
3(black)	RX (in)	+3.3V
4(black)	SCL2	+3.3V
5(black)	SDA2	+3.3V
6(black)	GND	GND

*A spare port for connecting sensors supporting serial communication or I2C e.g. a second GPS module can be connected here.

TELEM port *

Pin	Signal	Volt
1(red)	VCC	+5V
2(black)	TX (out)	+3.3V
3(black)	RX (in)	+3.3V
4(black)	CTS (in)	+3.3V
5(black)	RTS (out)	+3.3V
6(black)	GND	GND

CAN port

Pin	Signal	Volt
1(red)	VCC	+5V
2(black)	CANH	+3.3V
3(black)	CANL	+3.3V
4(black)	GND	GND

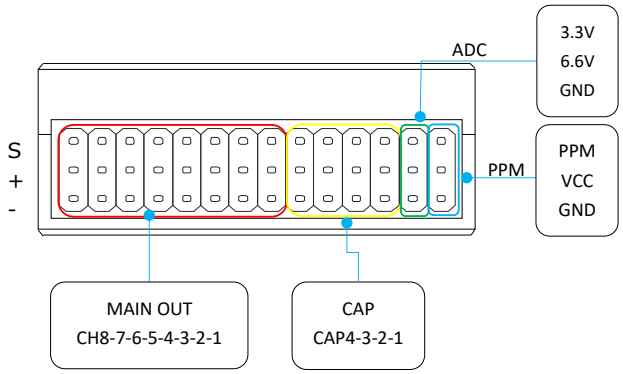
GPS MODULE ports

Pin	Signal	Volt
1(red)	VCC	+5V
2 black)	TX (out)	+3.3V
3(black)	RX (in)	+3.3V
4(black)	SCL1	+3.3V
5(black)	SDA1	+3.3V
6(black)	SAFETY_SWITCH	+3.3V
7(black)	SAFETY_SWITCH_LED	+3.3V
8(black)	VDD_3V3	+3.3V
9(black)	BUZZER	+3.3V
10(black)	GND	GND

POWER

Pin	Signal	Volt
1(red)	VCC	+5V
2 black)	VCC	+5V
3(black)	CURRENT	+3.3V
4(black)	VOLTAGE	+3.3V
5(black)	GND	GND
6(black)	GND	GND

Figure 7: Pin Layout of Pixhawk 4 Mini for RC In, CAN, GPS, Telemetry, UART, I2C, and Power. *Source*: Holybro.



MAIN OUT

Pin	Signal	Volt	+	-
1	FMU_CH1	+3.3V	VDD_SERVO	GND
2	FMU_CH2	+3.3V	VDD_SERVO	GND
3	FMU_CH3	+3.3V	VDD_SERVO	GND
4	FMU_CH4	+3.3V	VDD_SERVO	GND
5	FMU_CH5	+3.3V	VDD_SERVO	GND
6	FMU_CH6	+3.3V	VDD_SERVO	GND
7	FMU_CH7	+3.3V	VDD_SERVO	GND
8	FMU_CH8	+3.3V	VDD_SERVO	GND

CAP

Pin	Signal	Volt		
1	FMU_CAP1	+3.3V	+5V	GND
2	FMU_CAP2	+3.3V	+5V	GND
3	FMU_CAP3	+3.3V	+5V	GND
4	TIM5_SPARE_4	+3.3V	+5V	GND

PPM

Pin	Signal	Volt
S	PPM	+3.3V
+	VCC	+5V
-	GND	GND

ADC

Pin	Signal	Volt
3.3V	ADC1_SPARE_1	+3.3V*
6.6V	ADC1_SPARE_2	+6.6V*
GND	GND	GND

* WARNING: Sensors connected to this pin should not send a signal exceeding this voltage!

Figure 8: Pin Layout of Pixhawk 4 Mini for Main (PWM) Out, PPM, ADC, and CAP. *Source*: Holybro.

A telemetry modem is used for communication with the ground station (QGroundControl software). It is already installed in the aircraft and connected to the flight control computer. It also uses a UART interface. The PX4 software uses the MAVLink protocol for data transmission. The transmission frequency is 433 MHz and the maximum transmission power is 100 mW. According to the original manufacturer, distances of up to 300 m are possible. A second telemetry modem (included in the set) must be connected to the user's computer via USB (FT230X Basic UART to USB).

For sets with the remote control add-on, a FrSky Q X7 RC transmitter 2.4 GHz with 16 channel and a FrSky R-XSR EU LBT receiver are included in the set. The remote control is already configured. The receiver is already connected to the Pixhawk via the S.BUS interface.

For sets with the LiDAR sensor add-on, the Benewake TFmini Plus Micro LiDAR is included. It allows the measurement of height above ground. The maximum measurable height is 12 m, the lowest measurable height is 30 cm. It is connected to the flight control computer via the UART interface. The instrument is already installed in the aircraft and connected to the Pixhawk.

Further information on the flight control computer can be found on the PX4 website.[5] Information about the on-board sensors of the Pixhawk and other technical details are also provided there.

[5] https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4_mini.html.

3 Configuration of Pixhawk 4 Mini Flight Control Computer

3.1 Configuration and First Steps

The following steps are necessary to setup up a Windows computer for compiling and uploading the PX4 software. Other operational systems are currently not supported.

1. Open the file `toolchain_installer.msi` provided on the USB Stick and follow the instructions. (Do NOT check the "Clone PX4 Repository and Start Simulation" checkbox). Default installation folder is `C:\PX4`; if you change this avoid blanks in the directory name.
2. Switch to the directory, where the toolchain was installed. Open the file `run-console.bat`. (This Cygwin console is a Linux environment with tools for compiling the flight stack). It runs an initial setup. Close it afterwards.
3. Copy the folder `Firmware` from the USB Stick into the `home` folder of the current directory.
4. Open the file `QGroundControl-installer.exe` provided on the USB Stick and follow the instructions.

3.2 Using QGroundControl

QGroundControl (QGC) is a tool to access the running Pixhawk. It must be connected to the user's computer via USB cable or a telemetry modem. Be aware that if multiple telemetry modems are active, it is undefined which one connects to the ground station. The QGC interface is shown in Fig. 9 with indications of relevant icons and how to access NuttShell.

