Experts in Team Innovation / Project in Robotics

September, 2020

version 1.0a

Jes Grydholdt Jepsen jegj@mmmi.sdu.dk
Kristian Husum Terkildsen khte@mmmi.sdu.dk
Ulrik Pagh Schultz ups@mmmi.sdu.dk

1 Introduction

This document contains the technical description of the Experts in Team Innovation MSc (EiT MSc) / Project in Robotics (PiR) project. The course participants are engineering students, who during course will try to work with real-life innovations and to work in interdisciplinary innovation groups.

This document has the following learning objectives;

- Create and follow a development plan based on agile project management tool
- Implement a working Unmanned Aerial Vehicle (UAV)-based system design towards a specific application/business case.
- Perform a system design, develop the required infrastructure, and implement and test the UAV-based solution in a real world environment.
- Develop and integrate new UAV software and payload subsystems
- Estimate the feasibility of implementing a UAV-based application in a real-life scenario using cost/benefit analysis as well as safety analysis

You and your team will develop the teamwork skills and work on developing an open innovation mindset whilst participating in a technical project, where you will be developing and testing potential business ideas.

Your team will work either intrapreneurial or entrepreneurial. The intrapreneurial teams will work with a company. The entrepreneurial teams on the other hand will work as if they were to start their own company around their idea. This course is based on experience-based learning. A small traditional curriculum is provided to create a secure platform but the most important learning content is your own reflection on the two experiences designed for you: the interdisciplinary group process and the innovation process.

The course develops your innovation skills and deepens your understanding of group processes. The course thus aims at strengthening your general engineering competences. When you have finished the course, you will be able to plan and implement a multidisciplinary innovation process based on and using relevant innovation models and methods and assess the feasibility of this.

Table 1 list a number of tasks relevant to the project.

Task	Description
1	Define an agile project management model to be used by the team and create a development schedule (section 2).
2	Define the requirements for UAV system (section 5.1).
3	Define the design goals for UAV system (section 5.2).
4	Perform a system design (section 5.3).
5	Define the workflow for your Unmanned Aircraft Systems (UAS) operation (section 5.4).
6	Define the final acceptance test (section 5.6).
7	Adapation of a UAV platform to perform (section 3).
8	Design and develop User Interface (section 5.5).
9	Design and develop software for your UAS operation (section 4).
10	Perform a risk assessment for the UAS operation (section 7.3).
11	Build a simulation environment to be used for simulation of the system functionality and for reliability tests (section 8.1).
12	Develop the required infrastructure to perform indoor laboratory tests (section 8.2).
13	Perform outdoor tests at HCA Airport. (section 8.3).
14	Perform a final accept test. (section 8.4).
15	Create a development budget for the project (section 6).
16	Submit a group project report (section 9).

Table 1: *Suggested tasks relevant to the project.*

2 Project management

2.1 Overview

You will be using agile project management to manage the project. Each team will agree on a project management model to be used throughout the course. It is fine to choose some elements from existing models such as Scrum[1] and leave others out. The only restrictions to this are defined in the following:

- The project management model must be described in the report and documentation of the implementation must be part of the electronic appendix to the report described in section 9. When choosing tools for the project management you may want to consider that only pdf is accepted as document format.
- Each week there will be a formal meeting with your project supervisor. You are welcome to define the setting and agenda for the meeting, however it must include a review of completed work, a discussion of challenges met, and prioritizing the tasks for the next iteration.

For managing your project, several web-based tools exists, such as Trello[2] and Microsoft Teams (you can create your own Team for your EiT project group and add an Action tab).

A primary course aim is to provide you the competence of participating in UAS research and industrial projects, and to this end we wish to promote good engineering project skills. We expect that you at all stages of the project seek external knowledge and sparing from your team members, fellow students, the teachers and other domain experts at the university etc.

When technical challenges occur, make an effort solve them but do not get stuck for days trying to solve a problem that might be easily solved by others. Make sure to discuss design choices, implementations, problems etc. at the iteration meeting, and when relevant please also contact us during the normal working hours.

[1] Scrum Software development
[2] Trello

It is recommended that each prioritized tasks is assigned to two of the team members thereby ensuring both parallel work and promoting knowledge sharing to obtain the best result.

During the project you will probably need to order some hardware. Waiting for hardware delivery often leads to delays in a project, and given the relatively short project period the outcome of this project is very sensible to delays. It is therefore recommended that you take this into consideration when prioritizing your tasks.

2.2 Schedule Development Overview

In addition to the agile project management, you also create a development schedule[3] in order to create an overview of tasks that need to be accomplished in a specific sequence within a given period of time during the project. To create a development schedule, the following needs to be defined:

- Project scope / goal
- Milestones throughout the project
- Tasks and estimated duration for each task
- Resources allocated to each task

Each team will agree on a development schedule model to be used throughout the course, such as a Gantt chart[4] (see example further down). The only restrictions to this are defined in the following:

- The development schedule model must be described in the report and documentation of the implementation must be part of the electronic appendix to the report described in section 9.
- The development schedule must contain the defined elements mentioned above.
- Each week we will update the development schedule during the weekly supervisor meeting.

For creating your development schedule, several of solutions[5][6][7] exists, both desktop and online tools.

[3] Schedule (project management)
[4] Gantt chart
[5] GanttProject
[6] OpenProject
[7] Google Sheet template

3 UAV platform

The UAV platform that you will be using during the project is shown on figure 1. You will receive it assembled, configured and ready-to-fly.

This section will give you an overview of the hardware used (flight controller, motors, etc.), and section 4 will describe the software configurations (uploading firmware, calibration, etc.) of the UAV platform.

Figure 1: *UAV platform*

The technical and flight specifications for the UAV platform can be seen on table 2.

Technical specifications		Flight specifications	
Weight	2500 g	**Endurance**	19 min
W x L x H	578 x 578 x 330 mm	**Payload capacity**	700 g
Wingspan	832 mm	**MTOW**	3200 g
Material	Plastic/carbon fiber	**Operating temperature**	5 - 45 °C
Airframe	Hexacopter		

Table 2: *Technical and flight specifications for the UAV platform*

Figure 2 shows the wiring diagram of the UAV platform, and the individual hardware blocks are described more into details (including relevant links and/or videos) in the following subsections. The pinout for the Pixhawk 2.1 is shown on figure 18 in Appendix A.

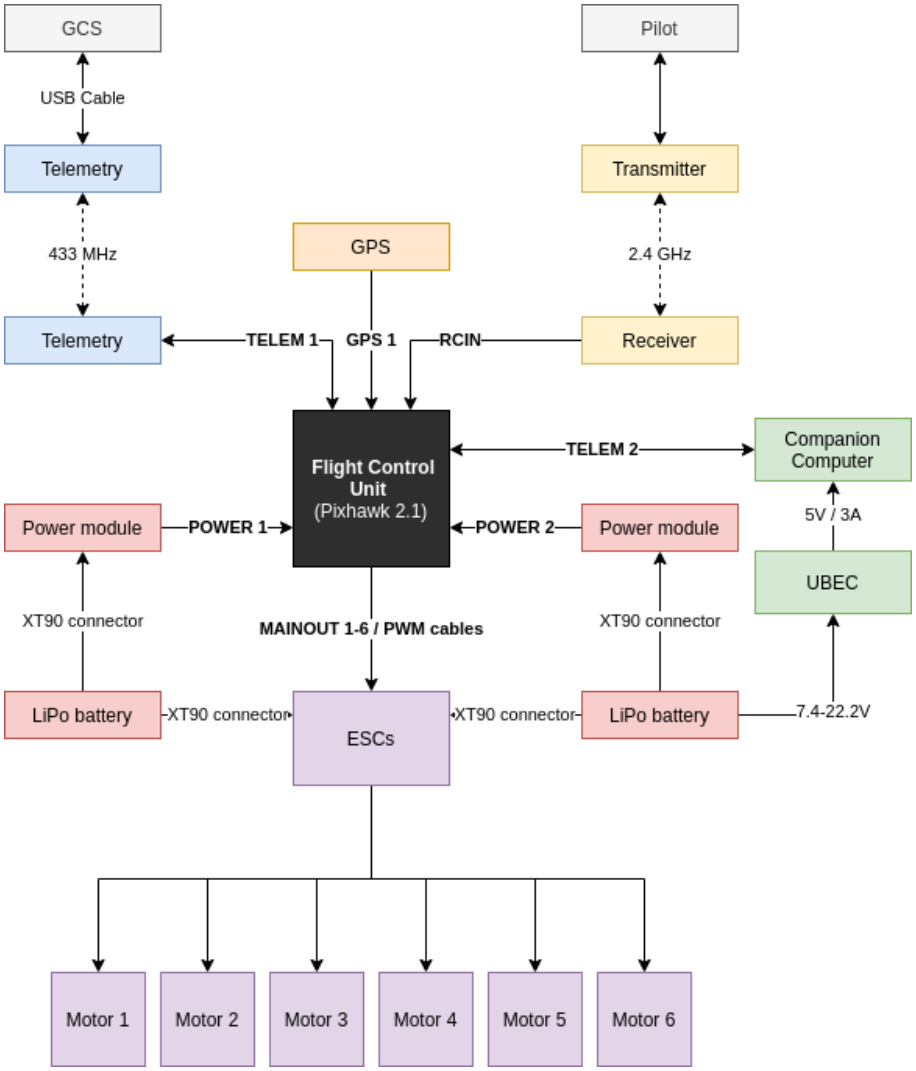


Figure 2: *Wiring diagram of the UAV platform.*

3.1 Flight Control Unit (FCU)

The Flight Control Unit (FCU) provided for this project is the Pixhawk 2.1 (Hex Cube Black) flight controller and is shown in figure 3.

Figure 3: *The Hex Cube Black flight controller.*
Source: PX4 Dev Guide: Hex Cube Black Flight Controller

PX4 Developer guide:

- Cube Wiring Quick Start

- Hex Cube Black Flight Controller

Video Guide(s)

- (1/9) Introduction to PixHawk 2.1: Introduction
- The Cube Pixhawk 2 Autopilot and Flight Controller Explained - All Versions Carrier Boards
- PixHawk/Mission Planner/ArduPilot Build for Beginners: Introduction

3.2 Power module

The power module provides a regulated power supply for the flight controller, along with information about battery voltage and current[8]. The provided power module for this project is a Power Brick Mini for Pixhawk 2.1, and is shown on figure 4

Figure 4: *Power Brick Mini for Pixhawk 2.1.*
Source: 3DXR: Power Brick Mini

PX4 Developer guide:

- Power Modules

3.3 Batteries

The batteries provided for the project are 2x Overlander 6250MAH 14.8V 4S 35C Lipo battery[9] (figure 5a), each with a XT90 connector.

Each group will not be provided a personal charger, but you will have access to a charger station in the Hangar at the SDU UAS Test Center[10]. The chargers available in at the charging station are iSDT D2 Dual Channel[11] chargers (figure 5b). When using the charger(s) available, you will also need a XT90 male to XT60 female converter (figure 5c).

IMPORTANT: Charging Lipo batteries can be very dangerous if you are not doing it right. It is important to use a LiPo compatible charger for LiPos, and ensure that the charger has the right settings for the battery. Wrong settings can in worst case course a Lipo Fire while charging. **Always ask if you are in doubt**, or watch the video guides listed further down.

[8] PX4 Dev Guide: Power Modules
[9] Overlander store
[10] SDU UAS Test Center, Beldringevej 252, 5270 Odense
[11] HobbyKing store: iSDT D2

(a) *Overlander 6250MAH 14.8V 4S 35C Lipo battery.*
Source: Overlander store

(b) *iSDT D2 Dual Channel Smart Balance Charger.*
Source: HobbyKing store: iSDT D2

(c) *XT90 male to XT60 female.*
Source: HobbyKing store: XT90 male to XT60 female.

Figure 5: Lipo and charger

PX4 Developer guide:

- Battery and Power Module Setup

Video Guide(s)

- iSDT D2 Charger review
- Quad RC LiPo Battery Voltage Meter and Alarm

Other(s):

- A Guide to Understanding LiPo Batteries

3.4 Motors, ESCs and Propellers

The provided motors, ECSs and propellers are from the T-Motor Air Gear 450[12] Combo set (figure 6), which consist of

- AIR2216 880kv
- T1045 self-locking propellers (CW/CCW) in plastic
- AIR20A V2 compact ESC

IMPORTANT: Never power up the UAV system with the propeller mounted, unless it is before a test flight either in the indoor environment (section 8.2) or an outdoor environment (section 8.3).

[12] T-Motor store: Air Gear 450

Figure 6: *T-Motor Air Gear 450.*
Source: T-Motor store: Air Gear 450

PX4 Developer guide:

- ESCs and Motors
- PWM Servos and ESCs (Motor Controllers)

Video Guide(s)

- RC Basics - Understanding Electronic Speed Controllers (ESC)

3.5 Global Positioning System (GPS)

The Global Positioning System (GPS) provided for the project is a Here 2 GPS module[13] (figure 7).

Figure 7: *Here2 GPS.*
Souce: 3DXR: Here 2 – GPS with CAN / Serial M8N

PX4 Developer guide:

- GPS and Compass
- HEX/ProfiCNC Here2 RTK GPS

3.6 Transmitter

The transmitter provided for the project is a Taranis Q X7[14] (figure 8a). The transmitter is powered by 2x Panasonic 18650 Li-Ion-3500mAh battery (figure 8b), which can be charged using the provided Ansmann Li-ION and Ni-MH charger (figure 8c).

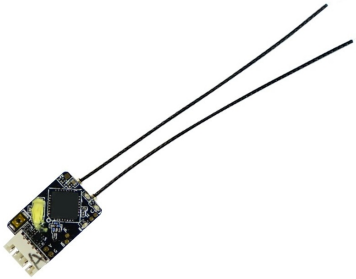
[13] 3DXR: Here 2 – GPS with CAN / Serial M8N
[14] FrSky: Taranis Q X7

The receiver provided for the project is a R-XSR receiver (figure 8d)

(a) Taranis Q X7
Source: RC-Netbutik

(b) Panasonic 18650 Li-Ion-3500mAh battery
Source: RC-Netbutik

(c) Ansmann Li-ION and Ni-MH charger.
Source: RC-Netbutik



(d) R-XSR receiver.
Source: RC-Netbutik

Figure 8: Transmitter and receiver setup

PX4 Developer guide:

- Radio Control Systems
- Radio (Remote Control) Setup
- Flight Mode Configuration

Video Guide(s)

- FrSky R-XSR Review

3.7 Telemetry

The provided telemetry link for the project is established using mRo telemetry 433 MHz[15] modules (figure 9). You will receive the pre-configured and ready to use (see the setting in Appendix C.4), but if

[15] mRo telemetry 433 MHz

you would like to make any changes in the telemetry link configuration, talk with your supervisor first.

