AlphaLink Engineering GmbH
Bismarckstraße 10-12
10625 Berlin

Unmanned Aircraft Experimental System

The Flying Lab for Applied Flight Control and Flight Mechanics

Manual

Version: 2.1
Date: April 30, 2021

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 is designed for computers with MS Windows as operational system and is tested with MATLAB R2019b. Upward and downward compatibility is not guaranteed.

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 V3, EU), and
6. USB Stick with all relevant software and this manual (see the directory in Appendix A).

The following components are 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, incl. battery),
2. LiDAR Sensor (Benewake TFmini Plus or S Micro LiDAR),
3. Hardware-in-the-Loop Simulator (AlphaLink software, Vector Informatik box VN1630A and CANoe software), and
4. Access to Virtual Flight Test Environment (AlphaLink software).

Contents

1 Setup and Commissioning of the Airborne System

The PX4 software is 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).

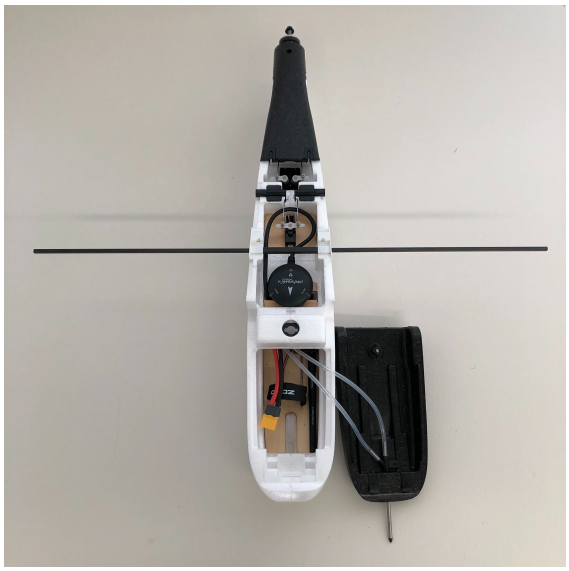


Figure 1: Installed Wing Spar.

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).

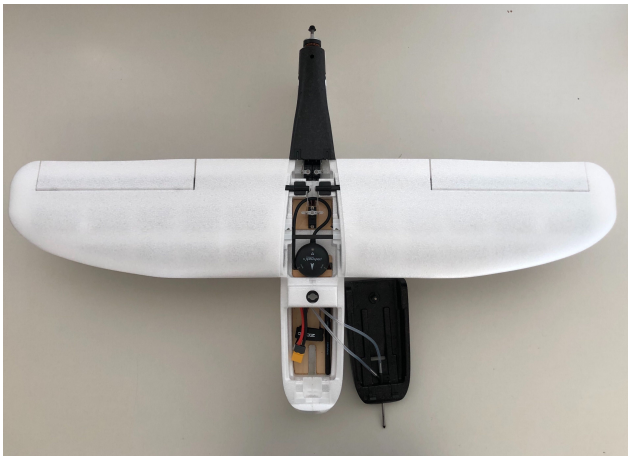


Figure 2: Installed Wing.

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

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).

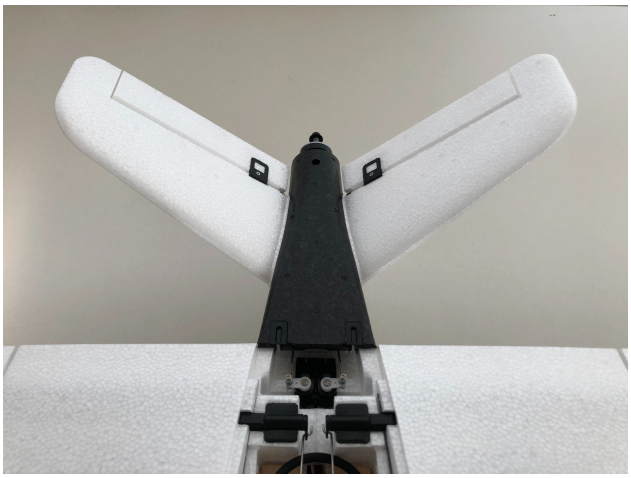


Figure 3: Installed V-Tail.

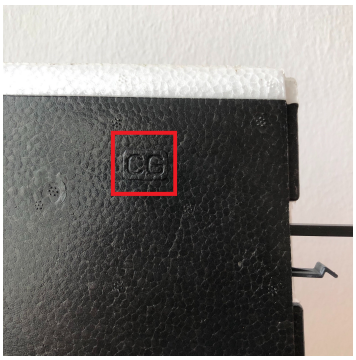
4. Stick the propeller onto the motor shaft with the markings facing forward. Then, tighten the nut with an $8\,\mathrm{mm}$ wrench (Fig. 4).

Figure 4: Installed Propeller.

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 (Fig. 5a). Underneath the wing, there are two center of gravity (CG) marks (Fig. 5b). Place a finger on each mark and balance the plane (Fig. 5c). 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).

(a) Battery.



(b) CG Mark.

(c) CG Balancing.

Figure 5: Center of Gravity Adjustment.

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 as shown in Fig. 6. Please note that Fig. 10 at the end of Sec. 2 shows the hardware dependencies in detail. 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.

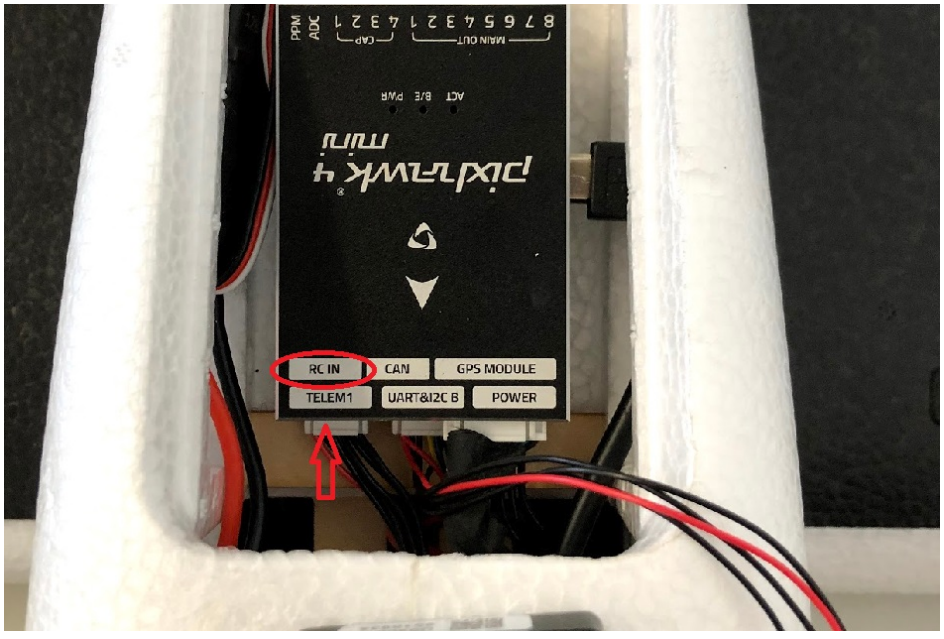


Figure 6: Connection of the RC Receiver to Pixhawk 4 Mini.

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

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	Channel 6: Switch 2
Channel 7: Switch 3	Channel 8: Switch 4	Channel 9: Switch 5
Channel 10: Switch 6	Channel 11: Switch 7	Channel 12: Switch 8

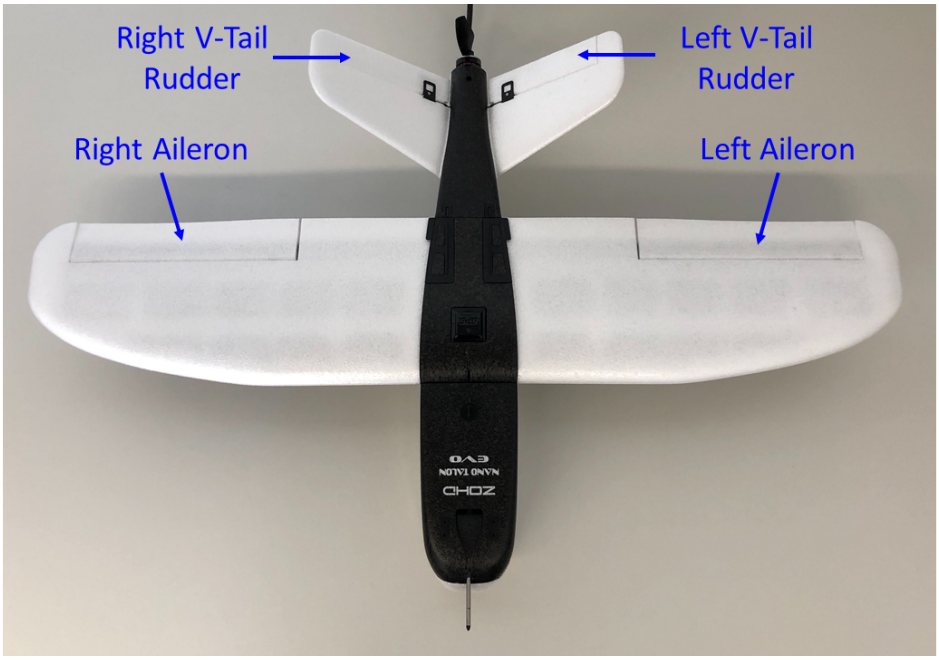


Figure 7: Assignment of the Rudders.

Be aware that the controls must be correct according to the flight mechanic algebraic sign. Taking Fig. 7 into account, this means: An input in the elevator channel with stick forward (command nose down) must deflect both flaps at the V-tail downwards (flight mechanic positive sign). An input in the elevator channel with stick to the rear (command nose up) must move both flaps at the V-tail upwards (flight mechanic negative sign). An input in the aileron channel with stick to the right (command roll right) must move the right aileron up (negative sign) and the left aileron down. An input in the aileron channel with stick to the left (command roll left) must move the right aileron down (positive sign) and the left aileron up. An input in the rudder channel with stick to the right (command yaw to the right) must deflect the left flap at the V-tail up and the right flap at the V-tail down. An input in the rudder channel with stick to the left (command yaw to the left) must move the left flap at the V-tail down and the right flap at the V-tail up. 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 8 shows the configuration of the FrSky Taranis Q X7 remote control (available as part of the pre-configured RC add-on) as an example.

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

Figure 8: Channel Layout of Exemplary Remote Control (Model: Taranis FrSky Q X7).

Be careful with the thrust! If you check the settings at thrust, make sure that the propeller is disassembled. If you make an input at the thrust channel to the rear, the motor must be at a standstill. If the input at the thrust channel is to the front, the motor must give its full power.

You may inadvertently or intentionally switch to the programming mode of the motor controller (ESC). If this is the case, please note the following:

1. Disconnect the battery from the Pixhawk. Disassemble the propeller to avoid injury or damage.
2. Turn on the remote control.
3. Move the stick for the thrust channel to the most forward position (full thrust).
4. Connect the Pixhawk to the computer, using a USB cable.
5. Connect the battery to the Pixhawk, wait about two seconds until you hear a beep-beep tone confirming that the thrust channel is commanding the highest thrust value.
6. Move the stick of the thrust channel to the rear. You will hear a beep-beep sound again, confirming that you are now commanding the motor to stop.