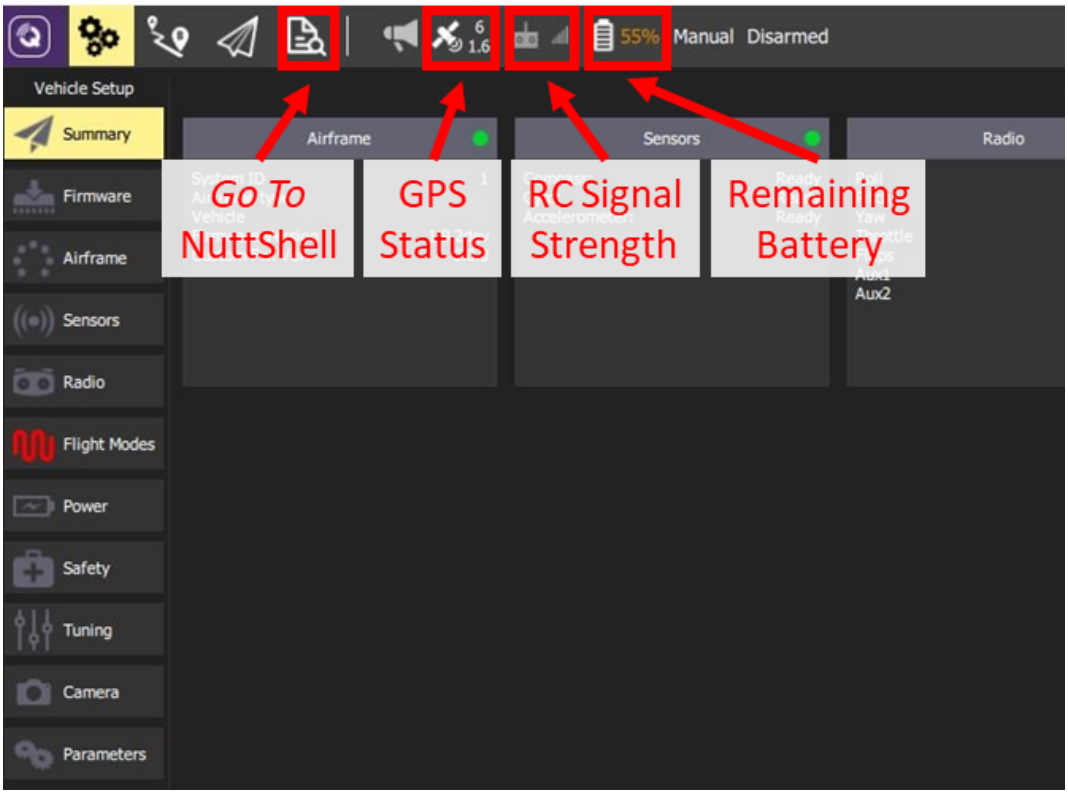


Figure 9: QGroundControl Interface.

The QGC interface gives a full overview of the vehicle's state. During flight tests, it is particularly important to observe the battery status displayed in the top line. Further, the icons for GPS status and RC signal strength give helpful information. Error messages may occur due to incompatibilities of different program versions and manipulations of the flight stack. All of these can be ignored, as they will not affect the functionality of the system.[6]
NuttShell (NuttX console) is another important tool. It is used to access the underlying operating system. During flight tests, flight logging will be initialised with the boot sequence, but it can be stopped and restarted from the NuttShell (see Sec. 3.6).

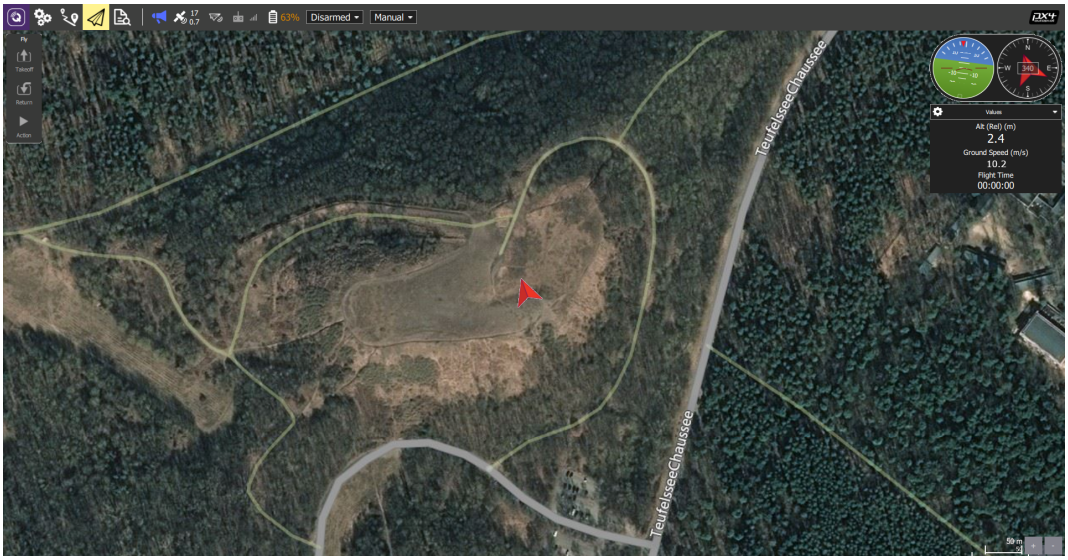


Figure 10: QGroundControl during a Flight.

Figure 10 shows QGC during a flight. On the right, vehicle attitude, course, relative altitude and ground speed are shown.

3.3 Sensor Calibration

For calibrating the sensors, remove the propeller, power the plane and connect QGC to it. Access the NuttShell (see Fig. 9) and enter `commander start`. After doing this, the motor may start spinning spontaneously. Change to the `Sensors` option in the `Vehicle Setup` view (see Fig. 11) and go through all calibration steps by following the instructions.

3.4 PX4 Software

A brief overview of the Pixhawk architecture with the PX4 Software is shown in Fig. 12.
The core is an STM32 microcontroller running NuttX, which is a minimised real-time operating system. Input and output drivers enable the communication with all used sensors and actuators. An extended Kalman filter is included to estimate the attitude of the plane. This is the general environment, in which the MATLAB/Simulink model operates. Further information about the original flight stack can be found on the PX4 website.[7]

[6] *Waiting For Vehicle Connection* is displayed in red letters on the top right only if the Pixhawk is not connected at all.

[7] `https://dev.px4.io/v1.9.0/en/concept/architecture.html#flight-stack`.

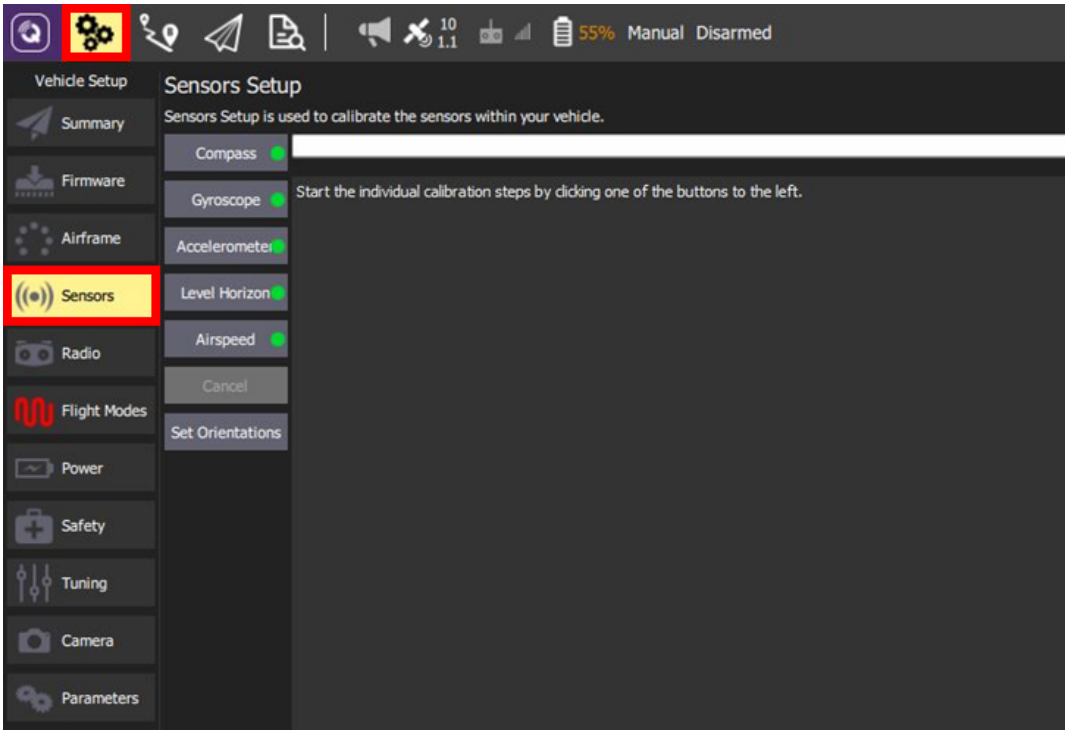


Figure 11: QGroundControl Calibration Interface.