Figure 9: *mRo telemetry 433 MHz modules.*
Souce: mRobotics store: mRo SiK Telemetry Radio V2 433Mhz

PX4 Developer guide:

- Telemetry Radios/Modems
- SiK Radio

Video Guide(s)

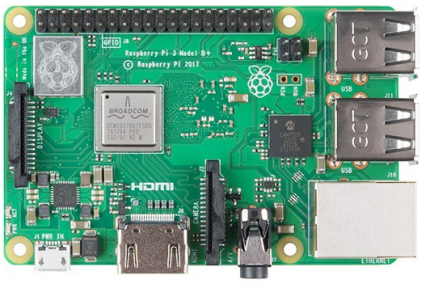
- (8/9) Introduction to PixHawk 2.1: Wireless temetry options (3DR Radios and FlightDeck)
- Example: Setting mRo radio parameters (also see Appendix C)

3.8 Companion Computer

The Companion Computer (CC) provided for this project is a Raspberry Pi 3 B+ (RPI3)[16] (figure 10a) with a 32 GB SD Card. For powering the RPI3, a 5V/3A Switching Power UBEC[17] (figure 10b) is provided. It can be used with 2-6S (7.4-22.2V) LiPoly batteries, and is connected to the UAVs batteries.

The interfaced using a FTDI 3.3V USB cable to the TELEM2 port on the Pixhawk 2.1. The TELEM2 port intended for this purpose, and the message format on this link is MAVLink[18].

[16] RaspberryPi.org
[17] 5V/3A Switching Power UBEC
[18] MAVLink Developer Guide



(a) Raspberry Pi 3 B+.
Source: RaspberryPi.org

(b) 5V/3A Switching Power UBEC.
Source: HobbyKing Store: 5V/3A Switching Power UBEC

Figure 10: Companion Computer setup

PX4 Developer guide:

- Companion Computer for Pixhawk Series

Video Guide(s)

- Connect a Raspberry Pi to a Pixhawk running Ardupilot/PX4

4 UAV Software

This section will give you an overview of the software configurations of the UAV platform and recommended Ground Control Station (GCS) and CC software.

4.1 Flight Stack

Pixhawk 2.1 supports multiple flight stacks, eg. PX4 Autopilot[19] or ArduPilot[20], but it is recommended to use PX4 Autopilot throughout the project since that is the flight stack that is used and supported in the SDU UAS Center, and the software tools that you will be provided have been tested and verified with the PX4 Autopilot. If you chooses to use another flight stack than the one recommended, it is not guaranteed that it can be supported by your supervisor(s) and you will be on your own.

The flight controller that you have been provided is flashed and configured with the latest stable version of PX4 (v1.10.1).

Figure 11 shows an overview of the building blocks of the flight stack, and figure 19 in Appendix A gives a more detailed overview of the software building blocks of PX4.

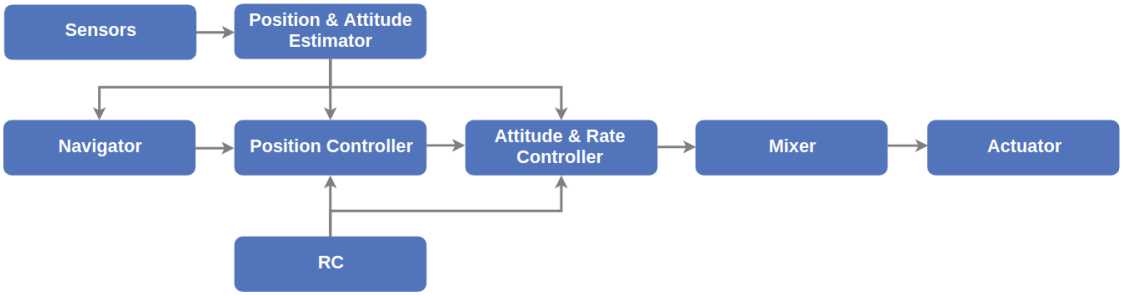


Figure 11: *Overview of the building blocks of the PX4 flight stack.*
Source: PX4 Dev: PX4 Architectural Overview

More detailed description of the PX4 Software Architecture can be found here:

- PX4 Developer Guide: PX4 Architectural Overview
- Overview of multicopter control from sensors to motors — PX4 Developer Summit Virtual 2020

4.2 Ground Control Station software

A GCS is normally a ground-based computer with a software application, that communicates with your UAV via wireless telemetry link. During the development it is recommended that you uses your laptop as GCS, since it is easy to transport and setup at different locations.

The GCS is typically used as a "virtual cockpit", where it displays real-time data on the UAV, such as its position, remaining battery, flight performance, etc.

A GCS can also be used to control the UAV, eg.

- control the flight of the UAV
- create and upload mission commands
- setting parameters (a list off the parameters can be found here)

Different GCS software is available, some of the most popular ones are

[19] PX4 Dev Guide
[20] ArduPilot Dev Guide

- QGroundControl[21]
- Mission Planner[22]
- Universal Ground Control Station (UgCS)[23]

4.2.1 QGroundControl (QGC)

As GCS software it's recommended that you use QGroundControl (QGC). It provides full flight control and vehicle setup for PX4 powered vehicles. You can see how to download and install QGC right here. If you chooses to use another GCS than the one recommended, it is not guaranteed that it can be supported by your supervisor(s) and you will be on your own.

One of the tasks in this project is to implement a User Interface (UI) (see section 5.5). If you choose to implement some kind of a graphical interface, then it might not be relevant for you to use QGC for your flights, rather you would only need it to set up parameters or do a calibration.

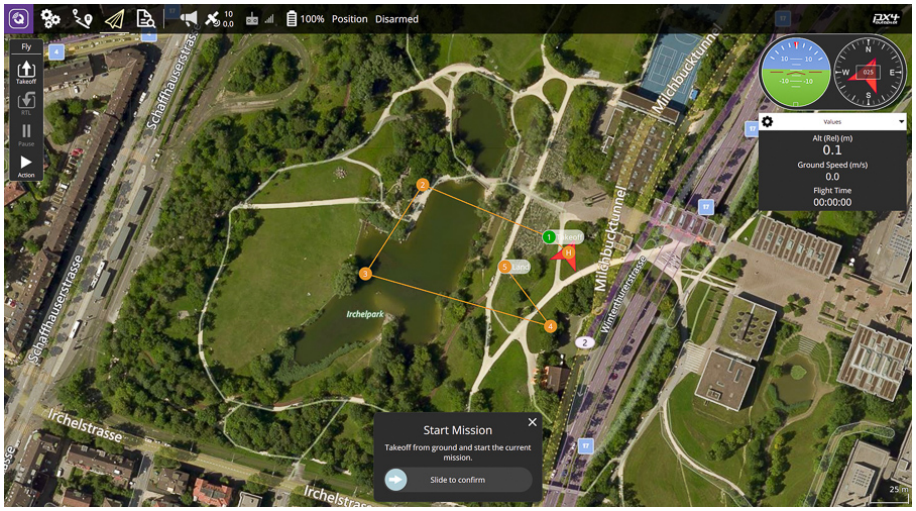


Figure 12: *QGC's user interface view.*
Source: QGroundControl User Guide

QGroundControl User Guide:

- QGroundControl User Guide
- QGroundControl Quick Start
- Download and Install

PX4 Developer guide:

- Finding/Updating Parameters

Video Guide(s)

- "Getting Started" guide for QGroundControl (Planning a Mission)

[21] QGroundControl webpage
[22] Mission Planner webpage
[23] UgCS webpage

4.3 Basic Configuration

When you receive the UAV platform is it already configured and ready to fly. The things that has been pre-configured (using QGC, see section 4.2.1) are

- Latest stable PX4 Firmware has been uploaded (see PX4 Dev Guide: Loading Firmware)
- Airframe set to Hexarotor X (see figure 13 and PX4 Dev Guide: Airframe Setup)
- Sensor calibration (see PX4 Dev Guide: Basic Configuration and QGC docs: Setup View)
  - **Sensor calibration is an important part of maintaining the UAV system**, so if you start experiencing that the performance (stability, positioning, etc.) of the UAV platform is starting to get worse, try re-calibrating the sensors mentioned below.
  - Flight Controller/**Sensor Orientation** (see PX4 Dev Guide: Flight Controller/Sensor Orientation)
  - **Compass** calibration (see PX4 Dev Guide: Compass Calibration)
  - **Gyroscope** calibration (see PX4 Dev Guide: Gyroscope Calibration)
  - **Accelerometer** calibration(see PX4 Dev Guide: Accelerometer Calibration)
  - **Level Horizon** calibration (see PX4 Dev Guide: Level Horizon Calibration)
- **Transmitter** / Radio Setup and Calibration (see PX4 Dev Guide: Radio (Remote Control) Setup)
  - Video guide(s) for channel setup, model, etc.
    - Taranis QX7 - First Initial Setup Steps
    - Taranis QX7 - Setting up switches
  - For setting up the transmitter, you can either do it directly on the transmitter or using the OpenTX Companion software on the laptop. You can download the default settings for the transmitter here.
- **Battery** and Power Module Setup (see PX4 Dev Guide: Battery and Power Module Setup)
- **ESC** Calibration (see PX4 Dev Guide: ESC Calibration)
- **Tuning** of the platform (see PX4 Dev Guide: Multicopter PID Tuning Guide)
- **Telemtry** radio setup (see PX4 Dev Guide: SiK Radio)
  - A more thorough setup guide can be found in Appendix C

You can find the default parameter list here.

IMPORTANT: If you start experiencing that the performance (stability, positioning, etc.) of the UAV platform is starting to get worse, try re-calibrating the sensors (guides linked above).

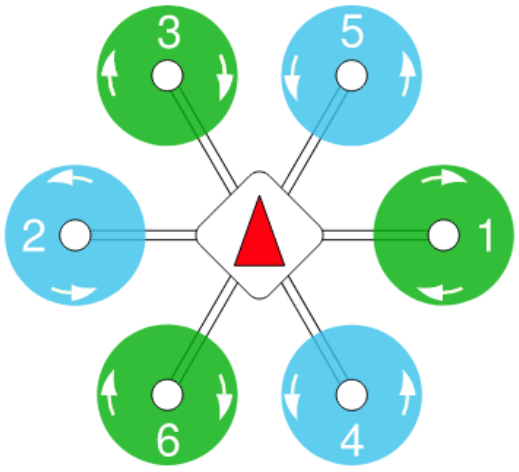


Figure 13: *Hexarotor x airframe setup.*
Source: PX4 Dev Guide: Airframes Reference

Video Guide(s)

- Youtube: PX4 Autopilot Setup Tutorial Preview
  - An older version of the Pixhawk and QGC is used in video, but the procedure is the same.

4.4 Radio setup (channels)

The following channels are setup when you receive the UAV system:

- **Channel 5** (marked with red on figure 14): **Flight mode** (see section 4.4.1)
- **Channel 6** (marked with blue on figure 14): **Kill switch** (see section 4.4.2)

Figure 14: *Taranis Q X7 with color-coded channels.*
Channel 5 (red): Flight mode control
Channel 6 (blue): Kill-switch
Source: RC-Netbutik

You can map other flight modes or use more channels by configuring the setting through QGC (see QGroundControl: PX4 Pro Flight Mode Setup). If you would like to use more channels, it could be for activating `OFFBOARD`[24], `RETURN`[25] or `LAND`[26] mode.

PX4 Developer guide:

- PX4 Flight Modes Overview
- PX4 Flight Modes (Multicopter)
- Radio (Remote Control) Setup

4.4.1 Flight modes

Flight Modes define how the autopilot responds to user input and controls vehicle movement. You can find a description of each flight mode here for both fixed wing, multicopter and VTOL. The flight modes that will be activated based on the PWM value of that channel. The mapped flight modes are:

[24] PX4 Dev Guide: Offboard Mode
[25] PX4 Dev Guide: Return Mode
[26] PX4 Dev Guide: Land Mode

- **1. position (top)**, channel 5: Stabilized Mode (see Manual/Stabilized Mode (Multicopter))
- **2. position (middle)**, channel 5: Position Mode (see PX4 Dev Guide: Position Mode (Multicopter))
- **3. position (bottom)**, channel 5: Mission Mode (see Mission Mode)

4.4.2 Kill-switch

The kill-switch is a safety switch that immediately stops all motor outputs, so be careful only to use it during a flight in the event of a serious problem or emergency where you are not able to activate Return mode or make a manual controlled landing. When activating the kill-switch, the drone will fall and crash, which will cause damages and the UAV system and whatever it hits.

- **1. position (bottom)**, channel 6: Kill-switch enabled
- **2. position (top)**, channel 6: Kill-switch disabled

4.5 Companion Computer software

As mentioned in section 3.8, you will be provided a CC with a SD card. You will need to run an Operating System (OS) on the CC in order to be able to more high level applications using a CC. It's up to you what image that you would like to run on the CC, but it is recommended to use Ubuntu Mate[27] (either 18.04 or 20.04) or Ubuntu Server image[28]. It's also recommended that the software architecture on the CC is Robot Operating System (ROS)-based.

The provided SD card in the CC is already pre-loaded with a OS that is ready-to-use. It has the following software installed:

- OS: Ubuntu Server 18.04
- ROS: ROS Melodic, see Appendix D
  - MAVROS (http://wiki.ros.org/mavros)
  - Created workspace (`catkin_ws`)
- Created `udev` rule for the serial device to the Pixhawk 2.1

You can find the pre-build CC image here, and see Appendix F to getting started with running MAVROS on the CC.

If you choose to use another OS for the CC than the one recommended, it is not guaranteed that it can be supported by your supervisor(s) and you will be on your own.

Furthermore, section 3.8 mentions that the message format between the FCU and the CC is the MAVLink. For handling the Mavlink messages between the CC and Pixhawk 2.1, use one of the already existing libraries, eg.

- MAVROS[29] - MAVLink extendable communication node for ROS with proxy for Ground Control Station (recommended solution).
- MAVSDK[30] - a MAVLink Library with APIs for C++, iOS, Python and Android (has a few limitations, e.g., in how you can change flight modes, but should be sufficient for your needs).

When controlling the UAV platform using the CC, remember to use the `OFFBOARD`[31] mode. This mode is intended for companion computers and ground stations.

PX4 Developer guide:

[27] Ubuntu Mate download
[28] Ubuntu Server download
[29] ROS Wiki - MAVROS
[30] MAVSDK Developer guide
[31] PX4 Dev: Offboard mode

- Companion Computer Setup
- Offboard mode
- Robotics using ROS

5 UAV system design

5.1 Requirements

Specify a set of requirements[32] that your UAV-based solution aims to satisfy. A requirement often describes a desired

- functionality of the system or product
- property/characteristic of the system or product.
- capability of the system or product.

Try to define your requirement is such a way that you are able to quantify them, and actually verify the specific requirements, eg. during the acceptance test.

5.2 Design goals

Specify a set of design goals that your UAV-based solution aims to satisfy. Design goals are somewhat similar to requirements, except that they are less strict compared to requirements but they will expectedly lead to a more optimal solution.

5.3 System design

A system design for the UAS-based system must be performed, defining the system architecture, subsystems, modules, their interfaces, and protocols etc. The aim of the system design is to satisfy the specified requirements you have defined in section 5.1.

Where possible Commercial Off The Shelf (COTS) technology should be used.

5.4 Workflow

A workflow[33] is a visual way of representing the flow, step-by-step, of a task or a process from start to end. Describe the task that you want to solve using a workflow diagram.