7. ESC will self detect, after the sound stops, detection is finished.

8. The motor controller is ready to go.

With a connected battery, the ESC always switches to programming mode when the Pixhawk commands full thrust to the ESC. To avoid that, make sure that the stick of the thrust channel commands the motor to stop, if the battery is connected to the system.

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. 12) and enter `reboot`.
- ☐ Check that GPS signal is valid. This is done by entering 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 motor, 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 ($11.1\,\mathrm{V}$).
- Generally use a charging current of $1.5\,\mathrm{A}$. If the battery needs to be charged faster, you can set the charger to $3.1\,\mathrm{A}$, 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
- If you enter the command `commander start` in the NuttShell of QGroundControl, the motor may start up. Therefore, always remove the propeller when you are not flying with the system!

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. This chapter describes the aircraft and its flight dynamic modeling and flight control system.

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. 9). 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 and is 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	3×9 g metal gear	Motor	SunnySky 2204-1870KV
Propeller	$6{\times}3$ (inch)	ESC	30 A, 5 V 2 A BEC

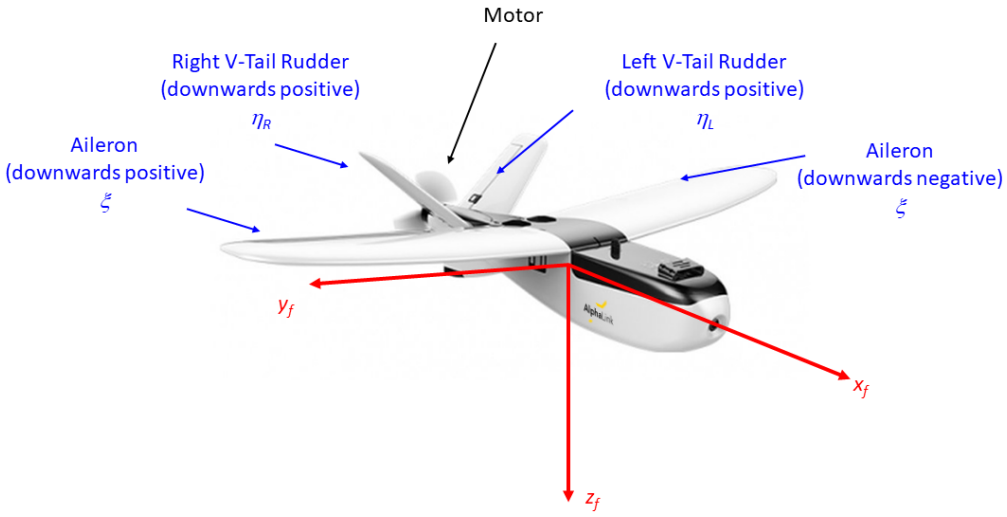


Figure 9: Illustration of Talon Nano Evo with Rudders and Motor in a Coordinate System.

The aerodynamic rudders are limited within $\pm 25°$. The rotational speed of the motor 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 motor.

2.2 Linearized Flight Dynamics Models

A flight mechanical model for the aircraft was linearized 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 linearized 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`).

All state space models have the form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\,\mathbf{x} + \mathbf{D}\,\mathbf{u} \end{aligned}$$

with $\mathbf{x}$ as state vector, $\mathbf{u}$ as input vector, $\mathbf{y}$ as output vector, $\mathbf{A}$ as dynamic matrix, $\mathbf{B}$ as input matrix, $\mathbf{C}$ as output matrix and $\mathbf{D}$ as feedforward matrix. The following state $\mathbf{x}_{\text{long}}$, output $\mathbf{y}_{\text{long}}$ and input variables $\mathbf{u}_{\text{long}}$ are used for the model of longitudinal motion

$$\mathbf{x}_{\text{long}} = \begin{bmatrix} \delta q \\ \delta \alpha \\ \delta V_{\text{IAS}} \\ \delta \Theta \end{bmatrix}, \quad \mathbf{y}_{\text{long}} = \begin{bmatrix} \delta q \\ \delta \alpha \\ \delta V_{\text{IAS}} \\ \delta \Theta \\ \delta \gamma \\ \delta V_{\text{K}} \end{bmatrix} \text{ and } \quad \mathbf{u}_{\text{long}} = \begin{bmatrix} \delta \eta \\ \delta \eta_{\text{TL}} \\ \delta w_{\text{wg}} \\ \delta u_{\text{wg}} \\ \delta q_{\text{wf}} \end{bmatrix}.$$

The meaning of the variables above and their units are listed in Tab. 3.

Table 3: Variables of Longitudinal Motion.

Variable	Meaning	Type	Unit
δq	Pitch rate	State / output	$1\,\mathrm{rad\,s^{-1}}$
$\delta\alpha$	Angle of attack	State / output	$1\,\mathrm{rad}$
δV_{IAS}	Indicated airspeed	State / output	$1\,\mathrm{m\,s^{-1}}$
$\delta\Theta$	Pitch angle	State / output	$1\,\mathrm{rad}$
$\delta\gamma$	Flight path angle	Output	$1\,\mathrm{rad}$
δV_{K}	Ground speed	Output	$1\,\mathrm{m\,s^{-1}}$
$\delta\eta$	Elevator deflection	Input (control)	$1\,\mathrm{rad}$
$\delta\eta_{\mathrm{TL}}$	Thrust lever	Input (control)	1
δw_{wg}	Vertical wind speed	Input (disturbance)	$1\,\mathrm{m\,s^{-1}}$
δu_{wg}	Horizontal wind speed	Input (disturbance)	$1\,\mathrm{m\,s^{-1}}$
δq_{wf}	Wind-induced pitch rate	Input (disturbance)	$1\,\mathrm{rad\,s^{-1}}$

The following state $\mathbf{x}_{\mathrm{lat}}$, output $\mathbf{y}_{\mathrm{lat}}$ and input variables $\mathbf{u}_{\mathrm{lat}}$ are used for the model of lateral motion

$$\mathbf{x}_{\mathrm{lat}} = \begin{bmatrix} \delta r \\ \delta\beta \\ \delta p \\ \delta\Phi \end{bmatrix}, \quad \mathbf{y}_{\mathrm{lat}} = \begin{bmatrix} \delta r \\ \delta\beta \\ \delta p \\ \delta\Phi \\ \delta a_y \end{bmatrix} \text{ and } \quad \mathbf{u}_{\mathrm{lat}} = \begin{bmatrix} \delta\xi \\ \delta\zeta \\ \delta v_{\mathrm{wg}} \\ \delta p_{\mathrm{wf}} \\ \delta r_{\mathrm{wf}} \end{bmatrix}.$$

The meaning of the variables above and their units are listed in Tab. 4.

Table 4: Variables of Lateral Motion.

Variable	Meaning	Type	Unit
δr	Yaw rate	State / output	$1\,\mathrm{rad\,s^{-1}}$
$\delta\beta$	Sideslip angle	State / output	$1\,\mathrm{rad}$
δp	Roll rate	State / output	$1\,\mathrm{rad\,s^{-1}}$
$\delta\Phi$	Bank angle	State / output	$1\,\mathrm{rad}$
δa_{y}	Vertical acceleration	Output	$1\,\mathrm{m\,s^{-2}}$
$\delta\xi$	Aileron deflection	Input (control)	$1\,\mathrm{rad}$
$\delta\zeta$	Rudder deflection	Input (control)	$1\,\mathrm{rad}$
δv_{wg}	East / west wind speed	Input (disturbance)	$1\,\mathrm{m\,s^{-1}}$
δp_{wf}	Wind-induced roll rate	Input (disturbance)	$1\,\mathrm{rad\,s^{-1}}$
δr_{wf}	Wind-induced yaw rate	Input (disturbance)	$1\,\mathrm{rad\,s^{-1}}$

For the coupled model of longitudinal and lateral motion the state $\mathbf{x}_{\text{full}}$, output $\mathbf{y}_{\text{full}}$ and input variables $\mathbf{u}_{\text{full}}$ are composed of

$$\mathbf{x}_{\text{full}} = \begin{bmatrix} \mathbf{x}_{\text{lon}} \\ \mathbf{x}_{\text{lat}} \end{bmatrix}, \quad \mathbf{y}_{\text{lat}} = \begin{bmatrix} \mathbf{y}_{\text{lon}} \\ \mathbf{y}_{\text{lat}} \end{bmatrix} \text{ and } \quad \mathbf{u}_{\text{lat}} = \begin{bmatrix} \mathbf{u}_{\text{lon}} \\ \mathbf{u}_{\text{lat}} \end{bmatrix}.$$

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 configured and installed in the aircraft. The Pixhawk is 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 open lightweight protocol 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 connected to this interface. Figure 10 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 11 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 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 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 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 built in the airframe and connected to the sensor. The sensor is connected to the Pixhawk via I2C. The maximum measuring range is $6894.76\,\text{Pa}$. The resolution is $0.74\,\text{Pa}$. The accuracy is in the range of $\pm\ 17.23\,\text{Pa}$ (at $25\,^{\circ}\text{C}$). The dynamic pressure sensor is pre-calibrated by the original manufacturer. It is recommended to perform a new calibration. The

[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.

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]

A telemetry modem is used for communication with the ground station (QGroundControl software). It is 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 $433\,\mathrm{MHz}$ and the maximum transmission power is $100\,\mathrm{mW}$. According to the original manufacturer, distances of up to $300\,\mathrm{m}$ 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).

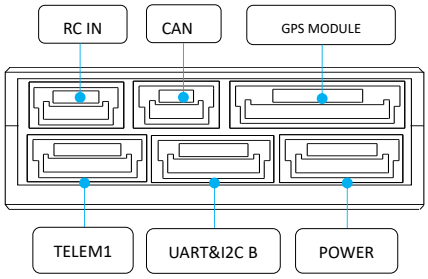
For sets with the remote control add-on, a FrSky Q X7 RC transmitter 2.4 GHz with 16 channel and a FrSky R-XSR EU LBT receiver are included in the set. The remote control is configured. The receiver is connected to the Pixhawk via the S.BUS interface.

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 $12\,\mathrm{m}$, the lowest measurable height is $30\,\mathrm{cm}$. It is connected to the flight control computer via the UART interface. The instrument is 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.

[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.

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



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

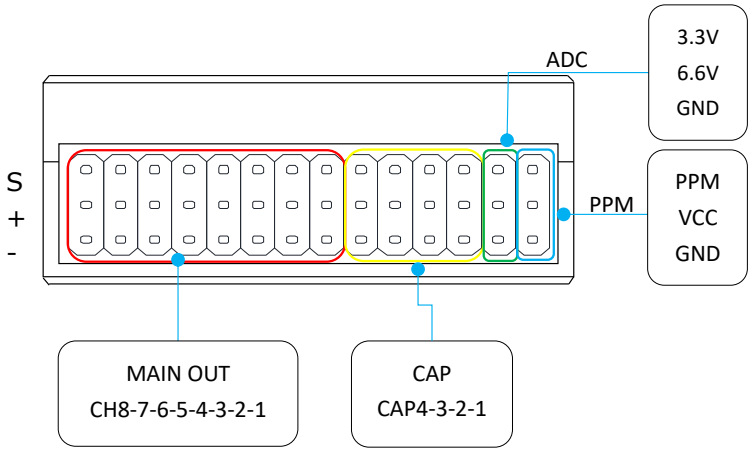
GPS MODULE ports

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 10: 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 11: Pin Layout of Pixhawk 4 Mini for Main (PWM) Out, PPM, ADC, and CAP. *Source*: Holybro.