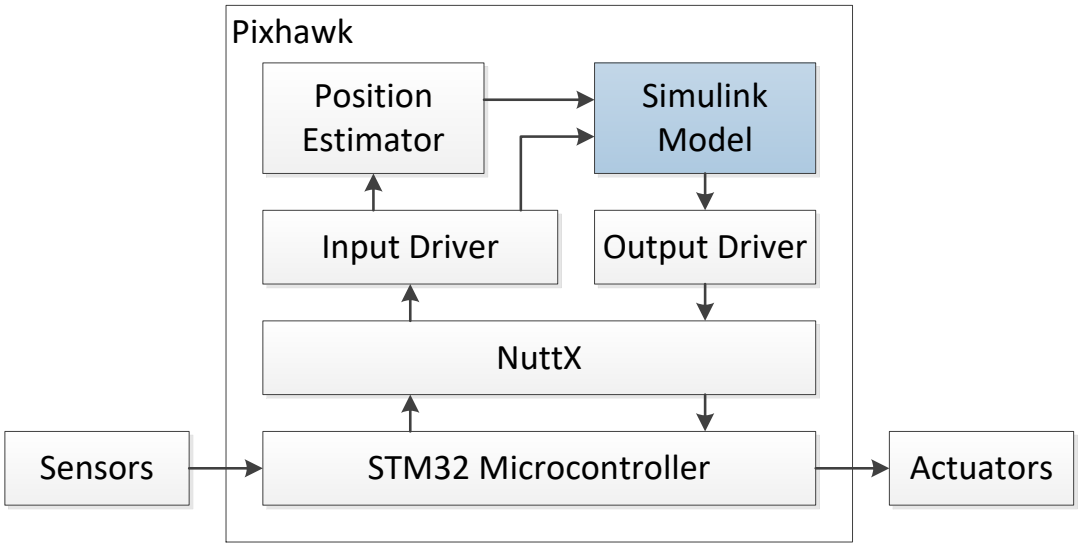


Figure 12: Pixhawk Architecture and PX4 Software.

3.5 Compilation and Software Upload

To implement new generated code as described in Sec. 4.3, go to the directory, where the toolchain was installed and follow these steps:

1. Run the Cygwin console (`run-console.bat`).
2. Change to Firmware directory by entering `cd Firmware` in the command line interface.
3. Connect the Pixhawk to the user's computer via USB cable.
4. Copy the generated code (`controller.h`, `controller.cpp`, `controller_data.cpp`, `controller_private.h`, `controller_types.h`, `rtwtypes.h`) from your code generation folder into the flight control module directory of the Pixhawk flight stack, where you installed the PX4 software (`/home/Firmware/src/modules/flight_control`).
5. Compile and upload the flight stack by entering `make px4_fmu-v5_fixedwing upload`. In case connection problems occur, replug the USB cable.

To upload new source code in the same session, repeat steps 4 and 5. Figure 13 shows the flow chart from MATLAB/Simulink model to executable software on the Pixhawk.

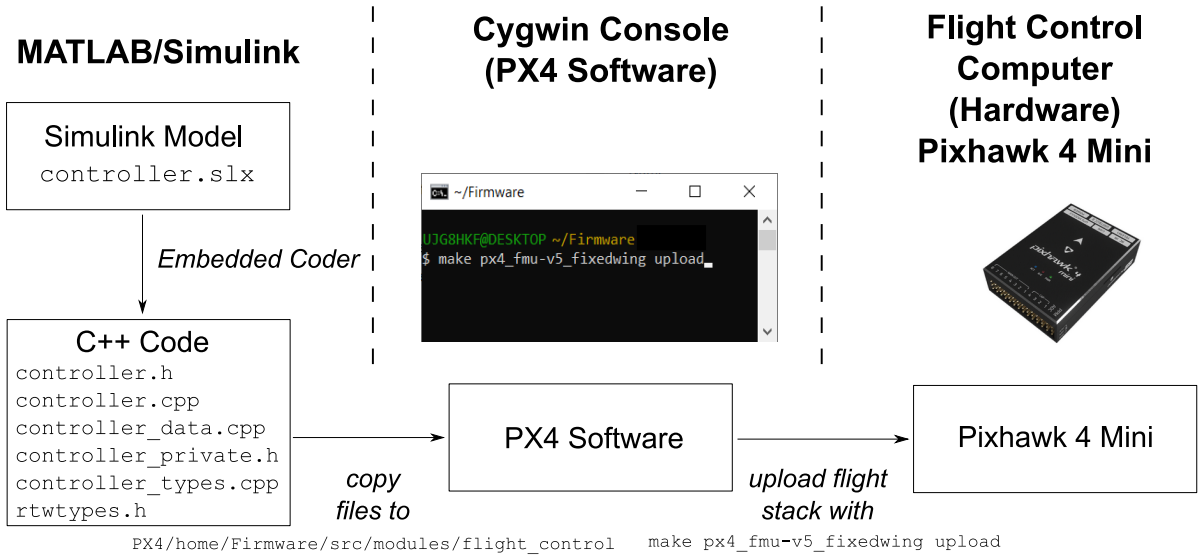


Figure 13: Flow Chart from MATLAB/Simulink Model to Executable Software on Pixhawk.

3.6 Access to Flight Test Data

The Pixhawk is capable of logging all data that are transferred between the different PX4 modules. These data transfers are organized in so-called *topics*. Logging starts with the boot sequence and ends with shutdown. It can be stopped and restarted using the NuttShell via QGC:

```
logger stop          // stop logging
logger start -f      // start logging
logger status        // get status of logger
```

Due to flight stack manipulations, it is not possible to use the `on` and `off` commands, described in the modules' documentation.
The Pixhawk Logger will write the log data into a file named `logs/<date>/<boot time>.ulg` on the SD Card. If the Pixhawk does not have information about the current time, the filename changes to `logs/sessXXX/log001.ulg`. The file format is `ulog`, which is a binary format. There are two ways to access the log data: i) by removing the SD card from the Pixhawk and plugging it into an SD Card reader or ii) via QGC. For using the second method, navigate through QGC (see Fig. 14):

1. Go to the `Analyze` view,
2. Select the `Log Download` option,
3. Refresh the list with the button on the right side, and
4. Download the relevant log files.

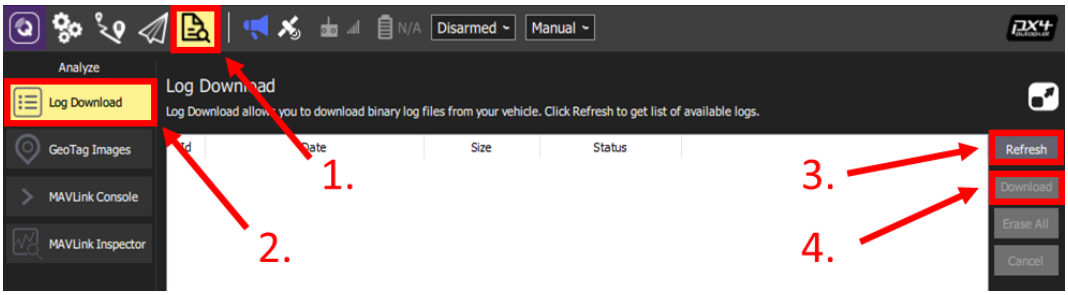


Figure 14: QGroundControl Access to Log Data (Order of Steps Highlighted).

The following two tools for evaluating and visualising the log data should be considered:

PX4 Flight Review:[8] The strength of this tool is its capability to visualise data. It is possible to get a full 3D playback of the mission. A disadvantage is that `.mat` or `.csv` files cannot be exported for further analysis. Also custom logs that are not part of the original PX4 software cannot be processed with PX4 Flight Review.
To upload a log file, click `Choose File...`, browse to the log file and then click `Upload`. Diagrams that visualise the flight log data will appear. To get the 3D flight playback, click `Open 3D View`.