5.5 User interface

The UI is to be used by end user, and should work as the human-computer interaction and communication between the system and the user, eg.

- summon a UAV to a specific location
- show relevant information, such as the estimated time of arrival of the requested UAV, remaining battery, etc.

The UI may be implemented as

- an app
- webpage
- external control center

[32] Requirement
[33] Workflow

5.6 Final acceptance test

An acceptance test[34] is a way to validate the performance the system and all subsystems, and determine if the requirements for your UAV-based solution are met.

Define a final acceptance test for your solution, and describe into details how the test should be conducted.

The final acceptance test should be defined together with the user/customer of the system. This ensures that the system that is being developed meets customer/user expectations.

Furthermore, as a part of your development schedule (section 2.2), create a list of smaller tests that you would like to conduct, eg. testing and verifying the functionality of a subsystem, that test-by-test will lead to the final acceptance test. Section 8 gives an overview of the testing environments that are available.

[34] Acceptance testing

6 Budgeting

6.1 Development budget

The development budget to be included in the project report should list the expected cost in terms of human resources (divided on required technical expertise) and cost of equipment, materials, required external consultancy, required access to test facilities etc. Any administrative cost such as office rental, accounting etc. not directly related to the project may be disregarded.

6.2 Pilot project budget

A pilot project is an initial small scale implementation of the system in order evaluate feasibility, duration, cost, adverse events, and improve the design prior to performance of a full-scale implementation.[35].

The budget for a single installation must include all relevant expenses for installation, operation, any relevant subscriptions and periodic service and maintenance for a single installation. The budget must present the actual cost without profit margin.

6.3 Operating budget

The operating budget takes into consideration what expenses that your system expects to have in the future (fixed costs, variable costs, etc.), as well as the income it predicts to make over a period of time, typically a year.[36]. Estimate the expected future costs and income for the system over the next year.

[35] Pilot experiment
[36] Operating budget

7 Risk and Safety

7.1 Practice

To be a good pilot, you will need to learn the basic and advanced UAV flying maneuver and practice them so that you get more experienced with taking the control of the UAV system and land it safely, when the system is not behaving as expected. Here are some example of basic maneuver that will help you improving your flight skills:

- Take off and landing
- Hovering in one place
- Hovering in different heights.
- Forward, backward, right, left
  - Try rotate the UAV System in different angles, and do the same commands so that you get comfortable flying UAV without it facing the same direction as you.
- 360 Yaw (Hover and rotate the quadcopter.)
- Flying in Square patterns (with same orientation)
- Flying in Square patterns (with orientation pointing towards next corner)
- Flying in Circle patterns (with same orientation)
- Flying in Circle patterns (with orientation pointing inwards)
- Figure 8 flying (with orientation pointing where you are going)
- Else, see PX4 Dev Guide: Flying 101 or practice videos on Youtube:
  - Basics of Drone Flight - Takeoff, Controlled Hovering and Landing
  - Basics of Drone Flight - Figure 8 Flying

It's always a good idea to start out with practicing using a simulator, which is why we as part of this material provide simulation environments modeling the SDU UAS Hangar and the outside environment of HCA Airport.

Practice the basic maneuver in both Manual/Stabilized Mode and Position Mode.

It is recommended that you **dedicate one team member to be pilot during all flights,** so you would need to decide who in the team will be the primary pilot. Remember, when flying the UAV system, always wear a helmet and safety glasses. Being struck by a UAV or its propellers can cause severe injuries.

7.2 Drone logbook

Document all your flights in a drone logbook. It is up to you how to document it (eg. simple text file, an existing drone logbook solution, etc.), as long it contains the most relevant flight informations[37]:

- the drone,
- the control station,
- the pilot,
- place or area for the flight
- date and time for the flight,

[37] TBST: Order on flights with drones in built-up areas, Chapter 5

- duration of the flight,
- maximum flight level,
- type of operation (e.g. VLOS, BVLOS), and
- any problems, incidents or accidents.

You will be using DroneLogBook for this purpose. When you have selected your primary pilot, contact your supervisor and that person will then get an account for logging the flight using DroneLogBook.

7.3 Risk assessment

The risk assessment should be performed according to the Specific Operations Risk Assessment (SORA) package[38]. The SORA is a tool from EASA released in July 2017 and as such very little experience concerning the use of the SORA exists. Trafikstyrelsen[39] welcomes applications based on the SORA. Figure 15 shows a diagram of the SORA process.

When assessing the risk associated with the UAS operation, inspiration may be found in [5, 9, 4, 3, 8]. In this project a simplified but adequate approach to the risk assessment is desired.

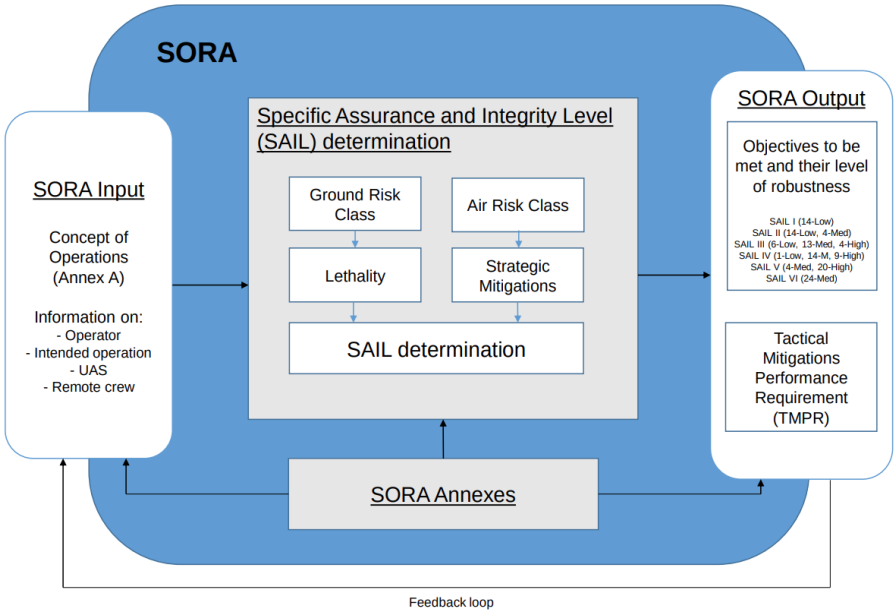


Figure 15: *SORA overview. Source: link*

7.4 Safety

PX4 has a number of safety features and failsafe that can help protecting your UAV platform if something goes wrong.

- Each failsafe defines some conditions that if not fulfilled will trigger a failsafe action.
- Each failsafe action is a set of actions that will be executed if it is triggered.

Some of the most common failsafe actions are:

- Hold mode: The vehicle will stop and hover at its current position.
- Return mode: The vehicle will return to safety, eg. its home position, depending on the settings.

[38] JAR doc 06 Specific Operations Risk Assessment(SORA) package
[39] droneregler.dk

  - Return Mode Settings

- Land mode: The vehicle will stop and land immediately.
  - Land Mode Settings
- Flight Termination: Similar to the kill-switch (see section 4.4.2), all the motor outputs will stop immediately, so careful only to use it in the event of a serious problem or emergency where the other failsafe actions isn't an option, eg. if a failure has been detected (Failure Detector).

The most common failsafes that has been pre-configured on the UAV Platform are:

- Low Battery Failsafe: The configuration of the Low Battery Failsafe has three level values (Warn > Failsafe > Emergency):
  - **Warn** (15% capacity): Warning messages will be sent to QGC if capacity drops below the respective level.
  - **Failsafe** (7% capacity): Activate Return mode if capacity drops below the respective level.
  - **Emergency** (5% capacity): Activate Land mode if capacity drops below the respective level.
  - For **indoor flights**: The failsafe should be Land mode instead of Return mode.
- RC Loss Failsafe: The RC Loss failsafe is triggered if the RC transmitter link is **lost more than 5 seconds**. The failsafe action is set to Return mode.
  - For **indoor flights**: This should be **disabled**.
- Data Link Loss Failsafe: The Data Link Loss failsafe is triggered if a telemetry link is **lost more than X seconds**. It is **disabled** by default, but you can set the failsafe action is set to activate Return mode if the data link is lost more than 10 seconds.
  - For **indoor flights**: This should be **disabled**.
- Geofence Failsafe: A basic Geofence has been define, which is a simple cylinder with a **max radius** of 200 meters and a **max altitude** of 100 meters centered around the home position. The Geofence Failsafe will be triggered if the UAV System moves outside the radius or above the altitude specified. The failsafe action is set to Return mode.
  - For **indoor flights**: This should just give you a **warning**.
  - You can define more complicated Geofence geometries using QGC (see Geofence).

You can find the default parameter list here.

IMPORTANT: If you would like to change / remove / add any of the pre-configured failsafes, talk with your supervisor first and discuss the failsafe change(s).

PX4 Developer guide:

- Safety Configuration (Failsafes)

Other:

- QGroundControl: Safety

8 Testing environments

This section will give an overview of the different testing environments that are available during the course.

When testing during the development is it good practise to starts with verifying that the most basic functions of the system works as expected in the simulated environment, and then step-by-step further additional functionalities are added in each test and it is verified that the added functionality works. When you feel that you are ready to test it on the actual UAV platform, then start over with testing the most basic functionalities in a more controlled environment, eg. an indoor positioning system, and again, step-by-step add and test more functionalities. When tested and verified in the indoor environment, it is ready to test the UAV platform in a outdoor environment, eg. the test field at HCA Airport or another location that makes sense for the solution that you are developing. The described development process is illustrated on figure 16.

Figure 16: Development workflow

8.1 Simulation environment

A simulation environment must be set up using PX4 Software-In-The-Loop (SITL)[40] and Gazebo[41]. The simulation environment should be used for simulation of the overall UAS functionality including user interface, link to an external control center etc. thereby reducing development time, as performing lab- and real world tests require considerable more resources.

In addition the simulation environment could be used for stress testing the reliability of the functionality by defining an automatic test scenario generator thereby running a high number of tests. Inspiration can be found in [7, 10]. The output from such tests may substantiate the UAS reliability documentation needed for the SORA.

To get started with the PX4 SITL simulation environment, see Appendix D.

Video Guide(s)

- Testing the EiT Playground (PX4 + ROS + Gazebo) - Part 1
- Testing the EiT Playground (PX4 + ROS + Gazebo) - Part 2

8.2 Indoor test environment

The hangar in SDU UAS Test Center[42] at HCA Airport can be used for test flights using the Motion Capture System[43], called OptiTrack[44].

The indoor positioning system is based on OptiTrack which provides centimeter-level precision and thus can serve both as a replacement for GNSS and as a system for logging positioning data during experimental work.

To get started with the PX4 SITL simulation environment, see Appendix E.

More detailed description of the OptiTrack system will be updated

[40] PX4 Dev Guide: Simulation
[41] gazebosim.org
[42] SDU UAS Test Center, Beldringevej 252, 5270 Odense
[43] Motion Capture Lab
[44] OptiTrack

8.2.1 Safety equipment

When flying inside the OptiTrack system, always make sure that the safety net is closed and no one is inside. If you have to enter the area while the UAV is operating, wear a helmet and safety glasses. Being struck by a UAV or its propellers can cause severe injuries.

Always Remember, never power up the UAV system with the propeller mounted, unless it is before a test flight.

8.3 Outdoor test environment

The UAV test area at HCA Airport will be used for outdoor test flights. Please remember that testing at the airport takes place in strict time slots. Therefore, when flying there you must be ready to test exactly when your time slot begins. In order to minimize overhead and possible delays, it is recommended that you use simulation (section 8.1) or the indoor test environment (section 8.2) to the full extent possible. In addition, keep in mind that anyone piloting the UAV under these tests must have a hobby drone certificate (dronetegn)[45]. It is free and it takes around 15-20 minutes to take the theory test consisting of 12 questions.

8.3.1 Drone Flight Accountable (DFA)

When testing at HCA Airport, you will have to be assisted by a Drone Flight Accountable (DFA) at the Test field. The DFA is responsible for communication with the tower in order to minimize risk of any accidents or incidents to happen. If you need a DFA, coordinate it with your supervisor or the personal at the SDU UAS Hangar.

8.3.2 Safety equipment

You can be seriously hurt if you are hit, eg. in the eye by an propeller, so when you are outside and testing at the HCA Airport, you must always wear a helmet and safety glasses. Furthermore, you must wear a yellow vest and have an ID card, either a permanent or a guest card, before entering Air side at HCA Airport. Helmets, safety glasses and yellow vests are available at the SDU UAS Hangar (figure 17). You can get a quest card at the information desk in the HCA Airport.

Always Remember, never power up the UAV system with the propeller mounted, unless it is before a test flight.

[45] Droneflyvning

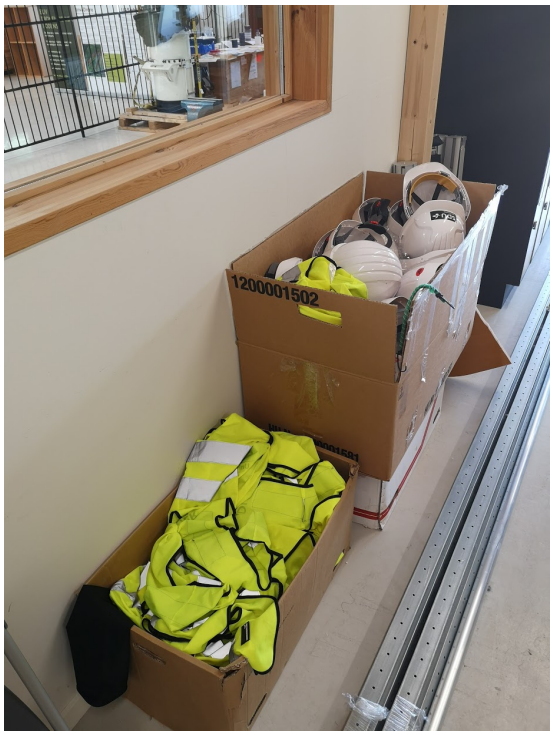


Figure 17: Safety equipment available at the SDU UAS Center.

8.4 Final acceptance test(s)

The final acceptance test(s) will be performed at HCA Airport at the end of the course, unless another location is more suited for the final acceptance test that you have defined.

9 Technical reporting (Best practice)

Concerning the technical reporting of the project, you are suggested to consult this Project Guide.

Moreover, please add as an appendix a table with a column for each team member and a row for each task (e.g., Table 1). The table should then be filled with the approximate contribution of each team member to the individual tasks.

Remember to archive:

- The risk assessment (see section 7.3).
- Any relevant appendices (formatted as PDF).
- Source code.

10 Acknowledgements

A big thank you to the employees at Rockwool and Lorenz Technology for providing the idea for the business cases and project ideas.

The Resuscitation Drones case described in Appendix B was provided from the previous course Unmanned Aerial System Design (RM-UASD) taught by Kjeld Jensen.

11 Copyright & license

[46] creativecommons.org/licenses/by/4.0/

References

[1] Justin J. Boutilier et al. "Optimizing a Drone Network to Deliver Automated External Defibrillators". In: *Circulation* (2017). ISSN: 0009-7322. DOI: 10.1161/CIRCULATIONAHA.116.026318. eprint: http://circ.ahajournals.org/content/early/2017/03/02/CIRCULATIONAHA.116.026318.full.pdf.