3 Configuration and Usage 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 in the folder `QGroundControl` 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. 12 with indications of relevant icons and how to access NuttShell.

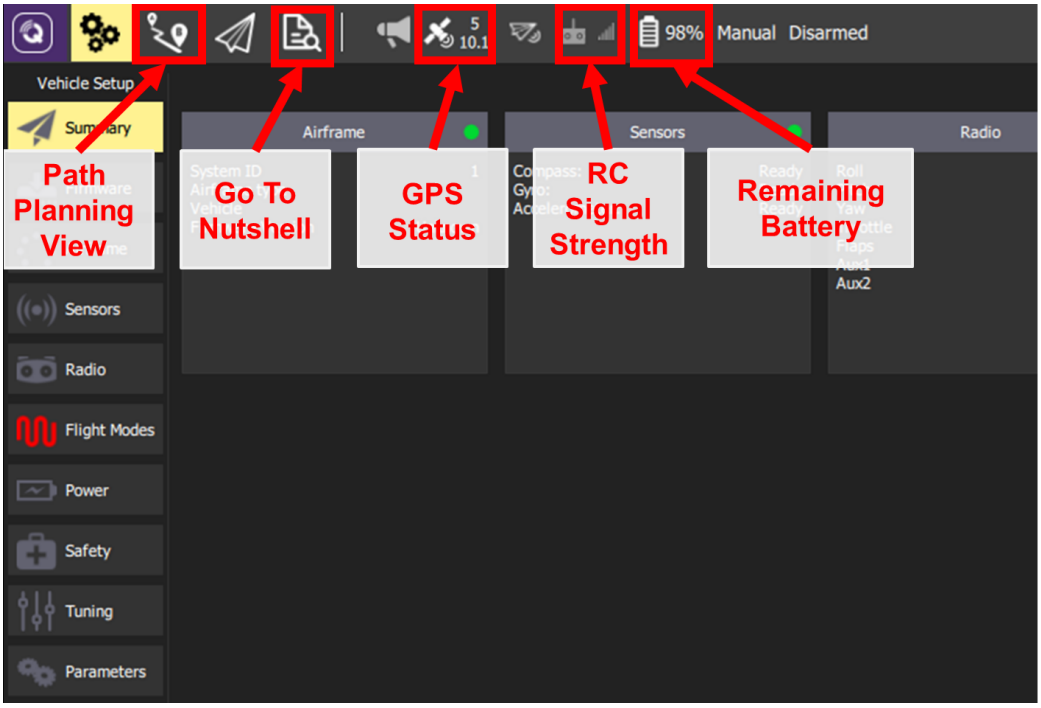


Figure 12: QGroundControl Interface.

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 a tool to access the underlying operating system. During flight tests, flight logging will be initialized with the boot sequence, but it can be stopped and restarted from the NuttShell (see Sec. 3.6).

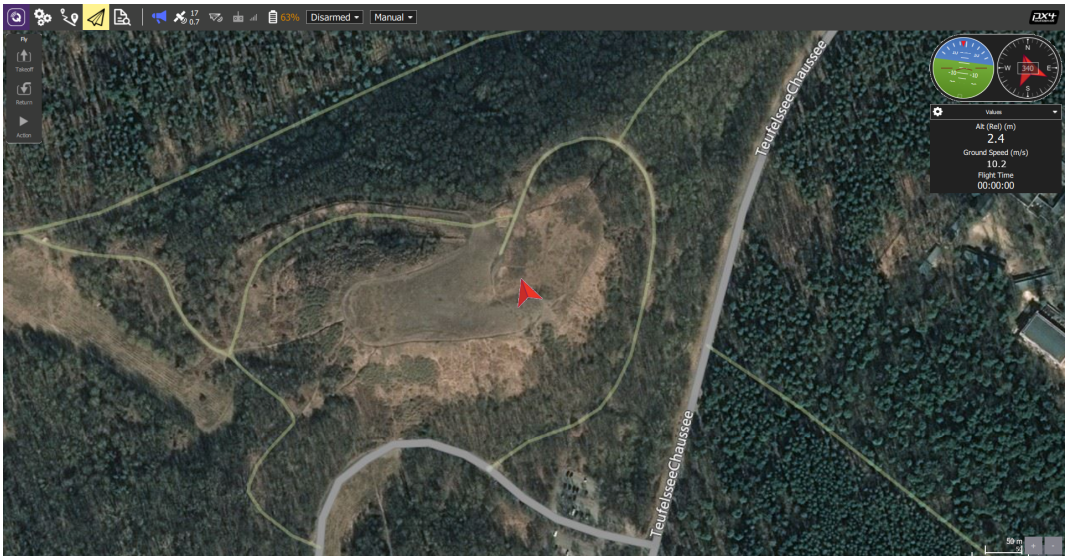


Figure 13: QGroundControl during a Flight.

Figure 13 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. 12) 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. 14) 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. 15.

The core is an STM32 microcontroller running NuttX, which is a minimized 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`.

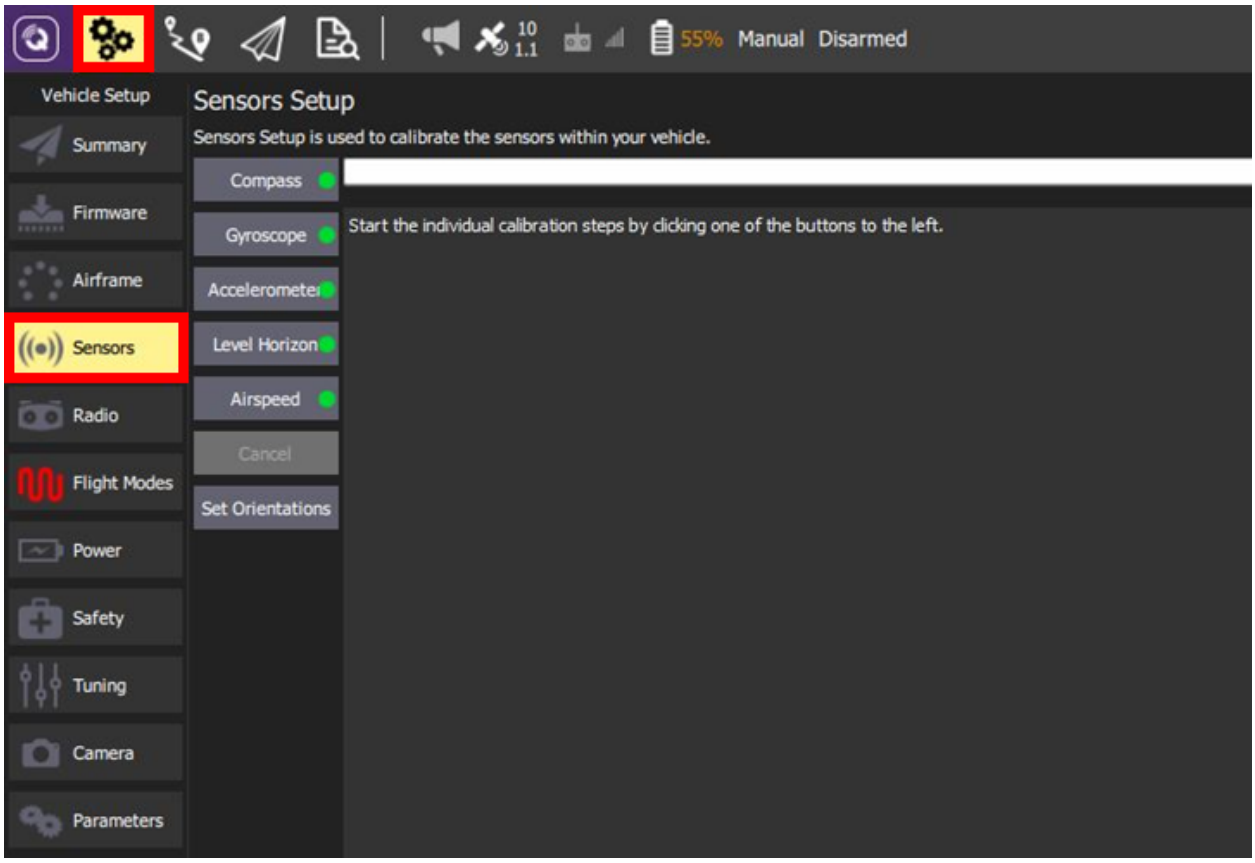


Figure 14: QGroundControl Calibration Interface.

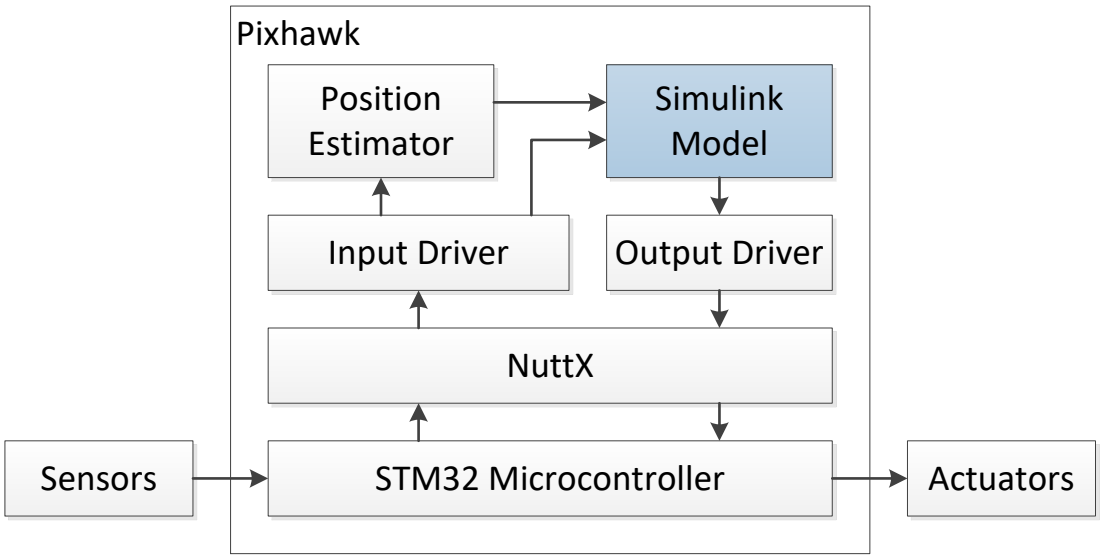


Figure 15: Pixhawk Architecture and PX4 Software.