Pyulog and Matulog in combination with MATLAB: Pyulog is a Python-based tool that, in combination with Matulog, converts `.ulg` files to `.mat` files. Python needs to be installed so that the Pyulog script is working properly. The software can be found on the provided USB stick. Afterwards, the Matulog script can be started in MATLAB to choose a file for file conversion. The Matulog MATLAB scripts can also be found on the provided USB stick. The toolchain is described in Fig.15.

[8] `https://logs.px4.io/`.

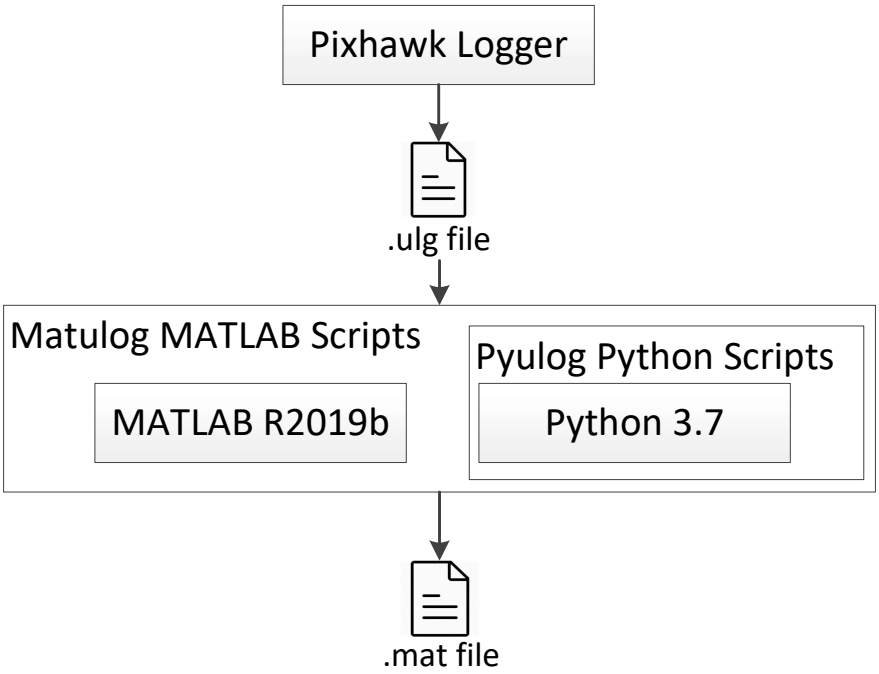


Figure 15: Toolchain for Log File Conversion.

A step-by-step installation guide for this tool in a MS Windows 10 environment follows:

1. Open `python-3.7.7-amd64.exe` (Python installation file) provided on the USB stick.
2. Check the `Add Python 3.7 to PATH` checkbox (see Fig.16).
3. Click `Install Now`. (If the `Customize installation` option is chosen, `pip` has to be checked for installation.)
4. Open Windows Powershell or Windows Command Prompt and switch to the USB drive, folder `/Pyulog/`.
5. Install Pyulog from the USB stick by entering the following command:

   ```
   python setup.py build install
   ```

   (If an error occurs, the python path was not added in the install menu, see Point 3. In this case, reinstall.)
6. Start MATLAB R2019b.
7. Open the Matulog script provided on the USB stick (folder: `/Matulog/`.
8. Click `Run` and choose the corresponding `.ulg` file from the Pixhawk SD card.

Figure 16: Python Installation Procedure.

The resulting `.mat` file contains a struct with 28 entries of logged raw data with different timestamps. An overview of the most important logged data is given in Tab. 3.

Table 3: Logged Data Overview.

Logged Data	Title of Category	Sample Rate
PWM signals for actuator control	`actuator_outputs_0`	$10\,\mathrm{s}^{-1}$
Simulink output signals for actuator control	`actuator_outputs_0_0`	$100\,\mathrm{s}^{-1}$
Airspeed sensor (indicated airspeed, true airspeed, air temperature)	`airspeed_0`	$100\,\mathrm{s}^{-1}$
Battery status (voltage, current, discharged capacity)	`battery_status_0`	$2\,\mathrm{s}^{-1}$
LiDAR distance	`distance_sensor_0`	$100\,\mathrm{s}^{-1}$
PWM signals of remote control and connection loss flag	`input_rc_0`	$100\,\mathrm{s}^{-1}$
Gyro and accelerometer sensor	`sensor_combined_0`	$100\,\mathrm{s}^{-1}$
Log data from Simulink model (to be defined manually)	`Talon_log`	$100\,\mathrm{s}^{-1}$
Barometer sensor (pressure, air temperature)	`vehicle_air_data`	$10\,\mathrm{s}^{-1}$
Rotation rates and quaternions (calculated by Extended Kalman Filter, EKF)	`vehicle_attitude_0`	$100\,\mathrm{s}^{-1}$
GPS position (measured data)	`vehicle_gps_position_0`	$10\,\mathrm{s}^{-1}$
Wind estimation (calculated by KF)	`wind_estimate_1`	$10\,\mathrm{s}^{-1}$

Most of the named signals are given or processed in the MATLAB/Simulink model and will be more discussed later in detail (see Tab. 4). It should further be mentioned that the created `.mat` file contains all raw data with the following limitations:

- Coordinate axes do not coincide for all data sets.
- Start time and end time of recordings vary with every data set.
- Not all data are stored in the correct unit.

To facilitate working with the log files, a MATLAB script `extract_logfile.m` is provided on the USB stick in the folder `/Matlab/LogData/ExtractData`. This script converts the raw log data to address the aforementioned limitations. Hence, it yields coinciding coordinate axes, provides coherent logging time spans and stores all data sets in the correct unit. The extraction function `extract_logfile` only needs the log file name as transfer parameter and returns a struct with easily accessible log data. Data sets of timestamps in seconds for each topic can be found in the converted struct. Furthermore, 20 log data can be individually defined in the MATLAB/Simulink model. Those can be found under the name `fc_log`.[9]

3.7 Reset

To set the flight control computer back to its default settings, follow these two steps:

1. **Resetting the Flight Stack:** Delete the folder `Firmware` in the `home` folder of the directory, where the toolchain was installed. Copy the original folder `Firmware` from the USB stick into that `home` folder (compare Step 3 in Sec. 3.1).
2. **Resetting the Parameter Configuration:** Connect the Pixhawk to the user's computer and open QGroundControl. Go to the `Vehicle Setup` view, select the `Parameters` option, and click the `Tools` button on the right side (see Fig. 17). Then choose "Load from file". Navigate to the USB Stick and choose the file `parameters_UAXS.params`. Finally, restart the Pixhawk by unplugging the power connection and reconnecting.

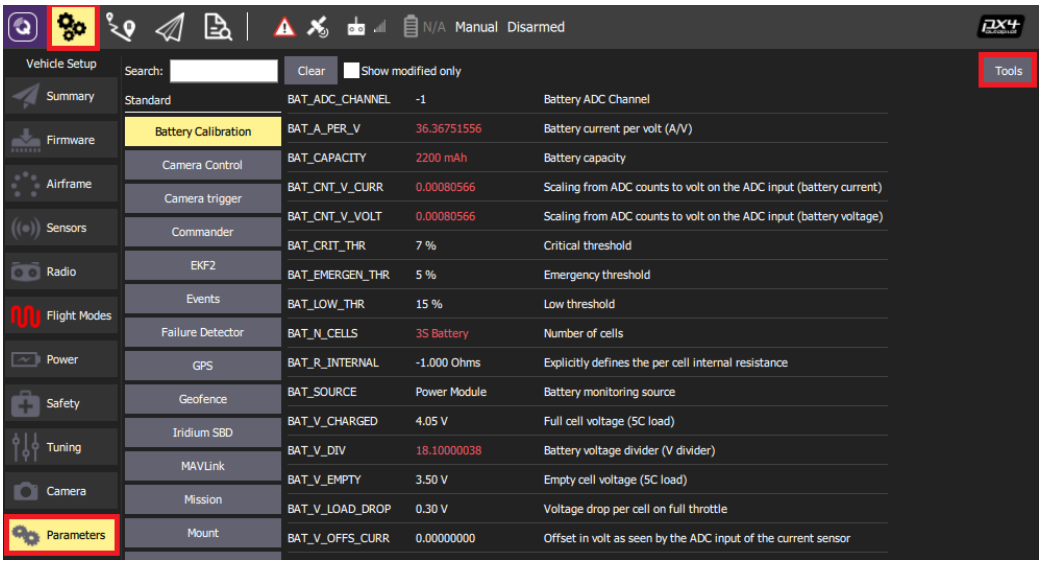


Figure 17: QGroundControl Interface to Load Parameters from File.