[2] A. Claesson, A. Bäckman, and M. Ringhl. "Time to delivery of an automated external defibrillator using a drone for simulated out-of-hospital cardiac arrests vs emergency medical services". In: *JAMA* 317.22 (2017), pp. 2332–2334. DOI: 10.1001/jama.2017.3957. eprint: /data/journals/jama/936301/jama_claesson_2017_ld_170012.pdf.

[3] Anders la Cour-Harbo. "Ground impact probability distribution for small unmanned aircraft in ballistic descent". In: *Reliability Engineering & System Safety* (2017).

[4] Anders la Cour-Harbo. "Mass threshold for 'harmless' drones". In: *Robotics and Autonomous Systems 00 1–13* (2015).

[5] Anders la Cour-Harbo. "Quantifying risk of ground impact fatalities of power line inspection BVLOS flight with small unmanned aircraft". In: *IEEE International Conference on Unmanned Aircraft Systems*. June 2017.

[6] Mathias Fleck. "Usability of Lightweight Defibrillators for UAV Delivery". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. Santa Clara, California, USA: ACM, 2016, pp. 3056–3061. ISBN: 978-1-4503-4082-3. DOI: 10.1145/2851581.2892288.

[7] D. Meltz and H. Guterman. "Verification of safety for autonomous unmanned ground vehicles". In: *2014 IEEE 28th Convention of Electrical Electronics Engineers in Israel (IEEEI)*. Dec. 2014, pp. 1–5. DOI: 10.1109/EEEI.2014.7005895.

[8] Monash. *Human injury model for small unmanned aircraft impacts*. Tech. rep. Civil Aviation Safety Authority and Monash University, 2013.

[9] Andrew V. Shelley. "A Model of Human Harm from a Falling Unmanned Aircraft: Implications for UAS Regulation". In: *International Journal of Aviation, Aeronautics and Aerospace* 3.3 (Apr. 2016).

[10] Jeremy Straub and Justin Huber. *Validating a UAV artificial intelligence control system using an autonomous test case generator*. 2013. DOI: 10.1117/12.2018591.

Acronyms

CC Companion Computer. 10, 12, 16, 46

COTS Commercial Off The Shelf. 18

DFA Drone Flight Accountable. 25

FCU Flight Control Unit. 5, 16

GCS Ground Control Station. 12, 13

GPS Global Positioning System. 8

OS Operating System. 16

QGC QGroundControl. 13–15, 23, 45

ROS Robot Operating System. 16, 41, 48

RPI3 Raspberry Pi 3 B+. 10

SITL Software-In-The-Loop. 24, 51

SORA Specific Operations Risk Assessment. 22, 24

UAS Unmanned Aircraft Systems. 2, 18, 22, 24, 36

UAV Unmanned Aerial Vehicle. 1, 2, 4, 7, 10, 12, 14–16, 18, 19, 21–25, 44, 46, 48–52

UI User Interface. 13, 18

Appendices

A Diagrams

CAN 1 & 2

Pin #	Name	Dir	Wire Color	Description
1	VCC_5V	out	red / gray	Supply to peripheral from AP
2	CAN_H	in/out	yellow / black	12V
3	CAN_L	in/out	green / black	12V
4	GND	-	black	GND connection

ADC

Pin #	Name	Dir	Wire Color	Description
1	VDD_5V_Periph	out		
2	Pressure sense in	in		
3	GND	out		GND

SPKT

Pin #	Name	Dir	Wire Color	Description
1	VDD_3v3_spektrum	out		Independent supply 3v3.
2	IO_USART1_RX	in		
3	GND	out		GND

POWER 1

Pin #	Name	Dir	Wire Color	Description
1	VDD 5V Brick	in	red / gray	Supply from Brick to AP
2	VDD 5V Brick	in	red / gray	Supply from Brick to AP
3	BATT_CURRENT_SENS_PROT			Battery current connector
4	BATT_VOLTAGE_SENS_PROT	in	black	Battery voltage connector
5	GND	-	black	GND connection
6	GND	-	black	GND connection

pixhawk 2
THE LATEST OPEN SOURCE AUTOPILOT

TELEM 1 & 2

Pin #	Name	Dir	Wire Color	Description
1	VCC_5V	out	red / gray	Supply to GPS from AP
2	MCU_TX	out	yellow / black	3.3V-5.0V TTL level, TX of AP
3	MCU_RX	in	green / black	3.3V-5.0V TTL level, RX of AP
4	MCU_CTS (TX)	out	gray / black	3.3V-5.0V TTL level or TX of AP
5	MCU_RTS (RX)	in	gray / black	3.3V-5.0V TTL level or RX of AP
6	GND	-	black	GND connection

GPS 1

Pin #	Name	Dir	Wire Color	Description
1	VCC_5V	in	red	Supply to GPS from AP
2	GPS_RX	in	black	3.3V-5.0V TTL level, TX of AP
3	GPS_TX	out	black	3.3V-5.0V TTL level, RX of AP
4	SCL	in	black	3.3V-5.0V I2C1
5	SDA	in/out	black	3.3V-5.0V I2C1
6	BUTTON	out	black	Signal shorted to GND on press
7	BUTTON_LED	out	black	LED Driver for Safety Button
8	GND	-	black	GND connection

GPS 2

Pin #	Name	Dir	Wire Color	Description
1	VCC_5V	out	red / gray	Supply to GPS from AP
2	MCU_TX	out	yellow / black	3.3V-5.0V TTL level, TX of AP
3	MCU_RX	in	green / black	3.3V-5.0V TTL level, RX of AP
4	SCL	out	gray / black	3.3V-5.0V I2C2
5	SDA	in	gray / black	3.3V-5.0V I2C2
6	GND	-	black	GND connection

USB

Pin #	Name	Dir	Wire Color	Description
1	VCC_5V	out	red / gray	Supply to GPS from AP
2	D_PLUS	in/out	green / black	3.3V
3	D_MINUS	in/out	red / black	3.3V
4	GND	-	black	GND connection
5	BUZZER	out	gray / black	VBAT (8.4 - 42V)
6	BE_LED	out	black	Boot / Error Led (FW updates)

I2C

Pin #	Name	Dir	Wire Color	Description
1	VCC_5V	out	red / gray	Supply to peripheral from AP
2	SCL	in/out	blue / black	SCL, 5V level, pull-up on AP
3	SDA	in/out	green / black	SDA, 5V level, pull-up on AP
4	GND	-	black	GND connection

POWER 2

Pin #	Name	Dir	Wire Color	Description
1	VDD 5V Brick	in	red / gray	Supply from Brick to AP
2	VDD 5V Brick	in	red / gray	Supply from Brick to AP
3	AUX_BATT_CURRENT_SENS			Aux Battery current connector
4	AUX_BATT_VOLTAGE_SENS	in	black	Aux Battery voltage connector
5	GND	-	black	GND connection
6	GND	-	black	GND connection

Figure 18: *Pixhawk 2.1 pinout.*
Source: ArduPilot discussion

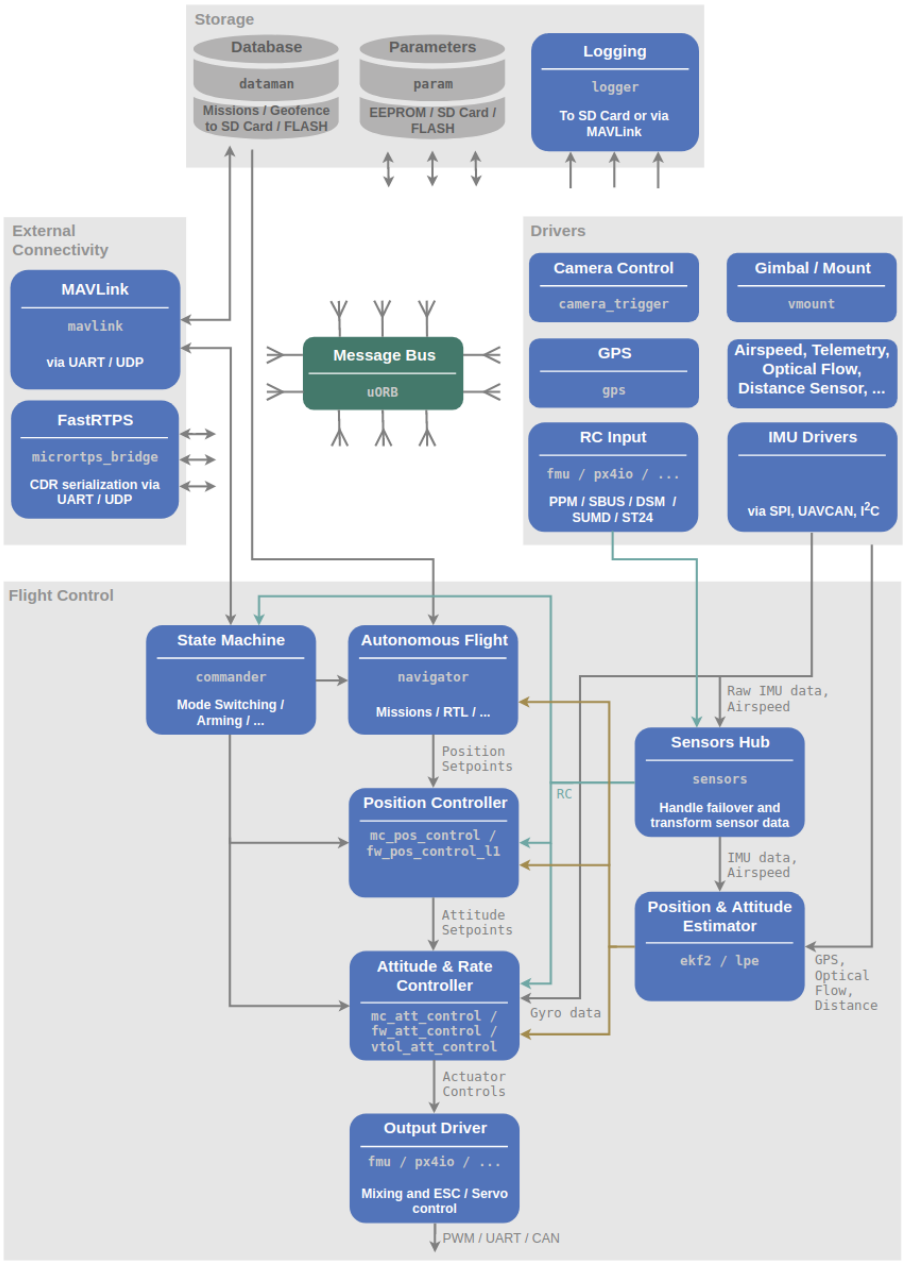


Figure 19: *Overview of the High-Level Software Architecture.*
Source: PX4 Dev: PX4 Architectural Overview

B Example: Resuscitation Drones (case)

Cardiac arrest is an unpredictable, life threatening condition that requires immediate medical action for the patient to survive. The probability of surviving a cardiac arrest outside a hospital is very low, in Denmark on average 11 persons suffer from an out of hospital cardiac arrest (OHCA) every day, and despite huge efforts aiming to increase the probability of survival, it is still only at 13% [47]. About two third of all OHCA's occur in private homes, and about one third in public places.

The Chain of Survival depicted in figure 20 summarizes the vital links needed for successful resuscitation. One of the key solutions to increase the probability of surviving an OHCA is reducing the time to defibrillation. According to the European Resuscitation Council (ERC) guidelines[48] each minute of delay to defibrillation reduces the probability of survival to discharge by 10–12%. When bystander cardiopulmonary resuscitation (CPR) is provided, the decline in survival is more gradual and averages 3–4% per minute delay to defibrillation.

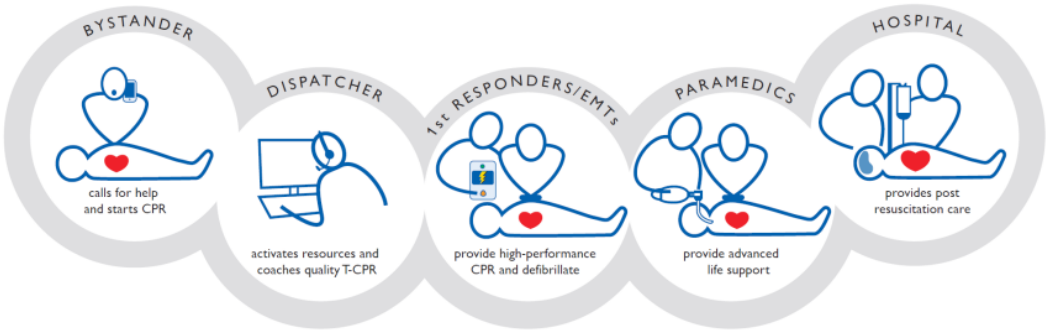


Figure 20: *Chain of survival, illustrates a series of actions which give the best chance of survival from an out of hospital cardiac arrest (OHCA). Souce: European Resuscitation Council (ERC).*

Early defibrillation requires quick access to an Automated External Defibrillator (AED). In Denmark and many other countries numerous AED's are installed at public places, schools, train stations, shopping centers, sports facilities, office complexes etc. Their locations are registered in public databases such as hjertestarter.dk[49] and associated phone apps aim to quickly identify the nearest location of an AED. Despite the high availability of nearby AED's, it takes minutes to first identify the location and then retrieve the AED, - minutes that lower the probability of survival.

One idea of reducing the time to defibrillation is to have a drone deliver the AED to the patient. This idea is not new, and the perhaps most well known concept demonstration was performed in 2014 by a Dutch student at TU Delft[50]. Since then the idea of drone-based AED delivery has often been presented in the media, at conferences etc. Some research has been conducted, examples are [2, 6, 1] and there are examples of product development as well[51].

The business model of using drones for AED delivery at golf courses is potentially very interesting and in addition many golf courses constitute an almost ideal environment considering obtaining permission for autonomous Beyond Visual Line Of Sight (BVLOS) flight.

In Denmark golf is a quite popular sport. About 152,500 members were registered by Dansk Golf Union (DGU) in 2016, and golf is thus only superseded in popularity by soccer. Figure 21 illustrates the age distribution among the members. 45% of the members are 60 years old and above.