3.5 Compilation and Software Upload

To implement new generated code as described in Sec. 4.4, 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.cpp`, `controller.h`, `rtmodel.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; this should be (`/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 16 shows the flow chart from MATLAB/Simulink model to executable software on the Pixhawk.

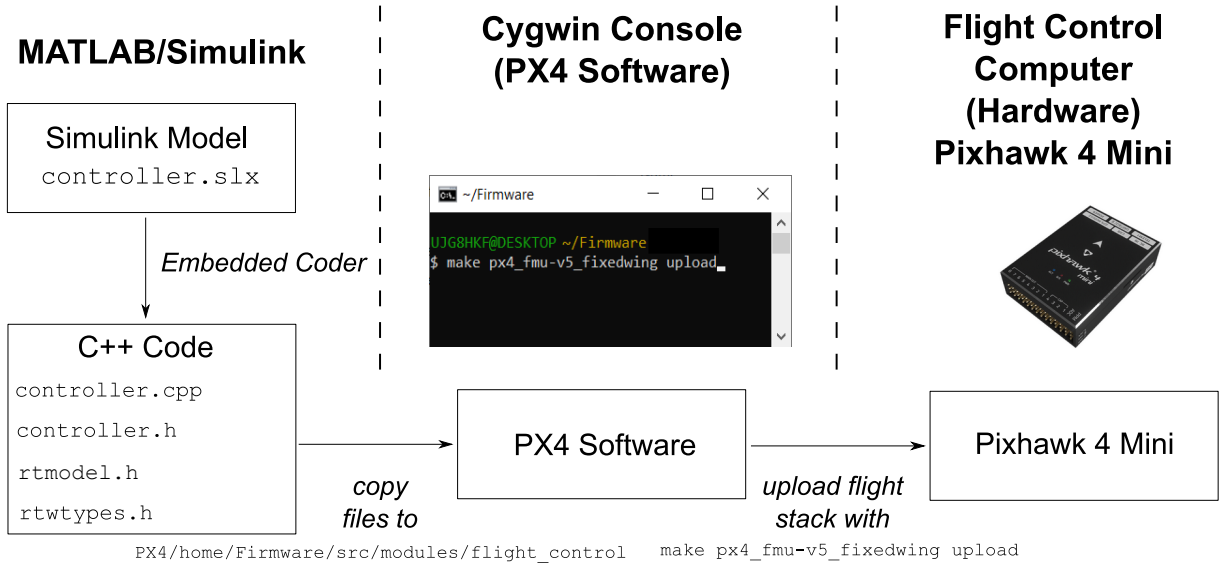


Figure 16: Flow Chart from MATLAB/Simulink Model to Executable Software on Pixhawk.

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. 17):

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.

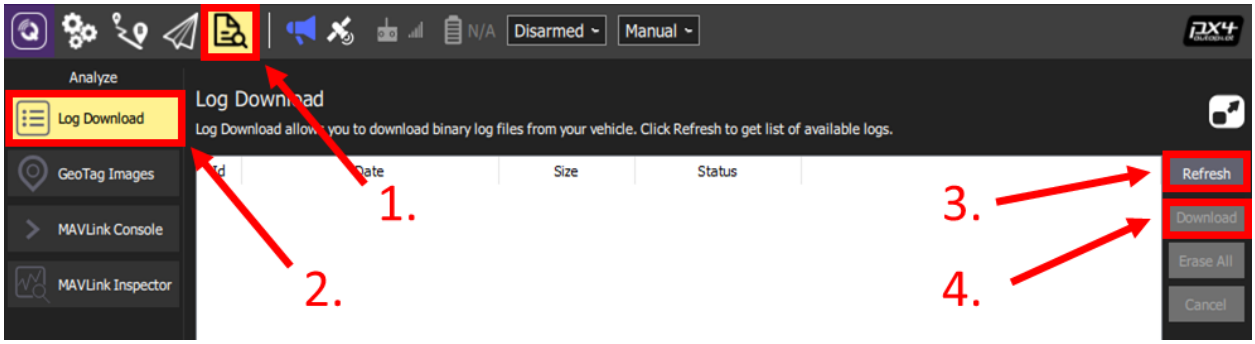


Figure 17: QGroundControl Access to Log Data (Order of Steps Highlighted).

The evaluation of flight data can be done either with the PX4 Flight Review or with MATLAB. The PX4 Flight Reviewer[8] is a web-based tool. The strength of this tool is its capability to visualize 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 visualize the flight log data will appear. To get the 3D flight playback, click `Open 3D View`.

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

3.7 Flight Test Evaluation with MATLAB

The analysis of flight data in MATLAB is carried out with a MATLAB script. Figure 18 shows the workflow.

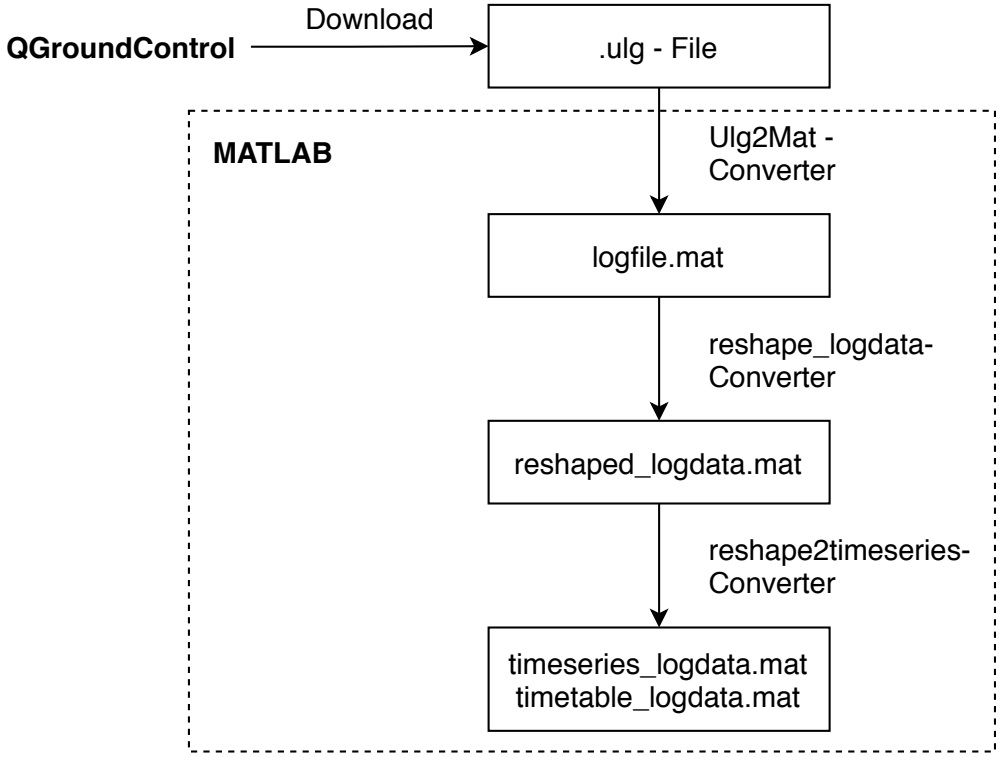


Figure 18: MATLAB Evaluation Workflow.

Ulg2Mat-Converter: The Ulg2Mat-Converter is a MATLAB-based tool to convert binary `.ulg` files to `.mat` files. MATLAB version R2019b or higher and the Mapping Toolbox need to be installed so that the tool is working properly. The software `Ulg2Mat.p` can be found in the folder `Matlab/Log Data` on the USB stick. To convert the binary, follow these steps:

1. Copy the `Ulg2Mat.p` script to the same directory as the `.ulg` file.
2. Open MATLAB and start the `Ulg2Mat.p` script by entering `Ulg2Mat` in the MATLAB Command Window.
3. A file selection window will appear. Choose the `.ulg` binary that shall be converted to `.mat` format.
4. Wait until all data are read. A progress bar will be updated representing the current status.
5. A window to save the generated file will appear. Choose a name for the generated `.mat` file.

The resulting `.mat` file contains a struct with entries of logged raw data with different timestamps. An overview of the most important logged data is given in Tab. 5.

Table 5: Logged Data Overview.

Logged Data	Title of Category	Sample Rate
Simulink output signals for actuator control	`actuator_controls_0_0`	$100\,\mathrm{s}^{-1}$
PWM signals for actuator control	`actuator_outputs_0`	$10\,\mathrm{s}^{-1}$
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`	$100\,\mathrm{s}^{-1}$
Accelerometer Data	`sensor_accel`	$100\,\mathrm{s}^{-1}$
Gyroscope Data	`sensor_gyro`	$100\,\mathrm{s}^{-1}$
Magnetometer Data	`sensor_mag`	$100\,\mathrm{s}^{-1}$
Log data from Simulink model (to be defined manually)	`Talon_log`	$100\,\mathrm{s}^{-1}$
Barometer sensor (pressure, air temperature)	`vehicle_air_data_0`	$10\,\mathrm{s}^{-1}$
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`	$10\,\mathrm{s}^{-1}$

Most of the named signals are given or processed in the MATLAB/Simulink model and will be discussed later in detail (see Tab. 9). It should be mentioned that the created `.mat` file contains all raw data with the following limitations:

- Coordinate axes do not coincide for all data sets. Based on the installation direction, the Pixhawk-logger uses two coordinate systems to log data. One coordinate system is aligned to the installation direction of the Pixhawk (e.g. IMU-Data). Another is aligned to the Airplane (e.g. quaternions of attitude data).
- Start time and end time of recordings vary with every data set. The PX4 software will start logging data after initializing each topic. For instance, GPS typically needs the longest time to be initialized, this can lead to delays of up to 30 seconds for recording the topic `vehicle_gps_position_0`.
- Not all data are stored in SI units (e.g. time is stored in µs).

reshape_logdata-Converter: To facilitate working with the log files, a MATLAB script `reshape_logdata.p` is provided in the folder `Matlab/Log Data` on the USB stick. 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 `reshape_logdata.p` 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 `talon_log_0`.

reshape2timeseries-Converter: Due to the real-time operating system running on PX4, the start and end time as well as the sample times of different sensors may vary slightly. To bring all the data to the same time scale and sample time, the `reshaped2timeseries.p` function can be used. The function is provided in the folder `/Matlab/Log Data` on the USB stick. This creates a variable named `ts_data` containing several structs for different sets of time series data. An overview is given in Tab. 6. The used timescale starts from $0\,\mathrm{s}$ and ends with the last time a sensor sent data. A frequency of $100\,\mathrm{Hz}$ with fixed time steps is used. The data is interpolated to fit this time scale. Most sensors start sending data after some seconds. Before the first data is sent, a value of `NaN` (Not a Number) is saved, so that all data sets can be plotted using the same time vector.

The same data is also provided in a timetable named `tt_data`, which allows using different evaluation methods.

Examples: Flight Test Evaluation with MATLAB

To view a time series, it can be called by the name given in Tab. 6.

```
ts_data.euler.phi
```

The stored time and data vector can also be accessed independently.

```
ts_data.euler.phi.Time    % access time vector
ts_data.euler.phi.Data    % access data vector
```

Additional information such as time increment or unit of the stored data is also available.

```
ts_data.euler.phi.TimeInfo    % access time information
ts_data.euler.phi.DataInfo    % access data information
```

To visualize the flight test data with the connected time, the `plot` command can be used.

```
plot(ts_data.euler.phi)
```

To plot several data vectors simultaneously, the `hold on/off` or `subplot` command should be used.