[9]The MATLAB script renames the original file from its initial name `Talon_log` for reasons of compatibility.

4 Simulink Development Model

The MATLAB/Simulink model provided on the USB stick is tested and designed for MATLAB R2019b. To edit this model, MATLAB together with the Simulink toolbox, the Embedded Coder and Simulink Coder are required.

4.1 Model Inputs and Outputs

The MATLAB/Simulink template `controller.slx` for the editable flight control logic is provided on the USB stick in the folder `/Simulink/Template`. It is illustrated in Fig. 18. The inports and outports define the interface between the flight controls and the software of the set. Hence, inputs and outputs shall not be changed when modelling the flight control logic.

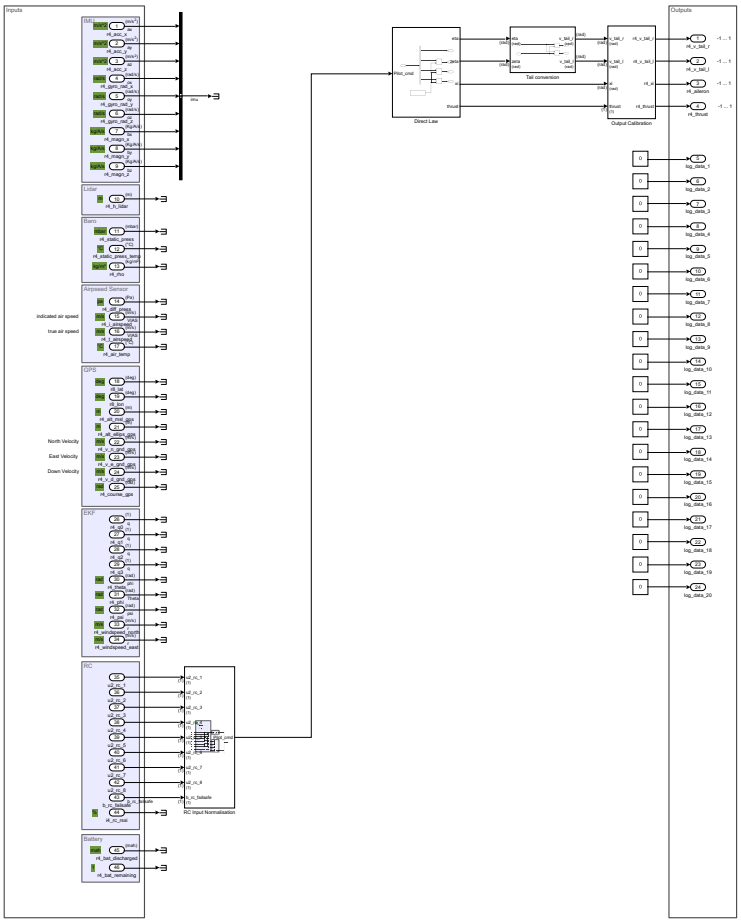


Figure 18: MATLAB/Simulink Template for the AlphaLink UAXS.

The left side shows the inports of the flight controller. Those include:

- Accelerometer, gyro and magnetic field data from internal Pixhawk inertial measurement unit (IMU),
- LiDAR distance from external LiDAR measurement equipment,
- Barometric measurements by internal Pixhawk sensor,
- Differential pressure data from external Pitot sensor,
- GPS data provided through PX4 Software from external GPS antenna,
- Estimated attitude (quaternions and Euler angles) and estimated wind speed computed by an extended Kalman filter (EKF) from PX4 software; EKF estimates are based on the internal Pixhawk IMU and GPS data,
- Inputs from the remote control, and
- Battery data.

The right side shows the outports of the flight controller. Those include:

- Commands for the 4 installed actuators and
- Configurable log data.

The MATLAB/Simulink inputs are pre-configured and shall be used as described in the Interface Control Table illustrated in Tab. 4. Contrary to the raw log file data, the inports of the MATLAB/Simulink model are already supplied with data in the correct coordinate axes and units. Changing the inputs or outputs may lead to source files that cannot be compiled and must therefore be strictly avoided!

Table 4: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
r4_acc_x	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-2}}$	10 ms	Acceleration measurement in x-axis direction
r4_acc_y	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-2}}$	10 ms	Acceleration measurement in y-axis direction
r4_acc_z	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-2}}$	10 ms	Acceleration measurement in z-axis direction
r4_gyro_rad_x	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{rad\,s^{-1}}$	10 ms	Gyro measurement about x-axis
r4_gyro_rad_y	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{rad\,s^{-1}}$	10 ms	Gyro measurement about y-axis
r4_gyro_rad_z	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{rad\,s^{-1}}$	10 ms	Gyro measurement about z-axis

Table 4: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
r4_magn_x	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{kg\,A^{-1}\,s^{-2}}$	10 ms	Magnetometer measurement of magnetic induction in x-axis direction
r4_magn_y	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{kg\,A^{-1}\,s^{-2}}$	10 ms	Magnetometer measurement of magnetic induction in y-axis direction
r4_magn_z	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{kg\,A^{-1}\,s^{-2}}$	10 ms	Magnetometer measurement of magnetic induction in z-axis direction
r4_h_lidar	single	0.3	3.4028 *10^38	m	10 ms	Distance measured by LiDAR sensor (range only measurable from 0.3-12 m)
r4_static_press	single	0	3.4028 *10^38	mbar	10 ms	Calibrated static pressure measured by Pixhawk barometer sensor
r4_static_press _temp	single	0	3.4028 *10^38	°C	10 ms	Temperature measured by Pixhawk barometer sensor
r4_rho	single	0	3.4028 *10^38	$\mathrm{kg\,m^{-3}}$	10 ms	Air density calculated by Pixhawk barometer sensor
r4_diff_press	single	-3.4028 *10^38	3.4028 *10^38	Pa	10 ms	Calibrated differential pressure
r4_i_airspeed	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	10 ms	Indicated airspeed calculated by airspeed sensor
r4_t_airspeed	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	10 ms	True airspeed calculated by airspeed sensor
r4_air_temp	single	-50	200	°C	10 ms	Air temperature measured by airspeed sensor
r8_lat	double	-90	90	deg	500 ms	Latitude measured by GPS sensor
r8_lon	double	-180	180	deg	500 ms	Longitude measured by GPS sensor