[47] Rapport fra Dansk Hjertestopregister 2001-2014
[48] European Resuscitation Council guidelines
[49] hjertestarter.dk
[50] TU Delft Ambulance Drone
[51] www.flypulse.se

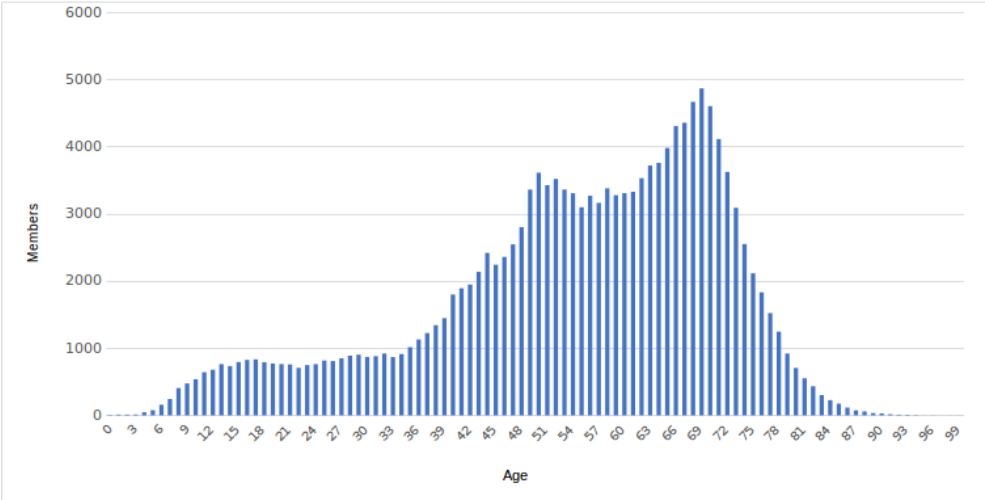


Figure 21: *Age distribution among members of Dansk Golf Union (DGU) in 2016. Souce: DGU*

There are 193 golf courses (August 2017) spread across Denmark. The golf courses are typically located in rural areas at the border of or outside cities. Many golf clubs have an AED located at the golf club house. According to a report from 2012 the average size of a Danish golf course is 68 Ha[52], depending on the location of the club house, shape of the golf course, traversible paths etc., the maximum distance to the AED would typically be from 750 m up to 2000 m (in other countries the distance may be significantly longer).

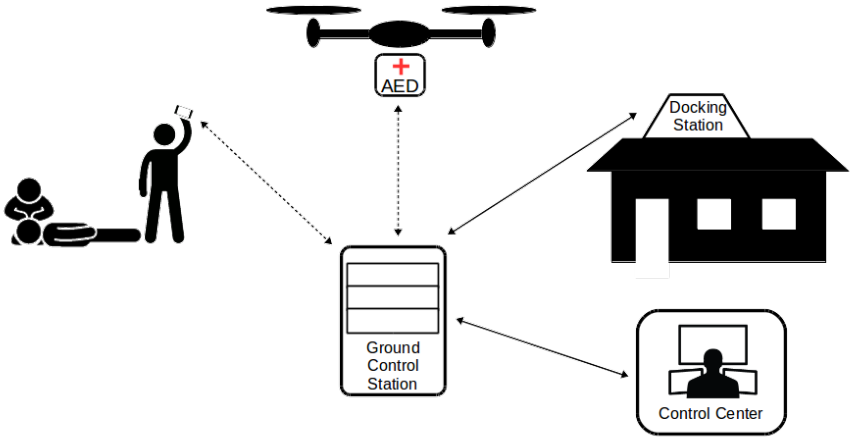


Figure 22: *A schematic overview of the UAS for AED delivery. The overall flow of the AED being delivered to the patient having a sudden cardiac arrest is depicted in figure 23. The dashed lines indicate wireless connections while the solid lines indicate wired connections.*

In case the AED is needed at the golf course, a bystander would have to fetch this, which would take about 5 minutes for a trained runner given a running distance of 500 m to the AED. In some situations it may be possible to save time by calling the golf club house and ask for delivery of the AED (this happened with a successful outcome in 2016[53]), however in comparison it would take a drone less than

[52] Golfsportens grønne regnskab 2012
[53] TV2: "Golftur udviklede sig dramatisk"

1 minute to complete the delivery at a distance of 500 m.

A schematic overview of the UAS is shown in figure 22 and figure 23 shows the overall (succesful) flow of an OHCA leading to the AED being delivered by a drone.

Workflow

Figure 23 shows the overall flow of an OHCA leading to the AED being delivered by a drone. The aim of this project is to build a UAS capable of supporting this flow including handling exceptions that may occur.

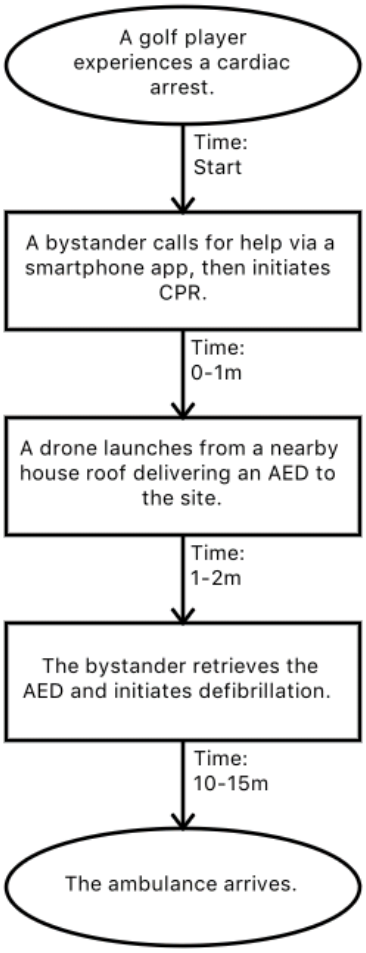


Figure 23: *Overall flow of an out of hospital cardiac arrest (OHCA) leading to the AED being delivered by a drone. When the bystander requests help, the drone is activated and at the same time information is sent to the emergency service control center about the event.*

Requirements

#	Requirement
1	The drone must be able to fly at wind speeds up to 10 m/s.
2	The drone must be able to fly a ground distance of 2500 m compensating for wind speed within specifications.
3	The drone must be able to make the delivery within 15 min.
4	The drone must be capable of autonomous take off from a docking station, autonomous flight to the accident site, and autonomous landing on the grass.
5	The docking station and any other infrastructure exposed to outdoor conditions must comply with Ingres Protection level IP54 while in standby mode (docking station lid closed).
6	The docking station must support the temperature requirements by the AED payload.

Table 3: *List of requirements for the AED delivery UAS.*

Design goals

#	Design goal
1	Rapid delivery of the AED to the patient is a primary goal. Any delaying factor such as flight permission validation, docking lid opening, takeoff, flight and landing must be conducted as quickly as possible. Where relevant parallel processing should be utilized.
2	The UAS must be designed towards the highest possible level of reliability and fault tolerance to increase the probability of a succesful AED delivery and to ease the process of obtaining BVLOS flight permission. Where possible redundancy should be included in the design (but not necessarily fully implemented).
3	A high level of usability of the UAS is key to ensuring that requesting, delivery and use of the AED is not delayed by human errors.

Table 4: *List of design goals for the AED delivery UAS. The numbers are for reference, the list is not prioritized.*

System design

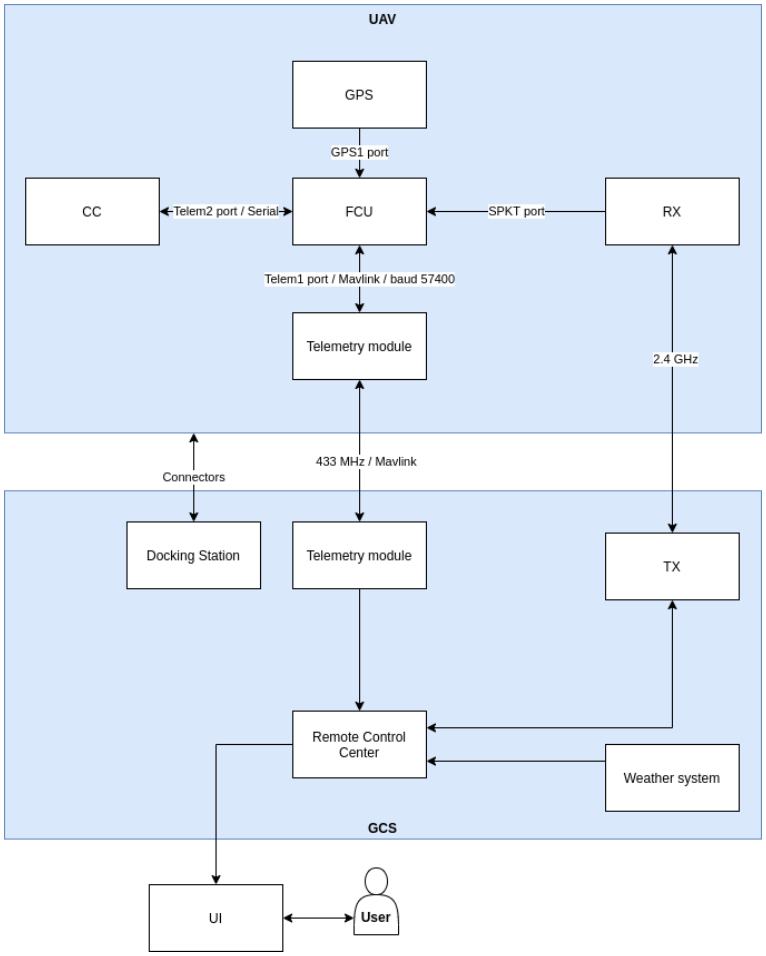


Figure 24: *Overall system design of the AED delivery UAS.*

User Interface

When a golf player experiences a cardiac arrest somewhere on the golf course, a bystander calls for help and then initiates cardiopulmonary resuscitation (CPR) (figure 23). Calling for help is activated via a smartphone based user interface, thereby activating the drone AED delivery and informing the emergency services control center. After alerting the user receives first aid instructions and receives continuous updates about the status of the AED delivery. In case of a failed delivery the user is informed about available options. The user interface is implemented as an app.

Final acceptance test

The final acceptance test will be performed at HCA Airport test field, where we will pretend that it is a golf course.

The purpose of the acceptance test is to validate and demonstrate the result of the UAS for AED delivery with respect to the requirements and design goals defined. During the test which will last several hours, a number of cardiac arrests at different locations within the golf course area will be simulated.

The results of these simulations will be documented and evaluated in order to show that the system and all subsystems worked as intended.

C Configuration of mRo radio settings

C.1 Update to latest firmware version

1. Open QGroundControl (QGC) and navigate to the firmware section, as shown in figure 25.

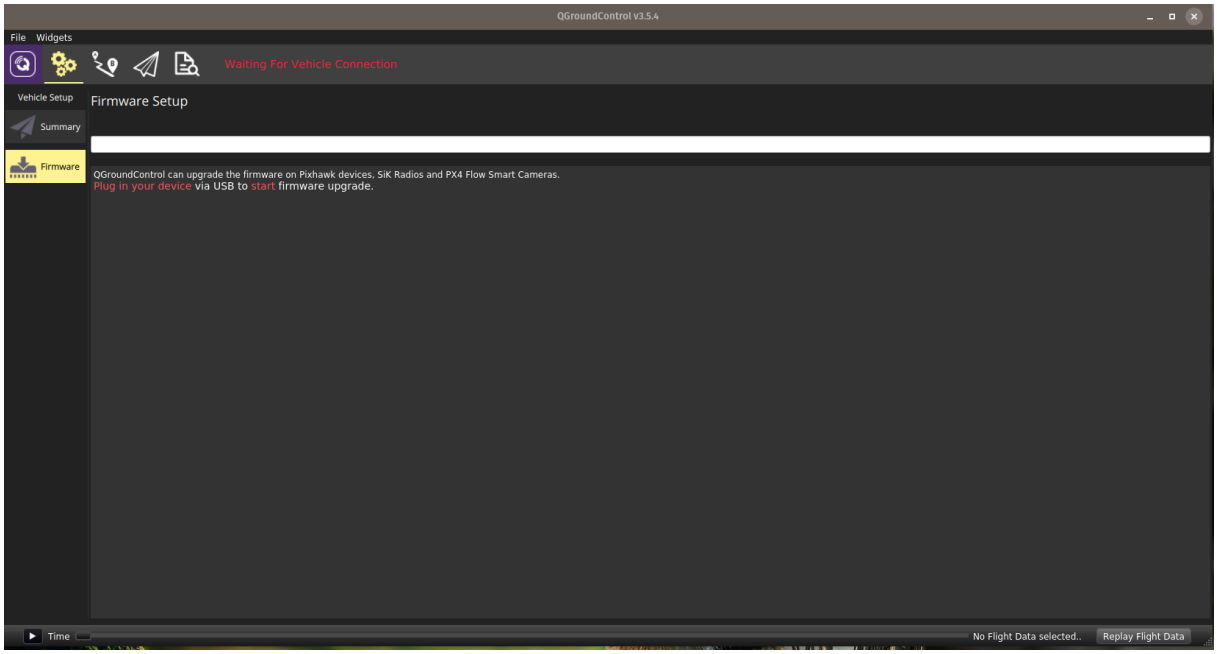


Figure 25: QGC firmware section.

2. Connect the mRo module to the computer by USB and wait for QGC to report the flashing is complete.

Remember to flash both modules to ensure they run the same version.

C.2 Change settings on the mRo Modules

OBS: Receiving data from the other radio while trying to get into configuration mode, might not work. If that happens ensure the other paired radio is offline and try again.

A video showing an example of changing the parameters using this guide is available here.

1. Install screen on your computer with: `sudo apt install screen`
2. Connect the mRo module to the usb port and find what port it uses. Typical it would be `/dev/ttyUSB0`. Port can be found by running `dmesg | grep tty`
3. Connect to the mRo module with screen: `screen /dev/ttyUSB0 57600 8N1` (This tries to open serial device USB0 with 57600 baudrate and 8N1 settings for stopbit etc.
4. Click on your + key three times, so you send three + characters. (**OBS: It doesn't indicate when you send them, only after you have sent all three!**). Then wait for it to respond `OK`. This signals you entered configuration mode. You might have to wait 1 sec from it showing `OK` to you can send commands.
5. Then use the commands shown in table 5 to change settings. The settings and their number are shown in code 1. Settings won't be saved until you write them to the EEPROM with the `AT&W` command.

   For example to change the `NETID` to 250 you would write: `ATS3=250`. Then finish by `AT&W` to save the changes.

Command	Description
`ATI`	Show radio version
`ATI2`	Show board type
`ATI3`	Show board frequency
`ATI4`	Show board version
`ATI5`	Show all user settable EEPROM parameters
`ATO`	Exit AT command mode
`ATSn?`	Display radio parameter number 'n'
`ATSn=X`	Set radio parameter number 'n' to 'X'
`ATZ`	Reboot the radio
`AT&W`	Write current parameters to EEPROM
`AT&F`	Reset all parameters to factory by default

Table 5: Radio commands

```
S0:FORMAT=26
S1:SERIAL_SPEED=57
S2:AIR_SPEED=64
S3:NETID=25
S4:TXPOWER=20
S5:ECC=0
S6:MAVLINK=1
S7:OPPRESEND=0
S8:MIN_FREQ=433050
S9:MAX_FREQ=434790
S10:NUM_CHANNELS=10
S11:DUTY_CYCLE=100
S12:LBT_RSSI=0
S13:MANCHESTER=0
S14:RTSCTS=0
S15:MAX_WINDOW=131
```

Listing 1: The default settings and their numbers

6. `ATI5` shows the current settings, do note that if frequencies are showing 0, you need to reboot the module with `ATZ` and then resend the three + characters. Then the frequencies should show the correct values.