Another approach to visualize data is the `stackedplot` command, which is only defined for tables and timetables. It is the easiest way to show different data sets within the same timescale. They are called directly by their names as shown in the right column of Tab. 6. To plot multiple data vectors in the same graph, the corresponding names can be written in an additional set of braces.

```
stackedplot(tt_data, {{'phi', 'theta'}, 'psi'})
% plots phi and theta in one graph and psi in a second graph
```

It is also possible to only regard a specific time interval.

```
t_start = 60;          % Start time in seconds
t_end   = 100;         % End time in seconds
dt      = 1/100;       % Time increment in seconds
stackedplot(tt_data(seconds(t_start:dt:t_end),:), ...
            {{'phi', 'theta'}, 'psi'})
```

Table 6: Overview of Data in `ts_data` struct.

Struct	Time Series	Meaning	Unit
`euler`	`Phi`	Bank angle	$1\,\mathrm{rad}$
	`Theta`	Pitch angle	$1\,\mathrm{rad}$
	`Psi`	Yaw angle	$1\,\mathrm{rad}$
`rotation`	`p`	Roll rate	$1\,\mathrm{rad\,s^{-1}}$
	`q`	Pitch rate	$1\,\mathrm{rad\,s^{-1}}$
	`r`	Yaw rate	$1\,\mathrm{rad\,s^{-1}}$
`acceleration`	`acc_x`	Acceleration in x	$1\,\mathrm{m\,s^{-2}}$
	`acc_y`	Acceleration in y	$1\,\mathrm{m\,s^{-2}}$
	`acc_z`	Acceleration in z	$1\,\mathrm{m\,s^{-2}}$
`magnetic`	`magn_x`	Magnetic induction in x	$1\,\mathrm{kg\,A^{-1}\,s^{-2}}$
	`magn_y`	Magnetic induction in y	$1\,\mathrm{kg\,A^{-1}\,s^{-2}}$
	`magn_z`	Magnetic induction in z	$1\,\mathrm{kg\,A^{-1}\,s^{-2}}$
`airspeed`	`ias`	Indicated airspeed	$1\,\mathrm{m\,s^{-1}}$
	`tas`	True airspeed	$1\,\mathrm{m\,s^{-1}}$
`barometer`	`static_press`	Static pressure	$1\,\mathrm{kg\,m^{-1}\,s^{-2}}$
	`baro_alt`	Barometric altitude	$1\,\mathrm{m}$
`quaternion`	`q0`	Quaternion q0	1
	`q1`	Quaternion q1	1
	`q2`	Quaternion q2	1
	`q3`	Quaternion q3	1
`gps`	`lat`	Latitudinal coordinate	1°
	`lon`	Longitudinal coordinate	1°
	`alt_msl`	Altitude above sea level	$1\,\mathrm{m}$
	`course`	GPS course	$1\,\mathrm{rad}$
	`gs`	GPS ground speed	$1\,\mathrm{m\,s^{-1}}$
	`vs`	GPS vertical speed	$1\,\mathrm{m\,s^{-1}}$
`battery`	`discharged`	Battery discharged	$1\,\mathrm{mA\,h}$
	`remaining`	Battery remaining	1
`remote`	`rc_1` - `rc_12`	Remote control channel	PWM
`actuators`	`v_tail_r`	V-tail right deflection	$1\,\mathrm{rad}$
	`v_tail_l`	V-tail left deflection	$1\,\mathrm{rad}$
	`aileron`	Aileron deflection	$1\,\mathrm{rad}$
	`thrust`	Thrust	1
`log`	`log_data_1` - `log_data_20`	Log data	$-$

3.8 Selection of Autopilot Command

There are two ways of sending autopilot commands to the Simulink model:

1. **Using the rotary knobs of the remote control** (see Fig. 8): In the Simulink model, the signals ôf the remote controller can be accessed directly. Signals 11 (`u2_rc_11`) and 12 (`u2_rc_12`) are rotary knobs that can be used to define demand values for the flight control software. When using the `RC_Input_Normalisation` subsystem from the Simulink template, the output range for these channels is between -1 and 1. With a linear mapping, values for a desired altitude and/or speed can be set. However, this way of defining default values is not very precise and, especially during flights, dangerous as the pilot is distracted from the actual flight task.

2. **Using the NuttShell**: This method allows a clear separation between the (safety) pilot and the flight test engineer. While the (safety) pilot flies the aircraft, the flight test engineer can set demand values for the autopilot using QGC. The pilot can switch between manual and automatic flight by using the remote controller. To use this method, the flight test engineer has to use the NuttShell from QGC. The procedure is described in detail below.

Table 7: Predefined Default Values for the Simulink model (Flight Control Software).

Simulink Signal Name	**Default Meaning**
`r4_V_cmd`	Demand value for the airspeed
`r4_course_cmd`	Demand value for the heading
`r4_alt_cmd`	Demand value for the altitude
`r4_hdot_cmd`	Demand value for the vertical speed

Table 7 lists the default signals available in the Simulink model and their intended meaning. All these signals are floating point numbers. In principle, the user can assign a different meaning to these signals, such as a desired bank or pitch angle. However, care must be taken here.

With every restart, all values are set to zero. To set or change the demand values for the flight control software, the NuttShell of QGC is used. In the NuttShell, the statement

```
flight_control ap_select <V> <course> <alt> <hdot>
```

sets the command values. The expression `<V>` is replaced by the commanded speed, the expression `<course>` by the commanded heading, the expression `<alt>` by the commanded altitude and the expression `<hdot>` by the commanded vertical speed. After executing the statement, the new autopilot selection is printed onto NuttShell. Values can also be set to zero if it is intended that they should not be used.

The use of the command line is illustrated by the following example: A demanded speed of $15.5\,\mathrm{m\,s^{-1}}$, a heading of $325°$, an altitude of $40\,\mathrm{m}$ and a vertical speed of $0\,\mathrm{m\,s^{-1}}$ is set with the following line:

```
flight_control ap_select 15.5 325 40 0
```

Figure 19 shows the output in the NuttShell after entering the line. To view the current selection without editing it, type:

```
flight_control ap_select
```

Note, that the units of the autopilot selection are not specified, so the user must ensure that these are consistent with the design of the controller.

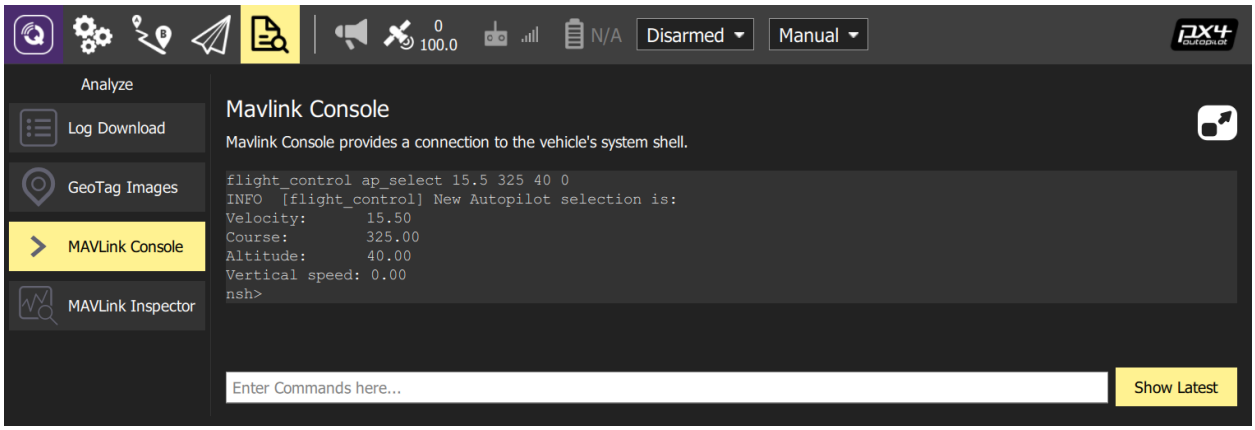


Figure 19: Example for setting the selection for the autopilot inputs

3.9 Command Waypoints in QGC

QGC can be used to send up to 15 waypoints to the Simulink model. These waypoints consist of: latitude, longitude, and altitude, as well as velocity as an option. To send the waypoints from QGC, follow these steps (also see Fig. 20):

1. Go to the `Path planning` view (see Fig. 12).
2. Select `+ Waypoint`.
3. Place waypoints in the desired order on the map (this defines the latitude and longitude of the waypoints).
4. Adjust the altitude in the right panel (this must be done for every individual waypoint).
5. (optionally) Activate the `Flight Speed` checkbox, if a velocity shall be commanded for the current waypoint, and enter a velocity.
6. Upload waypoints.

Note: The number of waypoints is limited to 15. If more waypoints are set up in QGC, only the first 15 are used.

Figure 20: QGroundControl Path Planning Interface.

After starting the upload, all waypoints are sent to the UAXS and fed into the Simulink model. The corresponding inports are:

Table 8: Simulink Inputs for Waypoints.

Input	**Description**
r4_wp_lat	Array with 15 Values for latitude
r4_wp_lon	Array with 15 Values for longitude
r4_wp_alt	Array with 15 Values for altitude
r4_wp_V	Array with 15 Values for velocity (-1 if not set)
u1_wp_num	Number of waypoints sent by QGC
u8_wp_timestamp	PX4 relative timestamp of first data reception

As given in Tab. 8, the latitude, longitude, altitude, and velocity are fed into the Simulink model by individual inputs. All of those are arrays with 15 elements of type `float32`. If more than 15 waypoints are sent, these are truncated to 15. If less waypoints are sent, the unused entries are set to zero. In case no velocity is commanded for a waypoint, the corresponding entry in the velocity array is set to -1. The timestamp represents the elapsed ms since startup of the PX4 system, i.e. since the first message from QGC was received. It is directly comparable to the input port `u8_relative_time` (see Tab. 9).

The initial values for all inputs are zero. When all commanded waypoints are received, the inputs are updated. It may take up to a few seconds from pressing the `Upload` Button until the data is available in the Simulink model. If a new set of waypoints is received (typically due to a new upload), the current inputs are overwritten. This can be detected by a change in the timestamp input.

3.10 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 QGC. Go to the `Vehicle Setup` view, select the `Parameters` option, and click the `Tools` button on the right side (see Fig. 21). Then choose "Load from file". Navigate to the USB Stick and choose the file `parameters_UAXS.params` which is in the folder `/QGroundControl`. Finally, restart the Pixhawk by unplugging the power connection and reconnecting.