Table 4: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
r4_alt_msl_gps	single	-3.4028 *10^38	3.4028 *10^38	m	500 ms	Altitude above mean sea level measured by GPS sensor
r4_alt_ellips_gps	single	-3.4028 *10^38	3.4028 *10^38	m	500 ms	Altitude above ellipsoid measured by GPS sensor
r4_v_n_gnd_gps	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	500 ms	North velocity measured by GPS sensor
r4_v_e_gnd_gps	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	500 ms	East velocity measured by GPS sensor
r4_v_d_gnd_gps	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	500 ms	Down velocity measured by GPS sensor
r4_course_gps	single	0	2π	rad	500 ms	Course measured by GPS
r4_q0	single	-1	1	1	10 ms	Quaternion element 0 calculated by EKF
r4_q1	single	-1	1	1	10 ms	Quaternion element 1 calculated by EKF
r4_q2	single	-1	1	1	10 ms	Quaternion element 2 calculated by EKF
r4_q3	single	-1	1	1	10 ms	Quaternion element 3 calculated by EKF
r4_theta	single	$-\pi$	π	rad	10 ms	Pitch angle calculated by EKF
r4_phi	single	$-\pi$	π	rad	10 ms	Roll angle calculated by EKF
r4_psi	single	$-\pi$	π	rad	10 ms	Yaw angle calculated by EKF
r4_windspeed _north	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	10 ms	North wind speed calculated by EKF
r4_windspeed _east	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	10 ms	East wind speed calculated by EKF
u2_rc_1	uint16	982	2006	1	10 ms	PWM input of 1st RC channel
u2_rc_2	uint16	982	2006	1	10 ms	PWM input of 2nd RC channel
u2_rc_3	uint16	982	2006	1	10 ms	PWM input of 3rd RC channel
u2_rc_4	uint16	982	2006	1	10 ms	PWM input of 4th RC channel

Table 4: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
u2_rc_5	uint16	982	2006	1	10 ms	PWM Input of 5th RC channel
u2_rc_6	uint16	982	2006	1	10 ms	PWM Input of 6th RC channel
u2_rc_7	uint16	982	2006	1	10 ms	PWM Input of 7th RC channel
u2_rc_8	uint16	982	2006	1	10 ms	PWM Input of 8th RC channel
b_rc_failsafe	boolean	0	1	1	10 ms	RC connection loss flag
i4_rc_rssi	int32	0	100	%	10 ms	Receiver RC signal strength indicator
r4_bat_discharged	single	0	3.4028 *10^38	mAh	10 ms	Discharged capacity
r4_bat_remaining	single	0	1	%	10 ms	Remaining battery time
r4_v_tail_r	single	-1	1	1	10 ms	Actuator: right fin of V-tail
r4_v_tail_l	single	-1	1	1	10 ms	Actuator: left fin of V-tail
r4_aileron	single	-1	1	1	10 ms	Actuator: aileron
r4_thrust	single	0	1	1	10 ms	Actuator: thrust
log_data_n	single	-3.4028 *10^38	3.4028 *10^38	1	10 ms	Optionally loggable data

When reviewing Tab. 4, special attention must be paid to the update rate of the GPS data and the range of the LiDAR sensor. This range is between 0.3 m and 12 m, but the sensor will send random values above 12 m, when it exceeds its upper range limit.

4.2 Linking of Inputs and Outputs

As an example, the Direct Law is already implemented in the MATLAB/Simulink template (see Fig. 18). The Direct Law sets the actuators proportionally to the inputs, which are controlled by the remote control.
To discuss the design of the template, it will be shown how the inputs and outputs can be linked. Table 5 provides the name convention for setting the name prefix of the used inports and outports.

Table 5: Inport and Outport Name Convention

Data Type	Float64	Float32	Int32	Int16	Uint16	Boolean
Name Prefix	r8	r4	i4	i2	u2	b

RC_Input_Normalisation Subsystem: The first subsystem (see Fig. 19) normalises the PWM signal inputs from the remote control. The remote control channels must be allocated manually in the remote control menu (see Sec. 1.2). Here, the first 4 channels of the remote control are configured to control the actuators in the following order:

1. Thrust
2. Elevator
3. Aileron
4. Rudder

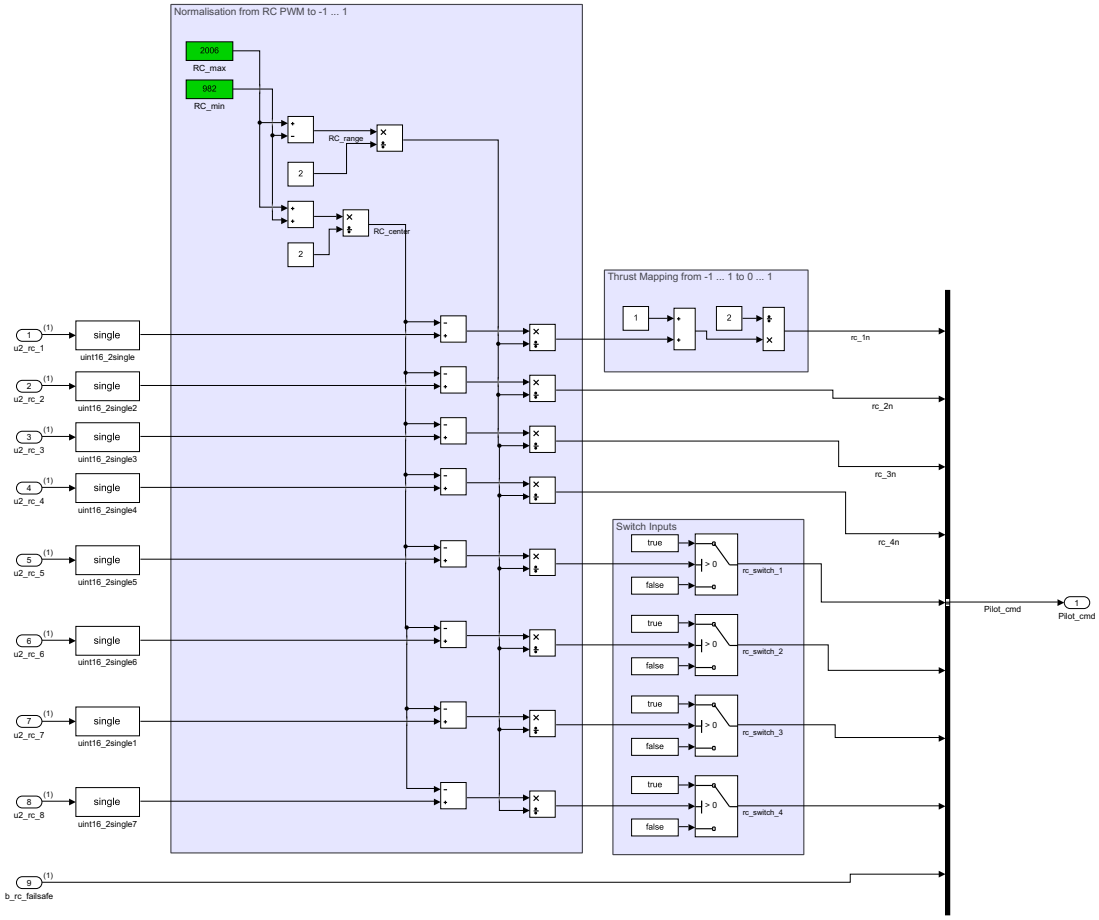


Figure 19: Subsystem RC_Input_Normalisation.

As shown in Ta. 4, the first 8 RC inputs are defined as `uint16` data type with a range between 982 and 2006. Therefore, the PWM signals need to be converted into a floating point format. This is done by data type conversion blocks, which are placed on the left side.
The blocks inside the first purple frame (placed on the left side) and the second purple frame (placed at upper right side) convert the outputs to ranges between 0 and 1 for the thrust signal and to values between -1 and 1 for elevator, aileron and rudder signals. The blocks inside the third purple frame (placed at the right bottom side) are implemented here to show how a switch for turning signals on and off could look like.

Direct_Law Subsystem: This subsystem sets the signals of elevator, aileron and rudder to the actual deflection in rad.

Tail_Conversion Subsystem: Here, the signal Eta (η) sets the V-tail flight control surfaces to a pitching moment, while the signal zeta (ζ) sets the V-tail flight control surfaces to a yawing moment. When both control surfaces are used, deflection angles are set by superposition. The coordinate system is right-handed with the z-axis pointing downwards.