7. When required settings has been changed, then remember to save with `AT&W` command and reboot the module with the `ATZ` command. Do the same to the other radio. Afterwards they should both connect to each other, which is shown as a solid green led on both modules when both are turned on and are in range.

C.3 Screen cheatsheet

Description	Command
Start a new session with session name	`screen -S <session_name>`
List running sessions / screens	`screen -ls`
Attach to a running session	`screen -x`
Attach to a running session with name	`screen -r <session_name>`
Detach a running session	`screen -d <session_name>`
Detach	`Ctrl-a d`
Detach and logout (quick exit)	`Ctrl-a D D`
Exit screen	`Ctrl-a :quit`
Force-exit screen	`Ctrl-a C-\(not recommended)`

Table 6: Screen cheatsheet

C.4 Telemetry setup

Common settings

```
S0:FORMAT=26
S1:SERIAL_SPEED=57
S2:AIR_SPEED=64
S3:NETID=See table below
S4:TXPOWER=20
S5:ECC=0
S6:MAVLINK=1
S7:OPPRESEND=0
S8:MIN_FREQ=See table below
S9:MAX_FREQ=See table below
S10:NUM_CHANNELS=10
S11:DUTY_CYCLE=100
S12:LBT_RSSI=0
S13:MANCHESTER=0
S14:RTSCTS=0
S15:MAX_WINDOW=33
```

Listing 2: The default settings for the provided telemetry modules.

Individual settings

Blue

```
S3:NETID=40
S8:MIN_FREQ=433050
S9:MAX_FREQ=433340
```

Listing 3: The settings for the **Blue** modules.

Brown

```
S3:NETID=60
S8:MIN_FREQ=433340
S9:MAX_FREQ=433630
```

Listing 4: The settings for the **Brown** modules.

Green

```
S3:NETID=80
S8:MIN_FREQ=433630
S9:MAX_FREQ=433920
```

Listing 5: The settings for the **Green** modules.

Grey

```
S3:NETID=100
S8:MIN_FREQ=433920
S9:MAX_FREQ=434210
```

Listing 6: The settings for the **Grey** modules.

Red

```
S3:NETID=120
S8:MIN_FREQ=434210
S9:MAX_FREQ=434500
```

Listing 7: The settings for the **Red** modules.

Yellow

```
S3:NETID=140
S8:MIN_FREQ=434500
S9:MAX_FREQ=434790
```

Listing 8: The settings for the **Yellow** modules.

D Get the PX4 SITL simulation up and running

First, ensure that you have ROS installed. The guide is based on Ubuntu 18.04, so to install ROS Melodic[54], follow the guide from the official ROS webpage: http://wiki.ros.org/melodic/Installation/Ubuntu

It is recommended to install the `ros-melodic-desktop-full` package.

Next, install dependencies for PX4:

```
sudo apt install astyle build-essential ccache clang clang-tidy cmake
    cppcheck doxygen file g++ gcc gdb git lcov make ninja-build python3
    python3-dev python3-pip python3-setuptools python3-wheel rsync
    shellcheck unzip xsltproc zip libeigen3-dev libopencv-dev libroscpp-dev
    protobuf-compiler python-pip python3-pip ninja-build gstreamer1.0-
    plugins-bad gstreamer1.0-plugins-base gstreamer1.0-plugins-good
    gstreamer1.0-plugins-ugly libgstreamer-plugins-base1.0-dev
    libgstrtspserver-1.0-dev xvfb
```

```
pip install --user argparse cerberus empy jinja2 numpy packaging pandas
    psutil pygments pyros-genmsg pyserial pyulog pyyaml setuptools six toml
    wheel rosdep
```

```
pip3 install --user --upgrade empy jinja2 numpy packaging pyros-genmsg toml
     pyyaml pymavlink
```

For convenience when working with ROS, also install the Python-based catkin tools:

```
sudo apt install python-catkin-tools
```

Then clone the PX4 firmware from github

```
cd ~
git clone https://github.com/PX4/Firmware.git
cd Firmware
git submodule update --init --recursive
```

Build the PX4 SITL firmware and run the Gazebo environment

```
cd ~/Firmware
DONT_RUN=1 make px4_sitl_default gazebo
```

Running the PX4 SITL environment

Run the PX4 SITL environment

```
cd ~/Firmware
make px4_sitl_default gazebo
```

And wait until the firmware has been built and the Gazebo environment has opened.

Become familiar with the PX4 SITL environment

The make command drops you into the PX4 console (press ENTER if you don't see the prompt). Run the following commands in the terminal `pxh>`

```
ver all
commander takeoff
commander land
commander arm
help
```

[54] ROS Melodic Morenia

Launching the PX4 SITL with ROS wrapper

First, source the Firmware

```
source /home/$USER/Firmware/Tools/setup_gazebo.bash /home/$USER/Firmware /
    home/$USER/Firmware/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/$USER/Firmware
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/$USER/Firmware/Tools/
    sitl_gazebo
```

(optional): Add the commands to your `.bashrc`

and then, run the simulation wrapped in ROS using

```
roslaunch px4 posix_sitl.launch
```

Remember to source and build the ROS workspaces that you would like to access in the right order.

Get started with QGroundControl and PX4 SITL

Download and run QGC

If you have not installed QGC yet, follow these steps. Download the QGC AppImage:

http://qgroundcontrol.com/downloads/

and run it:

```
cd <download-path>
chmod +x ./QGroundControl.AppImage
./QGroundControl.AppImage
```

Start experimenting with the different views in the main toolbar.

- Run the PX4 SITL environment
- **Start QGC**
- **Connect to the Vehicle:** QGC should connect to the PX4 SITL automatically
  - If not, go to **App>General** and make sure that **UDP** is checked under **AutoConnect to the following devices**. (Optional:) You can create/add a manual connection to the UDP under **App>Comm Links**.
- **Arm and Takeoff:** Using the Fly view, try to click the on the map and command the vehicle to fly to that location
- **Plan and fly a mission:** Create autonomous mission using the Plan view. Set the Flight mode to Mission and monitor your vehicle while flying the route.
- Finding and setting/changing parameters using the Setup view.

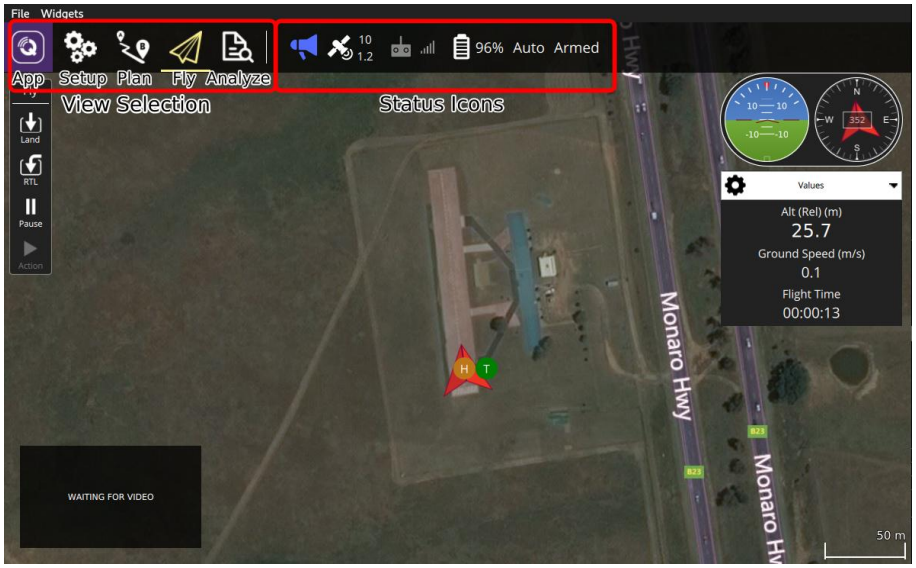


Figure 26: QGroundControl user interface overview. Source: link

Setting the Global coordinate

Set the Global coordinates by setting the following parameters

```
export PX4_HOME_LAT=55.4719762
export PX4_HOME_LON=10.3248095
export PX4_HOME_ALT=7.4000000
```

and build the PX4 SITL again

```
cd ~/Firmware
make px4_sitl_default gazebo
```

Start QGC, and verify that the start location of the vehicle has changed.

Running the PX4 SITL in HEADLESS mode

Building and running the PX4 SITL environment without the Gazebo environment.

```
cd ~/Firmware
HEADLESS=1 make px4_sitl_default gazebo
```

You can manually open the Gazebo environment using gzclient:

```
gzclient
```

E Getting started with the Indoor environment (OptiTrack system)

This appendix will guide you how to use the Motion Capture System, called Optitrack, at SDU UAS Hangar. The appendix will guide you how to

- Calibrate the OptiTrack system
- Configure your UAV so that it accepts external position information
- Stream position and orientation data of an UAV and feed it to your flight controller using MAVROS, and thereby be able to fly your UAV indoor with a high precision.

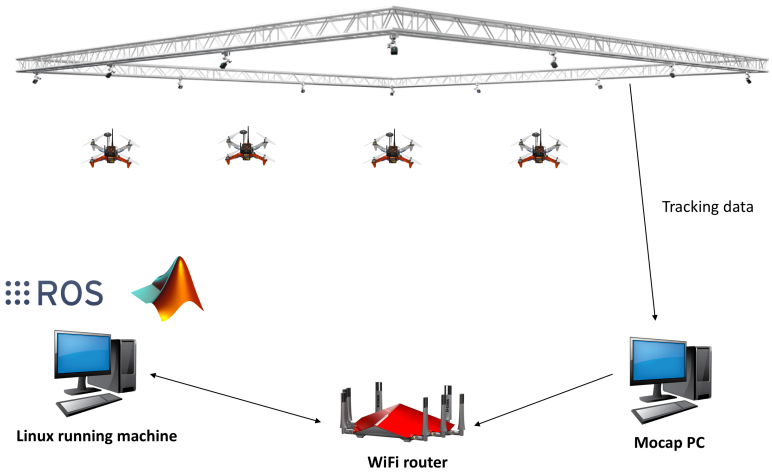


Figure 27: The overall system overview. Source: link

The overall system overview can be seen on figure 27, and it has the following elements:

- The OptiTrack system, consisting of
  - A computer running the OptiTrack software, called Motive[55]
  - 16x Prime cameras
- Wifi router
- UAVs with reflective markers on

Relevant links:

- PX4 Dev Guide: Using Vision or Motion Capture Systems for Position Estimation
- OptiTrack Wiki: Calibration
- RISC-Docs: Indoor flying

E.1 OptiTrack system

OptiTrack motion capture system works as follows:

- The Prime cameras sends out pulsed infrared light using the attached infrared LEDs
- The light will be reflected by the markers on the UAV and detected by the Prime cameras.
- Knowing the position of those markers in perspective of three or more of the Prime cameras, the actual 3D position of the markers can be calculated using triangulation. The position is provided in meters and orientation is in quaternions.

[55] OptiTrack - Software

E.2 Calibration process

The calibration involves three main steps

- Masking
- Wanding
- Ground Plane and Origin

E.2.1 Masking

E.3 Companion Computer

E.4 Wifi setup

Connect the

E.5 EKF2 Configuration

The following parameters must be set to use external position information from the OptiTrack system with the EKF2, eg. using QGC:

Parameter	Value	Description
SYS_MC_EST_GROUP	**EKF2**	Multicopter estimator group (default: EKF2)
EKF2_AID_MASK	**24**	Vision position and vision yaw fusion enabled
EKF2_HGT_MODE	**Vision**	Primary source of height data used by the EKF
EKF2_EV_DELAY	**50 ms**	Vision Position Estimator delay relative to IMU measurements

Table 7: EKF2 parameters for using external position information.

For more detailed information, check the EKF2 tuning guide.

```
gzclient
```

F Getting started with the CC

Connecting to the CC (Remotely): You can either connect to CC using a Ethernet connecting or setting up an access point, by configuring the `wlan0` settings. Open the `netplan` config file:

```
sudoedit /etc/netplan/50-cloud-init.yaml
```

and enter the `SSID-NAME` and `PASSWORD` for the access point, and apply the changes:

```
sudo netplan apply
```

After applying the changes, the CC should connect to the access point, and you can `ssh` into the CC.:

```
ssh ubuntu@<ip-of-the-cc>
```

Default password is `uas4ever`, but it is recommended that you changes the password after the first login and generates a new ssh key.

Testing the connecting between CC and Pixhawk

Connect MAVROS to the UAV platform using

```
roslaunch mavros px4.launch fcu_url:="/dev/<serial-device>:921600"
```

Verify that MAVROS is connected to the vehicle, eg. by echoing the IMU data

```
rostopic echo /mavros/imu/data
```

Udev rules for CC

To ensure that the programs on the CC always open the right ports udev rules are added for PX4. The rules are recognized by their vendorID and productID, and you can add more by editing the `/etc/udev/rules.d/10-local.rules`.

Remember to trigger the rules after editing the file by

```
sudo udevadm trigger
```

A rule for the Pixhawk have already been made, so you can connect using

```
roslaunch mavros px4.launch fcu_url:="/dev/PX4:921600"
```

Backup your CC image

It's always a good idea tp keep a copy of the SD card, so you can restore the system if something happens, eg. the SD card becomes corrupt. You can use `dd` to make a backup

```
sudo dd bs=4M if=/dev/sdb of=sdu_uas_rpi_`date +%d%m%y`.img
```

and compress it afterwards

```
xz sdu_uas_rpi_`date +%d%m%y`.img
```

This process will take some time, but it will save you a lot of memory.

G Expert-in-Teams Playground (Working example)

This appendix is a guideline to get the Expert-in-Teams Playground working example up and running.

First, ensure that your PX4 SITL Simulation environment is working (see Appendix D).

If you haven't installed MAVROS already, then

```
sudo apt install ros-melodic-mavros ros-melodic-mavros-extras -y
```

and, install GeographicLib datasets by running the `install_geographiclib_datasets.sh` script:

```
wget https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scripts
    /install_geographiclib_datasets.sh
chmod 755 install_geographiclib_datasets.sh
sudo ./install_geographiclib_datasets.sh
```

Launching the PX4 SITL with ROS wrapper and MAVROS

First, source the Firmware

```
source /home/$USER/Firmware/Tools/setup_gazebo.bash /home/$USER/Firmware /
    home/$USER/Firmware/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/$USER/Firmware
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/$USER/Firmware/Tools/
    sitl_gazebo
```

and then, run the simulation wrapped in ROS using

```
roslaunch px4 posix_sitl.launch
```

Open a second terminal, and connect MAVROS to the simulated vehicle (localhost) using

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

Verify that MAVROS is connected to the vehicle, eg. by echoing the IMU data

```
rostopic echo /mavros/imu/data
```

If you see that the IMU data is being published, MAVROS has been successfully connected to the simulated vehicle.