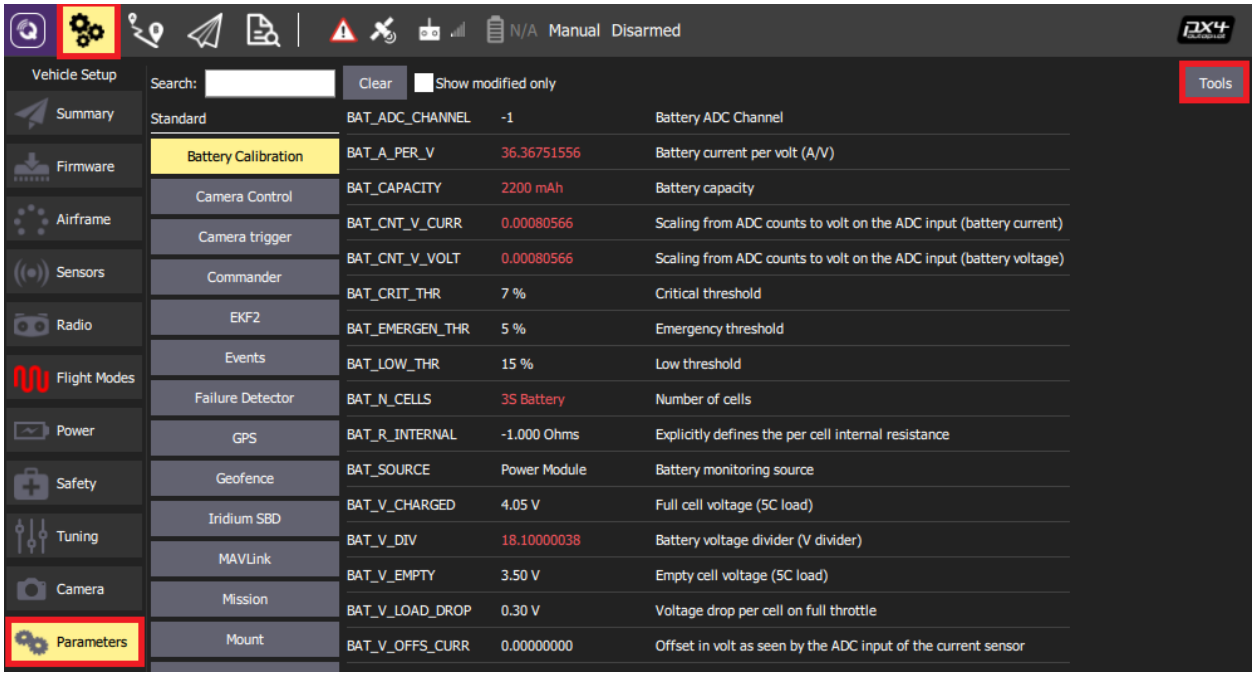


Figure 21: QGroundControl Interface to Load Parameters from File.

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, the Simulink Coder, and the Embedded Coder Support Package for ARM Cortex-M Processors 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. 22. 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 modeling the flight control logic.

The left side shows the inports of the flight controller. Those are split into two groups: `Sensors` and `Commands`. The inputs in the `Sensors` group consist of raw sensor data as well as preprocessed sensor data. The inputs in the `Commands` group are used to interact with the Simulink model in real-time. They are typically used to command reference values for different kind of controllers. The Simulink inports are:

Sensors

- 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,
- battery data, and
- system time.

Commands

- Inputs from the remote control,
- autopilot selection and
- way point selection.

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. 9. The inports of the MATLAB/Simulink model are 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!

When reviewing Tab. 9, special attention must be paid to the update rate of the GPS data and the range of the LiDAR sensor. This range is between $0.3\,\mathrm{m}$ and $12\,\mathrm{m}$, but the sensor will send random values above $12\,\mathrm{m}$, when it exceeds its upper range limit.

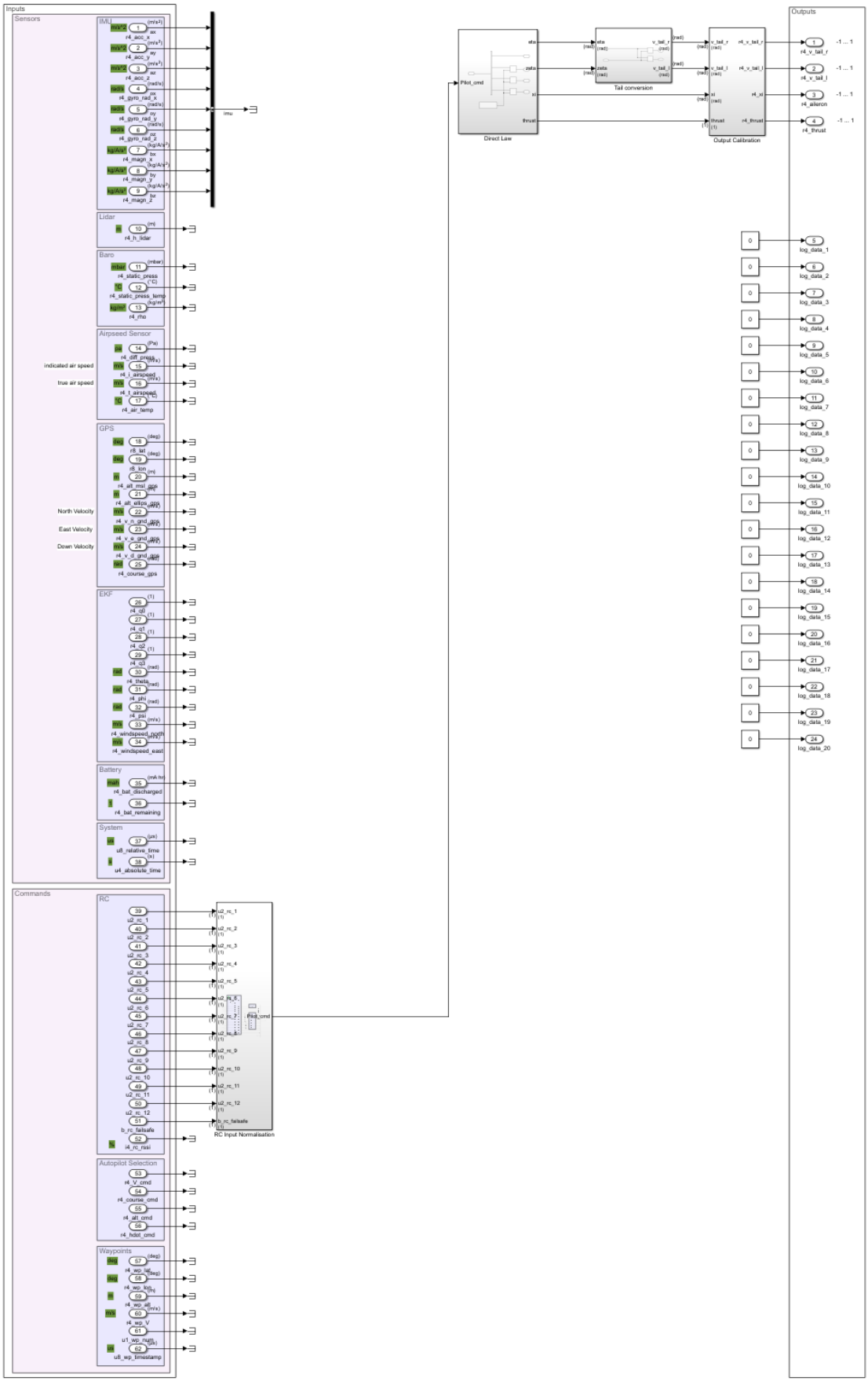


Figure 22: MATLAB/Simulink Template for the AlphaLink UAXS.

Table 9: 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	$\mathrm{m\,s^{-2}}$	$10\,\mathrm{ms}$	Acceleration measurement in x-axis direction
r4_acc_y	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-2}}$	$10\,\mathrm{ms}$	Acceleration measurement in y-axis direction
r4_acc_z	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-2}}$	$10\,\mathrm{ms}$	Acceleration measurement in z-axis direction
r4_gyro_rad_x	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{rad\,s^{-1}}$	$10\,\mathrm{ms}$	Gyro measurement about x-axis
r4_gyro_rad_y	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{rad\,s^{-1}}$	$10\,\mathrm{ms}$	Gyro measurement about y-axis
r4_gyro_rad_z	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{rad\,s^{-1}}$	$10\,\mathrm{ms}$	Gyro measurement about z-axis
r4_magn_x	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{kg\,A^{-1}\,s^{-2}}$	$10\,\mathrm{ms}$	Magnetometer measurement of magnetic induction in x-axis direction
r4_magn_y	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{kg\,A^{-1}\,s^{-2}}$	$10\,\mathrm{ms}$	Magnetometer measurement of magnetic induction in y-axis direction
r4_magn_z	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{kg\,A^{-1}\,s^{-2}}$	$10\,\mathrm{ms}$	Magnetometer measurement of magnetic induction in z-axis direction
r4_h_lidar	single	0.3	3.4028 *10^38	m	$10\,\mathrm{ms}$	Distance measured by LiDAR sensor (range only measurable from 0.3-$12\,\mathrm{m}$)
r4_static_press	single	0	3.4028 *10^38	mbar	$10\,\mathrm{ms}$	Calibrated static pressure measured by Pixhawk barometer sensor
r4_static_press _temp	single	0	3.4028 *10^38	°C	$10\,\mathrm{ms}$	Temperature measured by Pixhawk barometer sensor
r4_rho	single	0	3.4028 *10^38	$\mathrm{kg\,m^{-3}}$	$10\,\mathrm{ms}$	Air density calculated by Pixhawk barometer sensor
r4_diff_press	single	-3.4028 *10^38	3.4028 *10^38	Pa	$10\,\mathrm{ms}$	Calibrated differential pressure

Table 9: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
r4_i_airspeed	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$10\,\mathrm{ms}$	Indicated airspeed calculated by airspeed sensor
r4_t_airspeed	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$10\,\mathrm{ms}$	True airspeed calculated by airspeed sensor
r4_air_temp	single	-50	200	°C	$10\,\mathrm{ms}$	Air temperature measured by airspeed sensor
r8_lat	double	-90	90	deg	$500\,\mathrm{ms}$	Latitude measured by GPS sensor
r8_lon	double	-180	180	deg	$500\,\mathrm{ms}$	Longitude measured by GPS sensor
r4_alt_msl_gps	single	-3.4028 *10^38	3.4028 *10^38	m	$500\,\mathrm{ms}$	Altitude above mean sea level measured by GPS sensor
r4_alt_ellips_gps	single	-3.4028 *10^38	3.4028 *10^38	m	$500\,\mathrm{ms}$	Altitude above ellipsoid measured by GPS sensor
r4_v_n_gnd_gps	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$500\,\mathrm{ms}$	North velocity measured by GPS sensor
r4_v_e_gnd_gps	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$500\,\mathrm{ms}$	East velocity measured by GPS sensor
r4_v_d_gnd_gps	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$500\,\mathrm{ms}$	Down velocity measured by GPS sensor
r4_course_gps	single	0	2π	rad	$500\,\mathrm{ms}$	Course measured by GPS
r4_q0	single	-1	1	1	$10\,\mathrm{ms}$	Quaternion element 0 calculated by EKF
r4_q1	single	-1	1	1	$10\,\mathrm{ms}$	Quaternion element 1 calculated by EKF
r4_q2	single	-1	1	1	$10\,\mathrm{ms}$	Quaternion element 2 calculated by EKF
r4_q3	single	-1	1	1	$10\,\mathrm{ms}$	Quaternion element 3 calculated by EKF
r4_theta	single	$-\pi$	π	rad	$10\,\mathrm{ms}$	Pitch angle calculated by EKF
r4_phi	single	$-\pi$	π	rad	$10\,\mathrm{ms}$	Roll angle calculated by EKF
r4_psi	single	$-\pi$	π	rad	$10\,\mathrm{ms}$	Yaw angle calculated by EKF