Output_Calibration Subsystem: This subsystem (see Fig. 20) shall stay the last subsystem for calibrating the outputs. A short description of how to calibrate outputs can be found inside the subsystem. The calibration is needed to set the null position in line with the airfoil camber line. At least one further position needs to be measured to calibrate the deflection of flight control surfaces. In the provided set, the outputs are already pre-calibrated.

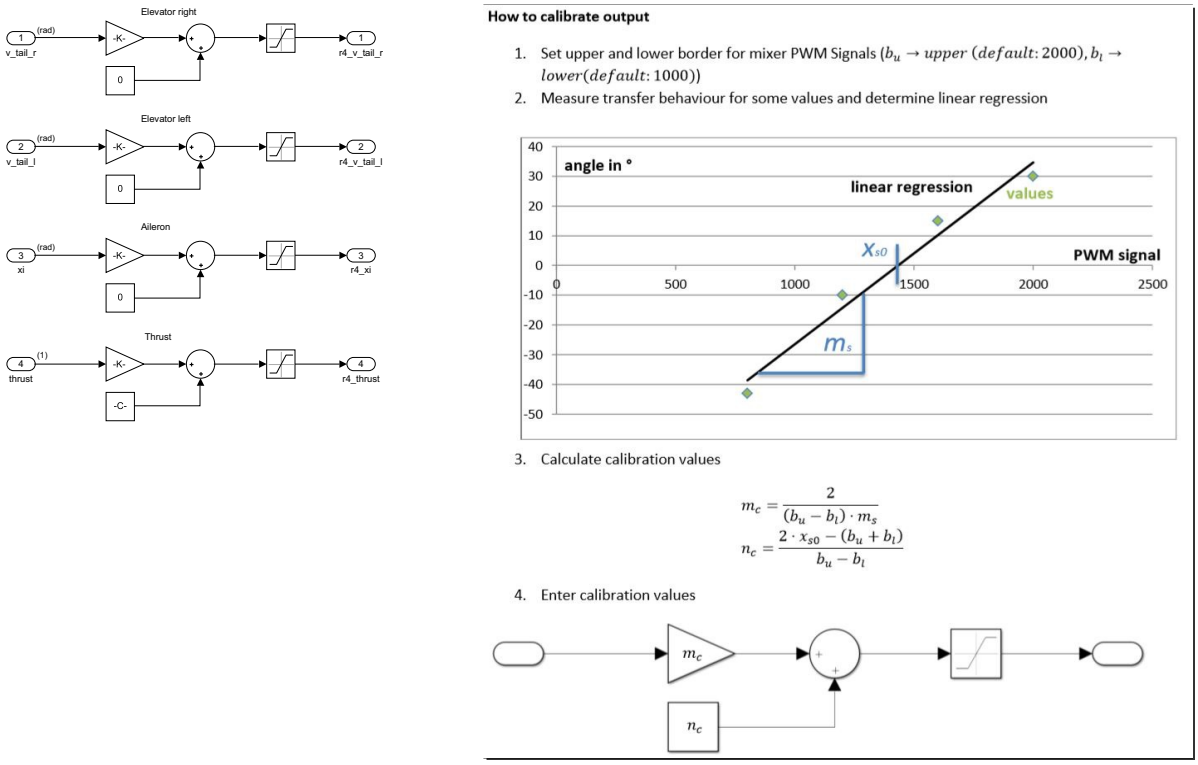


Figure 20: Subsystem Output_Calibration.

4.3 C++ Code Generation of the Simulink Model

MATLAB/Simulink is able to generate C++ code from created Simulink models if the required software packages are installed. This requires four steps:

1. **Installing required Software Add-Ons:** If the software add-ons are not already pre-installed, the software packages `MATLAB Coder` and `Simulink Coder` need to be in-

stalled. Inside the Simulink integrated development environment (IDE), the installation menu can be found by clicking `Get Add-Ons` as shown in Fig. 21.

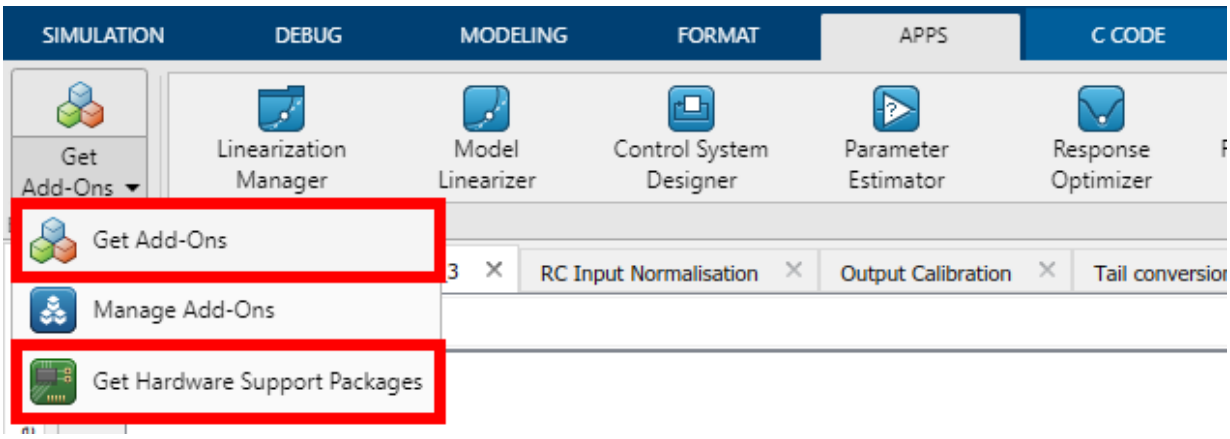


Figure 21: Simulink Add-Ons and Hardware Support Packages.

2. **Installing required Hardware Support Packages:** As already described in Fig. 12, the used microcontroller is an STM32. Mathworks provides hardware packages that support C++ code generation for STM32 microcontrollers. By clicking `Get Hardware Support Packages` as shown in Fig. 21, the Simulink hardware support menu will be opened. There, the support package `Embedded Coder Support Package for ARM Cortex-M Processors` can be found and needs to be installed (see Fig. 22).

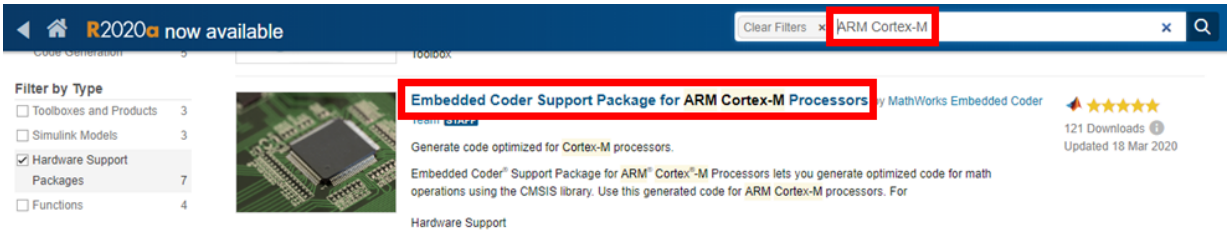


Figure 22: ARM Cortex-M Add-On.

3. **Restart MATLAB:** MATLAB has to be restarted. After the restart, the installed support package will automatically set MATLAB/Simulink preferences to the configuration that can be seen in Fig. 23.

4. **Generating C++ Code:** For the last step, the Simulink model has to be saved as `controller.slx`.

   Note: It is necessary to exactly use this filename (`controller.slx`) *when generating code so that all function calls are working properly!*

   The code can then be generated by clicking `Generate Code` (see Fig. 23). When the code generation is successful, MATLAB will show the Code Generation Report window with details of the generated code (e.g. the path of the selected code generation folder); otherwise the Diagnostic Viewer will be shown for debugging. To upload the generated flight controller code to the Pixhawk, review and follow the steps described in Sec. 3.5.