G.1 Get started with the EiT Playground

If you don't already have a ROS workspace you can work in, create one

```
source /opt/ros/melodic/setup.bash
mkdir -p ~/eit_ws/src
cd ~/eit_ws/src/
catkin_init_workspace
cd ~/eit_ws/
catkin build
```

Download the `eit_playground` package here, unzip it into your `~/eit_ws/src/` folder and build and source the workspace again. After building the workspace, remember to source the Firmware folder (mentioned earlier). Alternative, source the `setup_gazebo.bash` file in the `eit_playground` folder:

```
source ~/eit_ws/src/eit_playground/setup_gazebo.bash
```

Setup shortcuts (optional)

Add the following line to your line to your `.bashrc` file:

```
alias seit-ws="source ~/eit_ws/devel/setup.bash && source ~/eit_ws/src/
    eit_playground/setup_gazebo.bash"
alias peit-ws="cd ~/eit_ws/ && seit-ws"
alias beit-ws="peit-ws && catkin build"
```

Description of the `alias`:

- `s<project-name>`: Source the catkin workspace of the project and the Firmware
- `p<project-name>`: Source both projects and go to the catkin workspace of the project
- `b<project-name>`: Build, Source both projects and go to the catkin workspace of the project

Remember to build the workspace manually the first time, or else you will get a `../devel/setup.bash: No such file or directory` error.

G.1.1 Testing the `OFFBOARD` mode

First, build and source the `eit_playground` package, eg. using the defined alias

```
beit-ws
```

Then, start the PX4 SITL simulation environment

```
roslaunch px4 posix_sitl.launch
```

and the `MAVROS` node

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

When Gazebo is ready, run the `offb_node` node

```
rosrun eit_playground offb_node
```

This node will switch the UAVs flight mode to `OFFBOARD`, arm the UAV and make it position itself 2 meters above the ground and let it hover for 30 seconds, whereafter the UAV is commanded to make a landing. The original `OFFBOARD` example can be found here.

Explanation

Include all of the custom messages required to operate services and topics from the needed ROS packages

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
```

Create a callback that listens for the current state of the autopilot and keeps updating the `currrent_state` .

```
mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
```

Instantiate the node and the publishers, subcribers and services needed for the system to run. Figure 28 shows the `rqt_graph` of the example, where the communication between the nodes are visualized.

```
ros::init(argc, argv, "offb_node");
ros::NodeHandle nh;

ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/state"
    , 10, state_cb);
ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>("
    mavros/setpoint_position/local", 10);
ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::
    CommandBool>("mavros/cmd/arming");
ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode
    >("mavros/set_mode");
```

Set the publishing rate. The `OFFBOARD` mode is only accepted, if it receives `OFFBOARD` commands less than 500 ms apart, so the publishing rate must be faster than 2 Hz.

```
//the setpoint publishing rate MUST be faster than 2Hz
ros::Rate rate(20.0);
```

Check the current stat and wait until the connection to the UAV has been established

```
while(ros::ok() && !current_state.connected){
    ros::spinOnce();
    rate.sleep();
}
```

Create a local pose, and publish them. The mode switch to `OFFBOARD` mode will be rejected, if there is not a stream of setpoints.

```
geometry_msgs::PoseStamped pose;
pose.pose.position.x = 0;
pose.pose.position.y = 0;
pose.pose.position.z = 2;

//send a few setpoints before starting
for(int i = 100; ros::ok() && i > 0; --i){
    local_pos_pub.publish(pose);
    ros::spinOnce();
    rate.sleep();
}
```

Create a custom mode for both `OFFBOARD` and `AUTO.LAND`

```
// OFFBOARD mode
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";

// AUTO.LAND mode
mavros_msgs::SetMode land_set_mode;
land_set_mode.request.custom_mode = "AUTO.LAND";
```

Create an arm command

```
// ARM command
mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;
```

Start running the node in a while statement as long there is communication with the `roscore`. In every iteration, the created local pose is being published in the rate specified further up.

```
while(ros::ok()){
```

```
    // .
    // .
    // .

    local_pos_pub.publish(pose);

    ros::spinOnce();
    rate.sleep();
}
```

Check arming and `OFFBOARD` flags, and attempt to switch to `OFFBOARD` mode, after which UAV is armed and making a takeoff to the defined setpoint. If a request is rejected, try again after 5 seconds.

```
if( current_state.mode != "OFFBOARD" &&
    (ros::Time::now() - last_request > ros::Duration(5.0))){
    ROS_INFO("Trying to enable Offboard...");
    if( set_mode_client.call(offb_set_mode) &&
        offb_set_mode.response.mode_sent){
        ROS_INFO(">> Offboard enabled");
    }
    last_request = ros::Time::now();
} else {
    if( !current_state.armed &&
        (ros::Time::now() - last_request > ros::Duration(5.0))){
        ROS_INFO("Trying to arm...");
        if( arming_client.call(arm_cmd) &&
            arm_cmd.response.success){
            ROS_INFO(">> Vehicle armed");
        }
        last_request = ros::Time::now();
    }
}
```

Check if more than 30 seconds has passed since we entered the while statement. If so, switch to `AUTO.LAND` mode, whereafter the UAV will make a landing and afterward exit the while statement.

```
if( current_state.mode != "LAND" &&
    (ros::Time::now() - ts_start > ros::Duration(30.0))){
    ROS_INFO("Trying to enable AUTO.LAND...");
    if( set_mode_client.call(land_set_mode) &&
        land_set_mode.response.mode_sent){
        ROS_INFO("AUTO.LAND enabled");
        break;
    }
}
```

ROS Graph

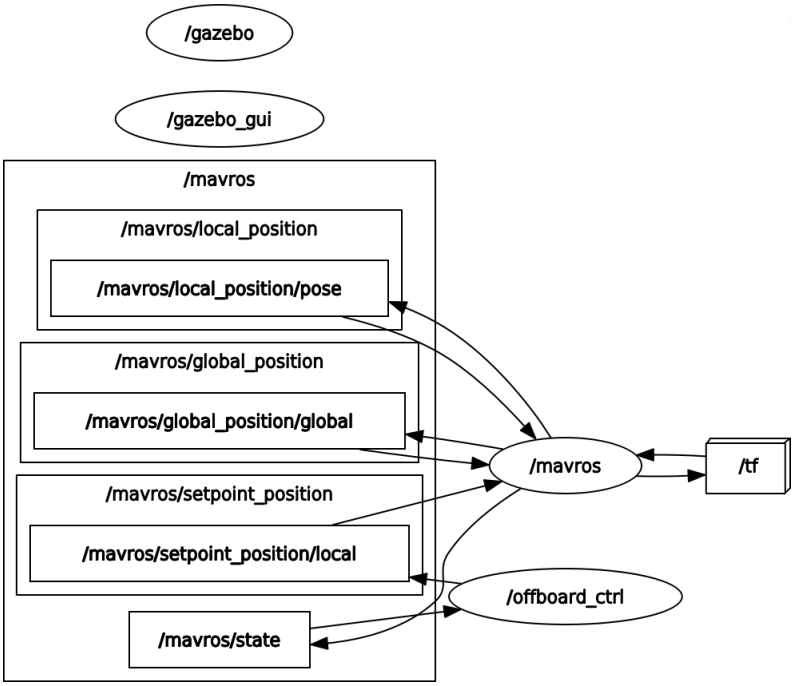


Figure 28: RQT Graph of testing the `offb_posctl` example

Vehicle selection

The default vehicle model in the PX4 SITL environment is the Iris drone (figure 29a), but other vehicle-model are also supported (eg. `solo`, `typhoon_h480`, `iris_opt_flow`). You can specify the vehicle model that you would like to spawn using the `vehicle` parameter, eg. if you would like to test with the 3DR Solo (figure 29b)

```
roslaunch px4 posix_sitl.launch vehicle:=solo
```

Setup the SDU Drone (custom UAV)

Adding a custom UAV to the PX4 SITL simulation environment requires the following things

- Create a Gazebo model of the UAV (`model.config` and `<custom-uav-name>.sdf`)
- Create an airframe file under `~/Firmware/ROMFS/px4fmu_common/init.d-posix/`

The Gazebo model and airframe files for the SDU Drone (figure 29c is available in the `eit_playground` package.

To get started with using the SDU Drone model with the SITL environment, you will first have to symlink the airframe files in the `init.d-posix` folder with the PX4 Firmware folder

```
ln -s /home/$USER/eit_ws/src/eit_playground/init.d-posix/* /home/$USER/
    Firmware/ROMFS/px4fmu_common/init.d-posix/
```

and the same with the `mixers` folder

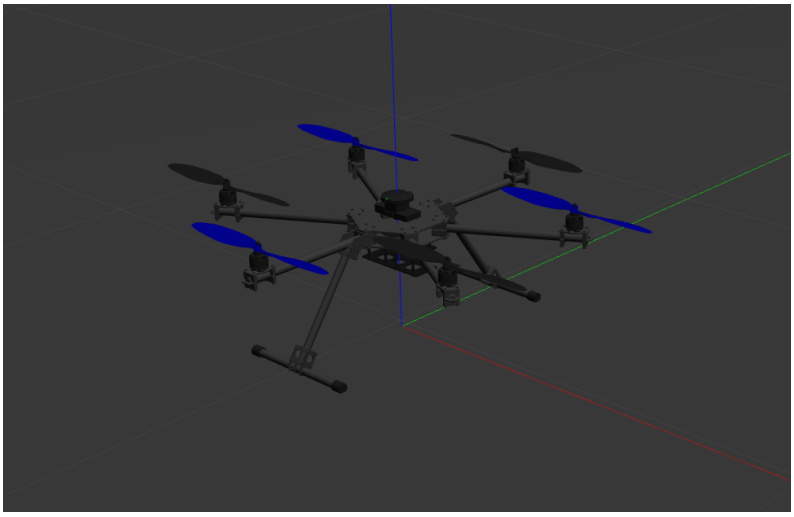
```
ln -s /home/$USER/eit_ws/src/eit_playground/mixers/* /home/$USER/Firmware/
    ROMFS/px4fmu_common/mixers/
```

After symlinking the airframe and mixers files, build and source the `eit_playground` package and start the simulation using the following command

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone
```

(a) Iris

(b) 3DR Solo



(c) SDU Drone

Figure 29: Different vehicle modules

Sensors

Gazebo supports several plugin types, which is a strong tool during the prototype development. You can find a list of the different sensors here, and you are able to access the sensor data via ROS topics.

The `eit_playground` package includes implementations of the most common sensors. Each sensor implementation is defined as a custom UAV, and can be selected using the `vehicle` parameter.

SDU Drone with Mono Camera

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone_mono_cam
```

Relevant ROS topic(s)

- `/mono_cam/image_raw`

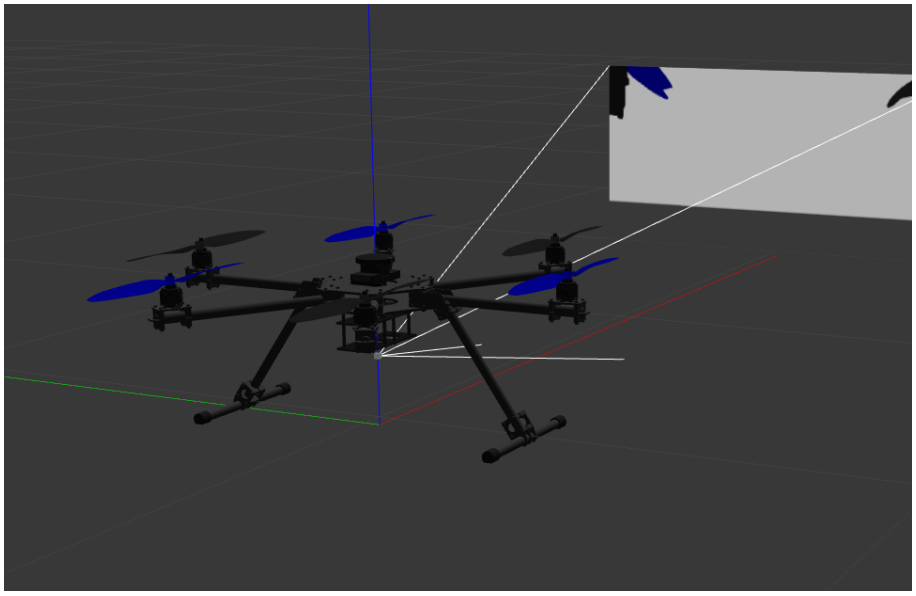


Figure 30: SDU Drone with Mono Camera

SDU Drone with Stereo Camera

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone_stereo_cam
```

Relevant ROS topic(s)

- `/stereo/left/image_raw`
- `/stereo/right/image_raw`

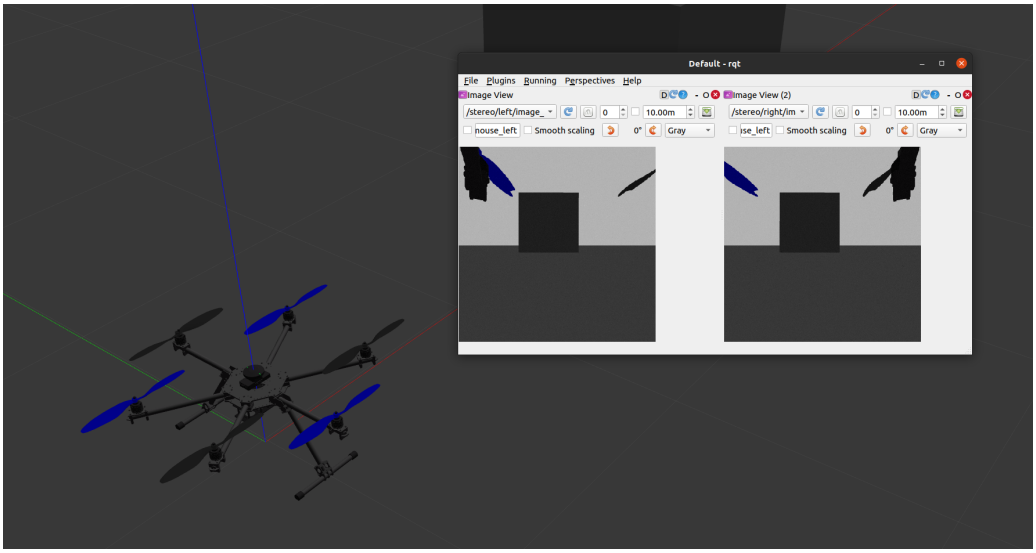


Figure 31: SDU Drone with Stereo Camera

SDU Drone with Depth Camera

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone_depth_cam
```

Relevant ROS topic(s)

- `/camera/rgb/image_raw`

- `/camera/depth/points`
- `/camera/depth/image_raw`

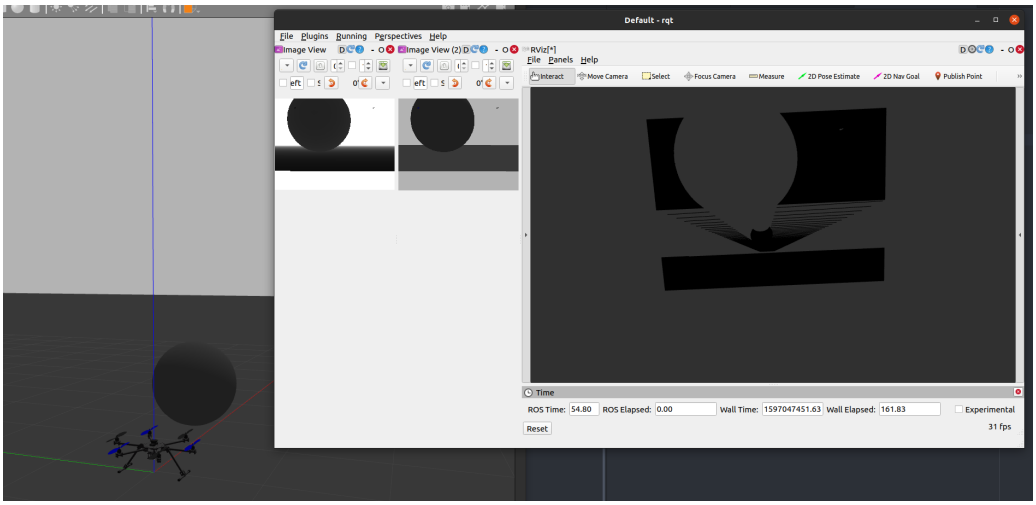


Figure 32: SDU Drone with Depth Camera

SDU Drone with 2D Lidar

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone_lidar
```