Table 9: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
r4_windspeed _north	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$10\,\mathrm{ms}$	North wind speed calculated by EKF
r4_windspeed _east	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$10\,\mathrm{ms}$	East wind speed calculated by EKF
u2_rc_1	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM input of 1st RC channel
u2_rc_2	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM input of 2nd RC channel
u2_rc_3	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM input of 3rd RC channel
u2_rc_4	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM input of 4th RC channel
u2_rc_5	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 5th RC channel
u2_rc_6	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 6th RC channel
u2_rc_7	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 7th RC channel
u2_rc_8	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 8th RC channel
u2_rc_9	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 9th RC channel
u2_rc_10	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 10th RC channel
u2_rc_11	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 11th RC channel
u2_rc_12	uint16	982	2006	1	$10\,\mathrm{ms}$	PWM Input of 12th RC channel
b_rc_failsafe	boolean	0	1	1	$10\,\mathrm{ms}$	RC connection loss flag
i4_rc_rssi	int32	0	100	$\%$	$10\,\mathrm{ms}$	Receiver RC signal strength indicator
r4_bat_discharged	single	0	3.4028 *10^38	mAh	$10\,\mathrm{ms}$	Discharged capacity
r4_bat_remaining	single	0	1	$\%$	$10\,\mathrm{ms}$	Remaining battery
u8_relative_time	uint64	0	2^64-1	µs	$10\,\mathrm{ms}$	Time elapsed since startup
u4_absolute_time	uint32	0	2^32-1	s	$10\,\mathrm{ms}$	Seconds elapsed since 00:00 01.01.1970

Table 9: Interfaces of the MATLAB/Simulink Model.

Port Name	**Data Type**	**Min. Value**	**Max. Value**	**Unit**	**Sample Time**	**Description**
r4_V_cmd	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	after a new command	Autopilot command for velocity
r4_course_cmd	single	-3.4028 *10^38	3.4028 *10^38	rad	after a new command	Autopilot command for course
r4_alt_cmd	single	-3.4028 *10^38	3.4028 *10^38	m	after a new command	Autopilot command for altitude
r4_hdot_cmd	single	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	$10\,\mathrm{ms}$	Autopilot command for vertical velocity
r4_wp_lat	single [15]	-3.4028 *10^38	3.4028 *10^38	1°	after WP update	Array with latitudes of selected waypoints (WP)
r4_wp_lon	single [15]	-3.4028 *10^38	3.4028 *10^38	1°	after WP update	Array with longitude of selected waypoints
r4_wp_alt	single [15]	-3.4028 *10^38	3.4028 *10^38	m	after WP update	Array with altitudes of selected way points
r4_wp_V	single [15]	-3.4028 *10^38	3.4028 *10^38	$\mathrm{m\,s^{-1}}$	after WP update	Array with velocities of selected waypoints
u1_wp_num	single	0	255	1	after WP update	Number of selected waypoints
u8_wp_timestamp	single	0	2^64-1	µs	$10\,\mathrm{ms}$	Time since waypoints were updated
r4_v_tail_r	single	-1	1	1	$10\,\mathrm{ms}$	Actuator: right fin of V-tail
r4_v_tail_l	single	-1	1	1	$10\,\mathrm{ms}$	Actuator: left fin of V-tail
r4_aileron	single	-1	1	1	$10\,\mathrm{ms}$	Actuator: aileron
r4_thrust	single	0	1	1	$10\,\mathrm{ms}$	Actuator: thrust
log_data_n	single	-3.4028 *10^38	3.4028 *10^38	1	$10\,\mathrm{ms}$	Optionally loggable data

4.2 Linking of Inputs and Outputs

As an example, the Direct Law is implemented in the MATLAB/Simulink template (see Fig. 22). The Direct Law sets the actuators proportionally to the inputs, which are controlled by the remote control.

It will be shown how the inputs and outputs can be linked. Table 10 provides the name convention for setting the name prefix of the used inports and outports.

Table 10: Inport and Outport Name Convention.

MATLAB 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. 23) normalizes 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.

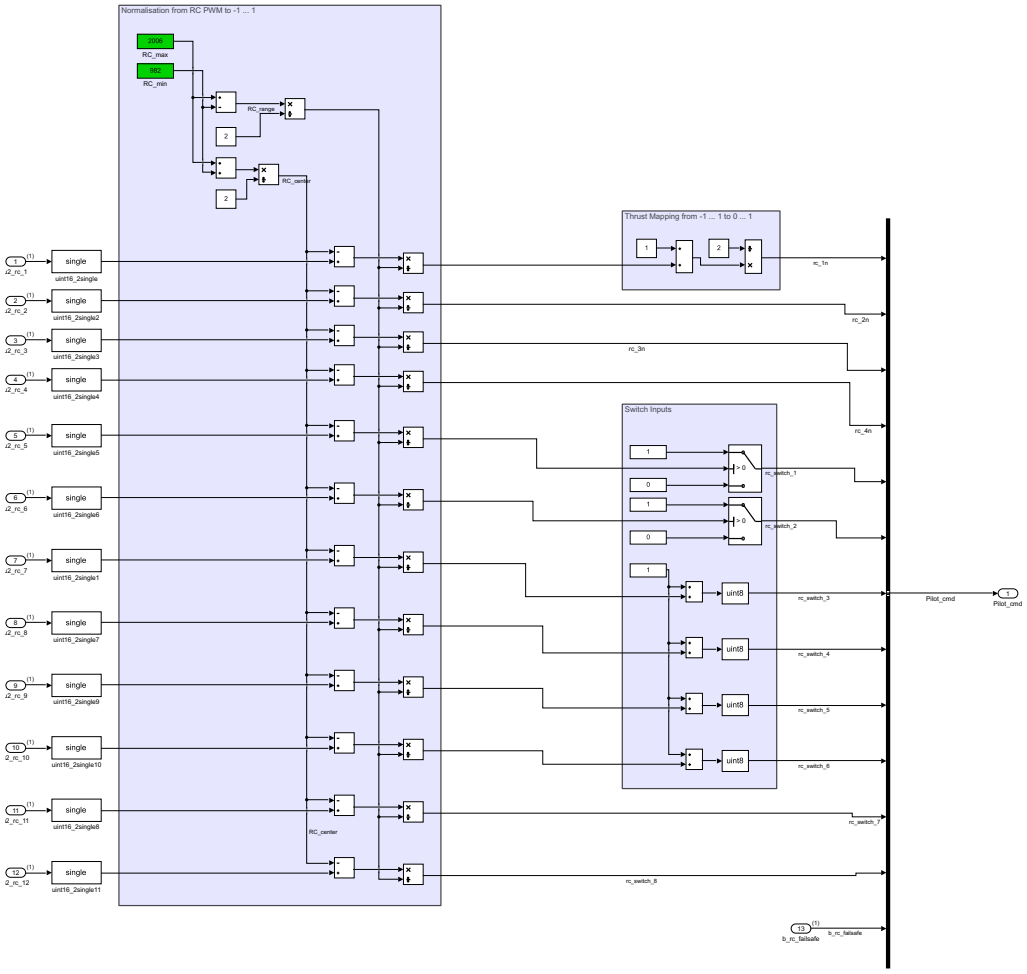


Figure 23: Subsystem RC_Input_Normalisation.

As shown in Tab. 9, the 12 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 (here `single` block), which are placed on the left side.
The blocks inside the first frame (placed at the left side) and the second frame (placed at the upper right side) convert the outputs to values 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 frame (placed at the right bottom side) are implemented here to show how processing signals from RC switches could look like.

Direct_Law Subsystem: This subsystem sets the signals of elevator, aileron and rudder to the actual deflection in `rad`. No priority switching for large control commands are implemented.

Tail_Conversion Subsystem: Here, the signal Eta (η) sets the V-tail flight control surfaces to generate a pitching moment, while the signal zeta (ζ) sets the V-tail flight control surfaces to generate 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. 24) 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 UAXS set, the outputs are pre-calibrated.

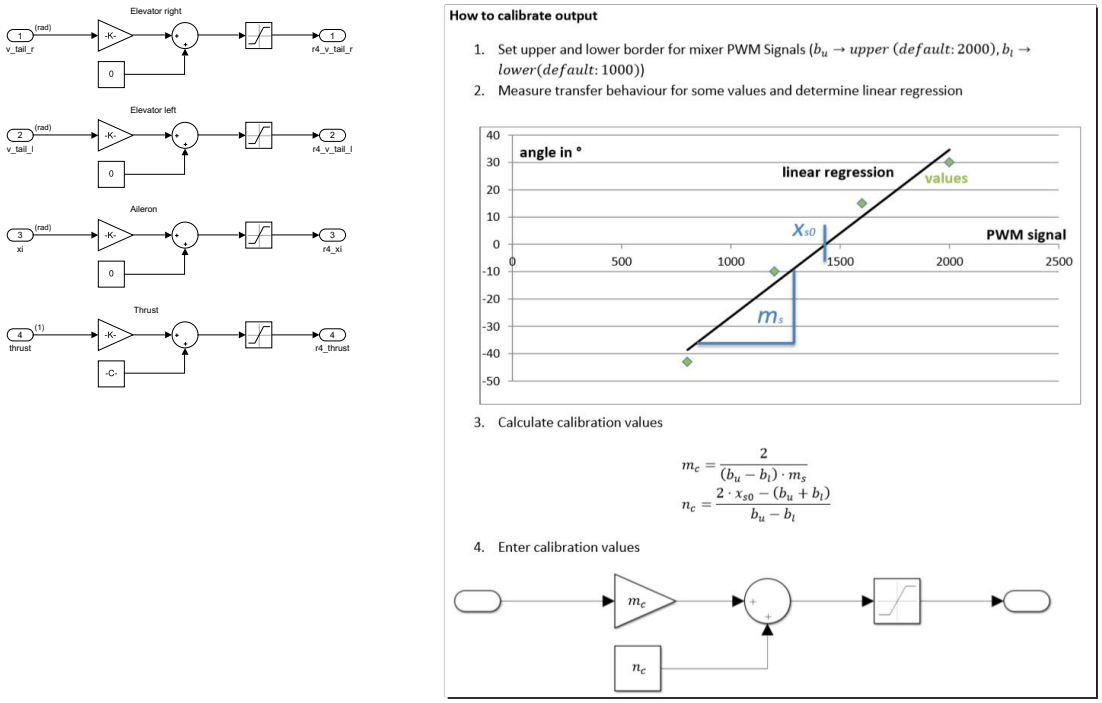


Figure 24: Subsystem Output_Calibration.