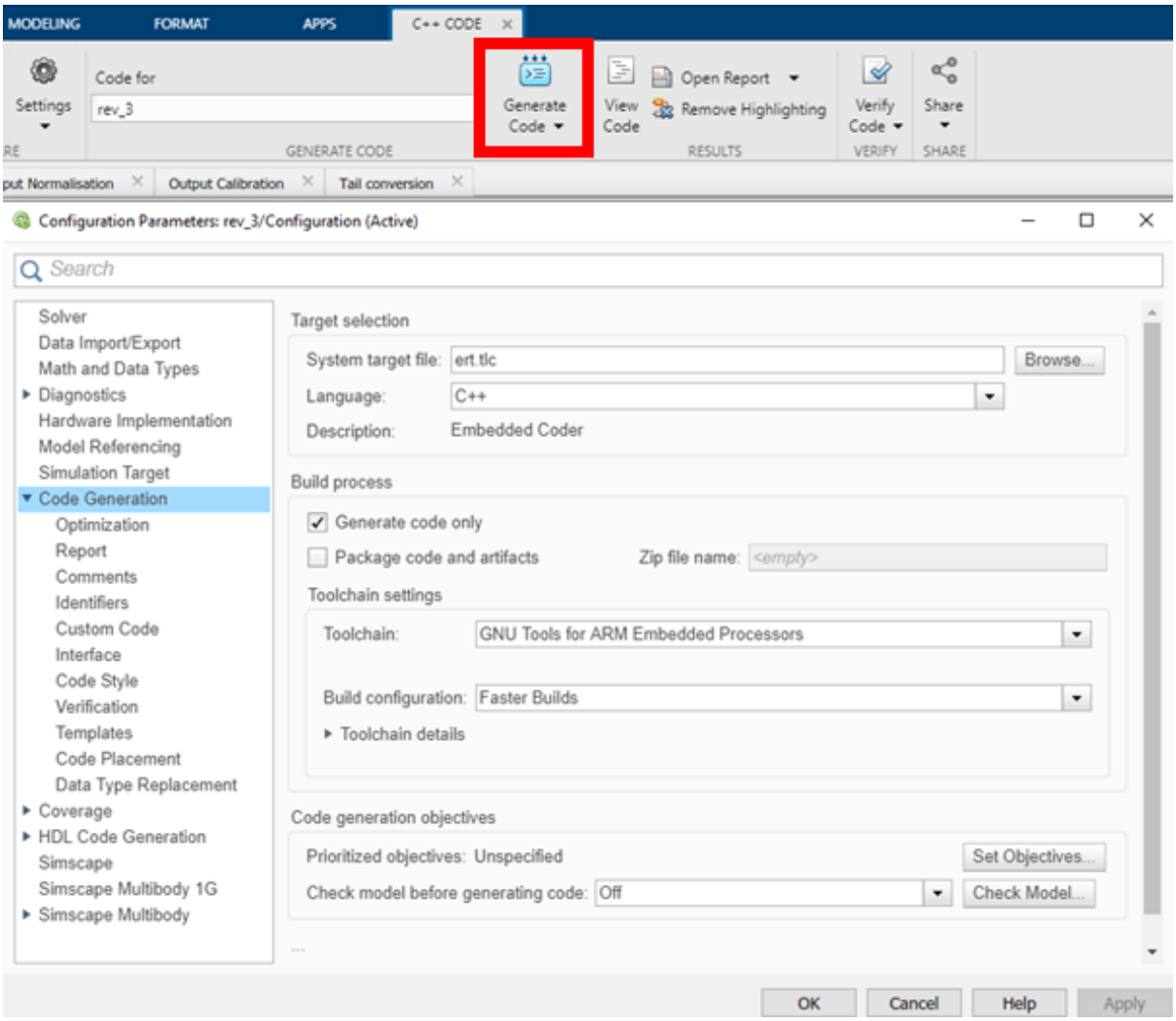


Figure 23: Settings for Code Generation.

4.4 Modelling Guidelines and Development Recommendations

Model driven development (MDD) with MATLAB/Simulink is characterised by finding a solution to meet functionality, optimization and model-design requirements. General guidelines for optimization and best practices of model design can be found in the MathWorks Advisory Board (MAB), among others.[10] Guidelines to meet functionality requirements and to ensure conformance of software standards can be found by using the MATLAB Model Advisor. The Model Advisor provides, among others, information about supported blocks for code generation and points out bad practices in the created Simulink Model.
With previous setups of the AlphaLink UAXS, the MATLAB Model Advisor i) did not always cover all bad practices that led to malfunctioning code generation and, to some extent, ii) labeled practices as wrong that were functioning just fine. Hence, the checks of the MATLAB Model Advisor should be treated at individual review. For instance, in the Model Advisor checks for `Embedded Coder` it is recommended to set the setting of `Hardware Board` to a specific model (default: `none`). When setting it to `ARM-Cortex`, code generation for C++ stops

[10] `https://de.mathworks.com/solutions/mab-guidelines.html`.

working while it works just fine on the Pixhawk if the settings are set to `none` there. To be clear: the hardware package for ARM-Cortex-M Boards itself must be installed to generate code, but it is just working for C++ code generation when it is set to `none`. Another example is that the Model Advisor does not point out that the `Pow` block leads to malfunctioning code. After generating code, the pow function is inaccurately casted to type `double`, when `single` is selected. The typecast needs to be changed manually in the generated code in this case.

Some helpful basic guidelines for code generation that take into account most common software standards - by no means exhaustive - are listed below.

- **Using consistent software environment:** It is recommended to use consistent software releases for MATLAB, Simulink and C++ compilers.
- **Avoiding usage of `MATLAB Function` blocks:** Custom created MATLAB blocks are supported for code generation. Nevertheless, MATLAB functions may be hard to debug because Simulink will only point out blocks that cause errors. If MATLAB functions are used, a maximum number of 60 lines shall not be exceeded.
- **Avoiding usage of time-continuous blocks:** Real-time operating systems (RTOS) such as `NuttX` can only process time discrete signals. Hence, the usage of continuous time blocks must be avoided.
- **Avoiding usage of `Mux` blocks for bus signals:** Avoid using Mux blocks to create bus signals. Always use `Bus Creator` blocks to allocate signals by name.
- **Avoiding floating point comparison for equality/inequality:** Due to rounding errors, most floating-point numbers end up being slightly imprecise. This leads to equality test failing and therefore must be avoided.
- **Avoiding usage of memory intensive blocks:** Memory intensive blocks such as `Fuzzy Logic controllers` shall not be used, when generating code for embedded platforms such as microcontrollers. The reason is that the generated code will not fit in the stack due to the high amount of floating-point variables required.
- **Set default data type:** MATLAB/Simulink sets default data type to `double`. This is sometimes a problem for code generation as not all targets can support these types of variables and it can be hard to change these once a model is created. Having this in mind will save a lot of time trying to find problematic typecasts. The provided template has already set `single` as default data type.
- **Using C/C++ code compatible names:** Names that use single-byte alphanumeric characters (a-z, A-Z, 0-9) and single-byte underscore (_), are compatible with C++ code generation. A maximum length for names of 32 characters should not be exceeded. Deviating from this naming practice may lead to errors.
- **Avoiding division by 0:** Division blocks must always be protected from division by 0.
- **Avoiding algebraic loops:** They are problematic for code generation as stated by Mathworks. If the algebraic loop cannot be avoided, use a `Delay` block to break up the loop.
- **Avoiding usage of `If` blocks without `else` condition:** An alternative (`else`) path must always be provided for Simulink.