Relevant ROS topic(s)

- `/scan`

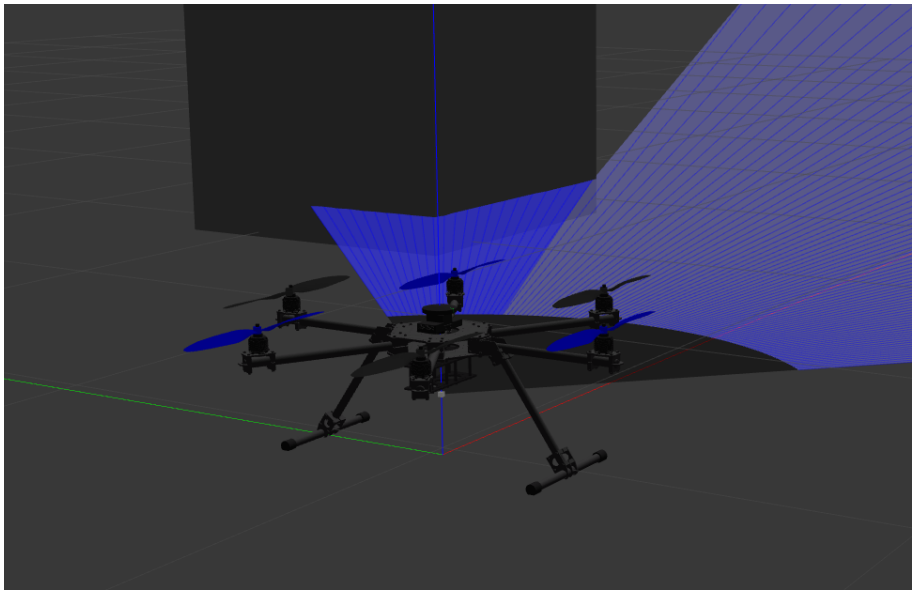


Figure 33: SDU Drone with 2D Lidar

SDU Drone with Sonar

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone_sonar
```

Relevant ROS topic(s)

- `/sonar`

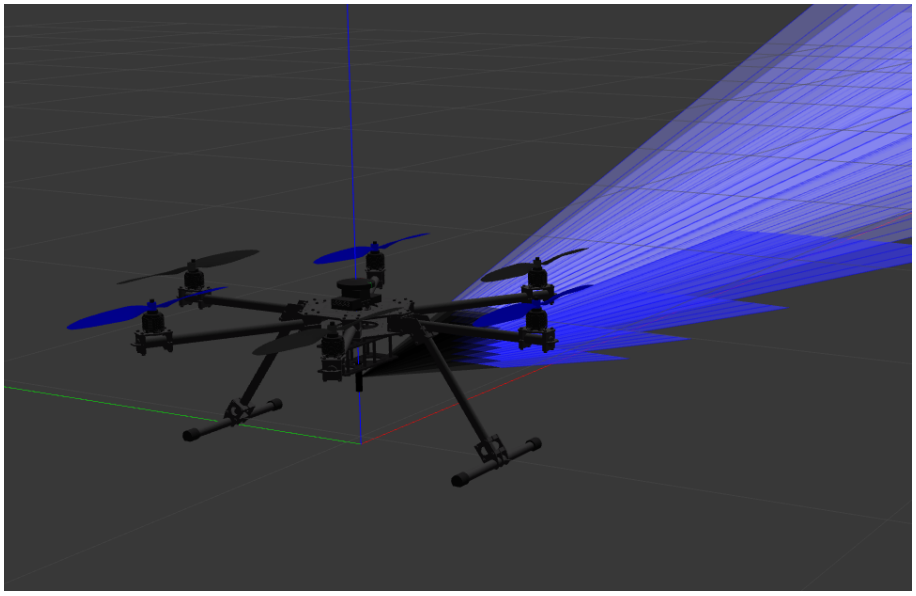


Figure 34: SDU Drone with Sonar

Environments

The `eit_playground` package includes a few pre-built Gazebo worlds. You can select one of the pre-built world using the `env` parameter.

OptiTrack environment

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone env:=optitrack
```

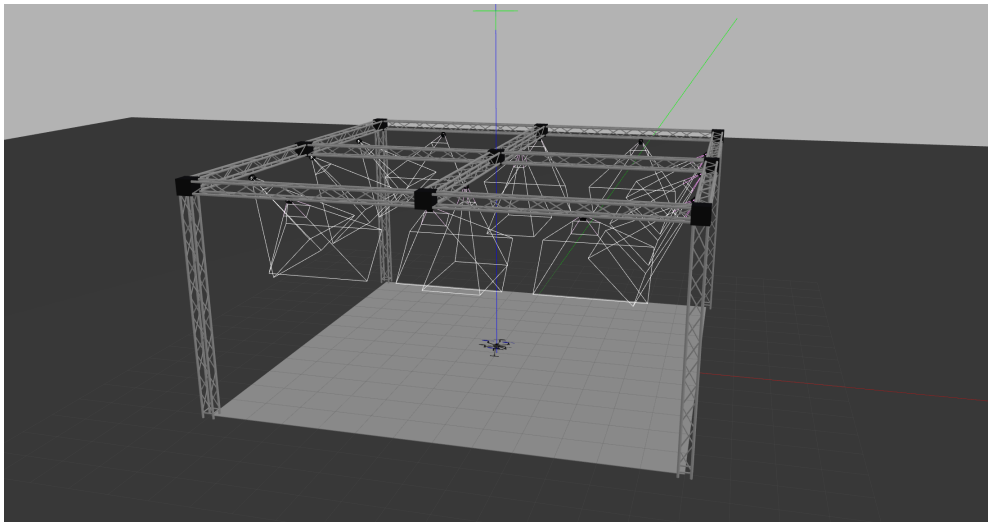
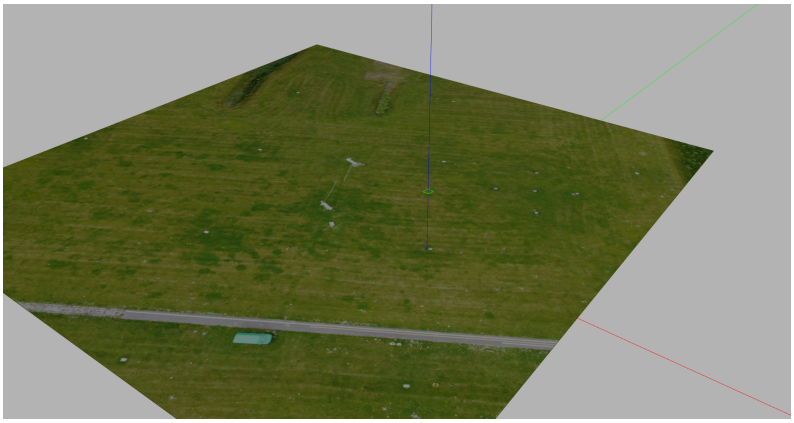


Figure 35: *The Optitrack simulation environment.*

SDU UAS Test Field

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone env:=hca_airport
```



(a) The HCA Airport simulation environment.

(b) The QGC veiw of the simulation.

Figure 36: Comparison of the HCA Airport simulation environment and the QGC view.

Warehouse

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone env:=warehouse
```

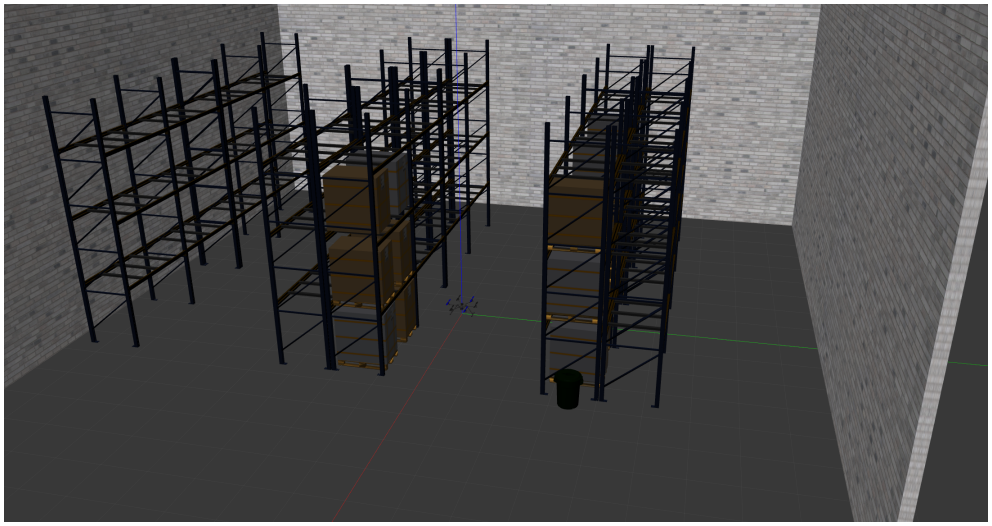


Figure 37: *The Warehouse simulation environment.*

Ocean

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone env:=ocean
```

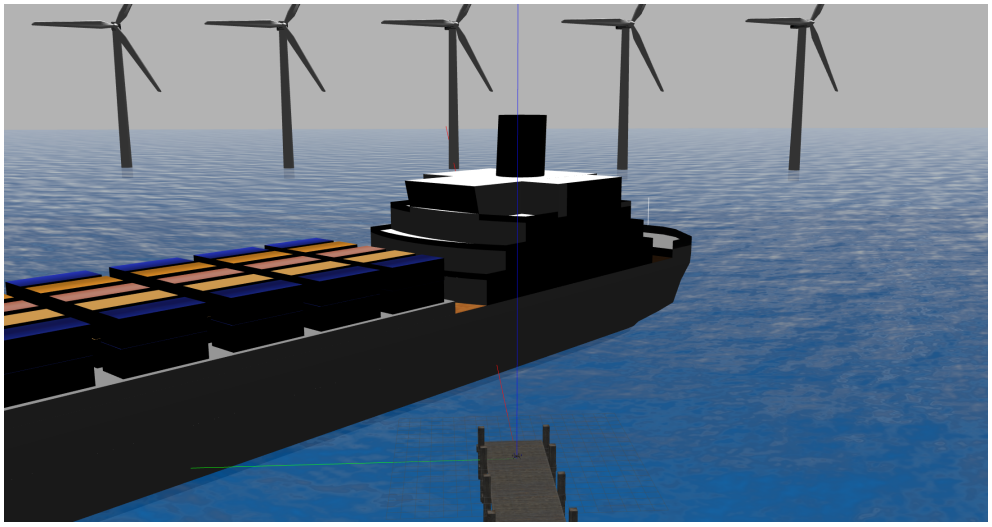
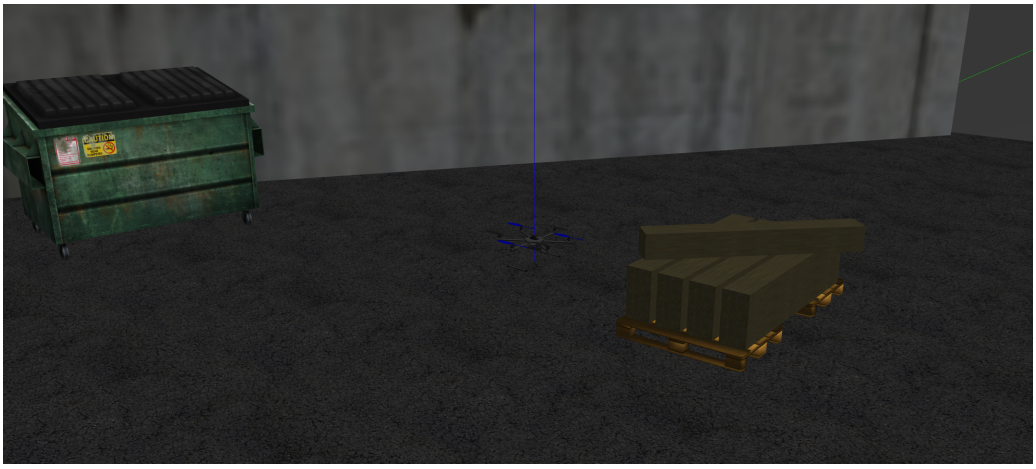


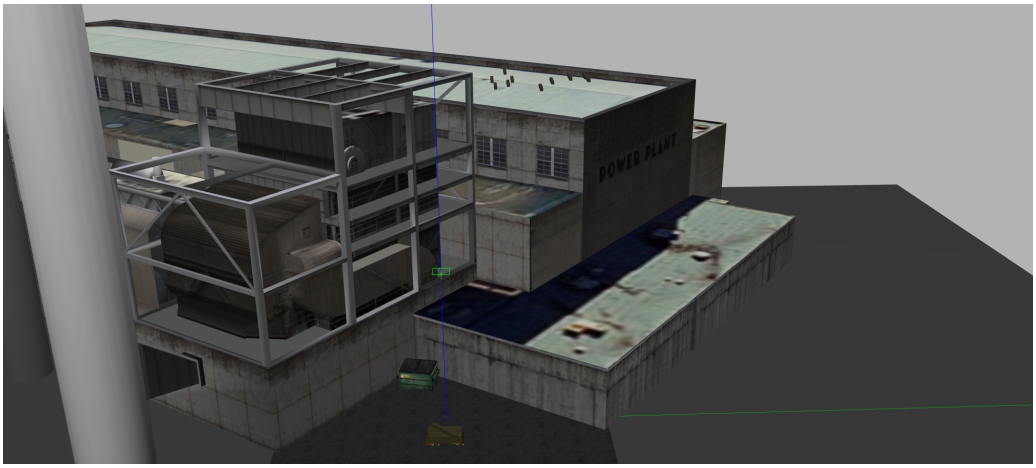
Figure 38: *The Ocean simulation environment.*

Rockwool

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone env:=rockwool
```



(a) Image of the SDU Drone close to a dumpster and a Rockwool pallet.



(b) Images of the Power Plant.

Figure 39: Images of the Rockwool simulation environment.

HEADLESS mode

If you are not interested in running the graphical interface, use the `gui` parameter

```
roslaunch eit_playground posix.launch vehicle:=sdu_drone gui:=false
```