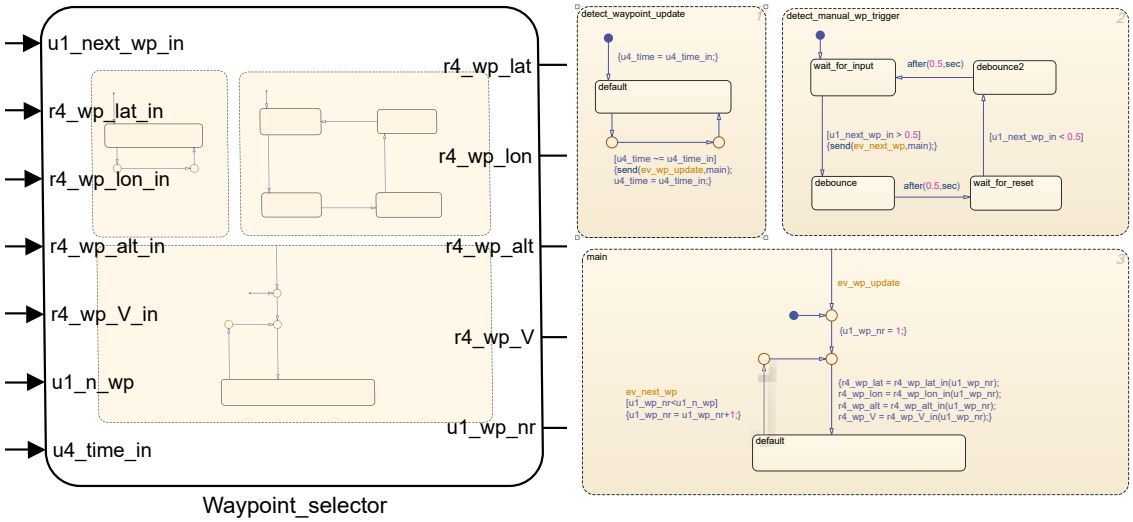


Figure 25: State Chart to Switch Waypoints. Left: Block for Integration into the Template; Right: Detailed View of the State Chart.

4.3 Integration of Waypoints

Section 3.9 describes how to specify waypoints in QGroundControl. Each waypoint is indicated by a quadruple consisting of longitude, latitude, altitude and speed. The World Geodetic System (WGS) 84 is used for the first three values. The individual values of the quadruple are available as an array with 15 elements each. Each of these arrays is an input to the Simulink model. To demonstrate the user how to work with arrays as inputs in a Simulink model, an example with the same name as the template (`Controller.slx`) is available in the `Simulink/Template Waypoint Selector` folder. In contrast to the normal template, the example contains the state chart `Waypoint_selector` (see. Fig. 25 left). The state chart takes the arrays `r4_wp_lat_in` (latitude), `r4_wp_lon_in` (longitude), `r4_wp_alt_in` (altitude) and `r4_wp_V_in` (speed) as well as the values `u1_wp_num_in` (number of uploaded waypoints), `u1_next_wp_in` (trigger for the next waypoint) and `u4_time_in` (timestamp of waypoint upload) as inputs. If less than 15 waypoints are selected, the residual space in the arrays gets initialized with zeros.

The state chart works as follows (see. Fig. 25 right): If new waypoints are uploaded via QGC (detected by the timestamp), the pointer is set to the first waypoint. If the trigger signal is set to high, the pointer goes to the next waypoint. However, if the pointer is already pointing to the last waypoint in the array, the pointer remains at the current position. The four values of the quadruple that represent the current waypoint (indicated by the pointer) are individual outputs of the state chart. These values can then be used as normal signals in Simulink. In addition to the quadruple of the currently selected waypoint, the position of the pointer is provided as an output of the state chart.

Note: This is only an example. You can modify the model according to your needs and also choose other options to advance the waypoints.

4.4 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 pre-installed, the software packages `MATLAB Coder` and `Simulink Coder` need to be installed. Inside the Simulink integrated development environment (IDE), the installation menu can be found by clicking `Get Add-Ons` as shown in Fig. 26.

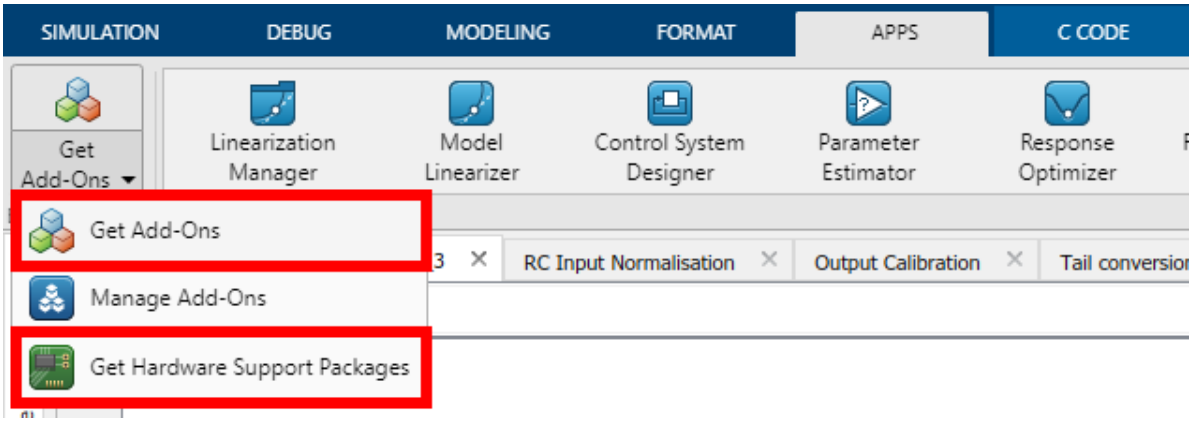


Figure 26: Simulink Add-Ons and Hardware Support Packages.

2. **Installing required Hardware Support Packages:** As shown in Fig. 15, 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. 26, 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. 27).

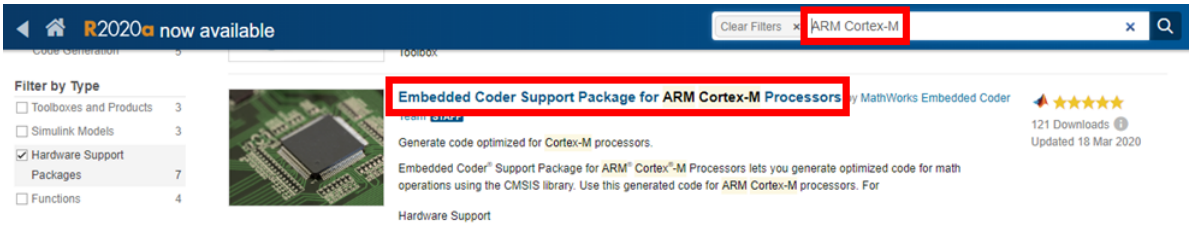


Figure 27: ARM Cortex-M Add-On.

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. 28.

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!*

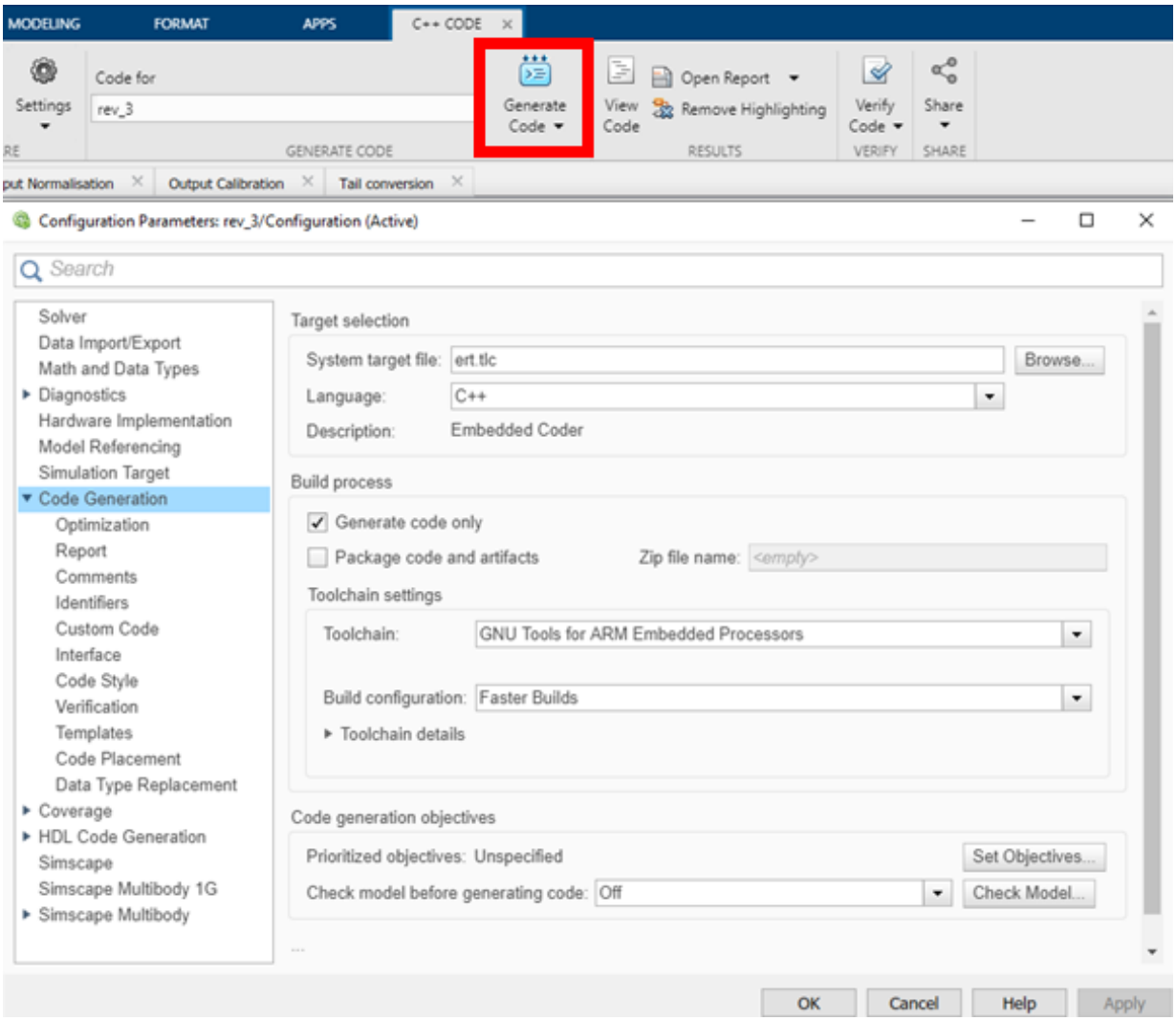


Figure 28: Settings for Code Generation.

The code can then be generated by clicking `Generate Code` (see Fig. 28). 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.

4.5 Modeling Guidelines and Development Recommendations

Model driven development (MDD) with MATLAB/Simulink is characterized 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.[9] 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.

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

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 working while the generated code 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`.

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 function` 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 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 usage of `If` blocks without `else` condition:** An alternative (`else`) path must always be provided for Simulink.

A USB Stick Directory