Blood Sample Transportation UAS
RMUASD - Group 3

Per, Blazej, Oscar, Mads, Thor, Mathias

January 2, 2019

Contents

Abbreviation

API Application Programming Interface

BEC Battery Eliminator Circuit

BVLOS Beyond Visual Line Of Sight

DOF Degrees of Freedom

DP Deconflict Pilot

ETA Estimated Time of Arrival

FC Flight controller

FoR Frame of Reference

GCS Ground Control Station

GDPR General Data Protection Regulation

GPIO general-purpose input/output

GSD Ground Sampling Distance

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

IC Integrated Circuit

LAP Landing Pilot

LED Light Emitting Diode

LOP Loiter Pilot

MP Mission Pilot

MPE Monocular Pose Estimator

MSL Mean Sea Level

NED North, East, Down

NFZ No Fly Zone

PID Proportional–Integral–Derivative

QGC QGroundControl

RC Remote Control

REST Representational State Transfer

ROI Region of Interest

ROS The Robot Operating System

RPi Raspberry Pi

SAIL Specific Insurance and Integrity Level

SBC Single Board Computer

SOC State Of Charge

TRL Technology Readiness Level

UAS Unmanned Aerial System

UAV Unmanned Aerial Vehicle

UI User interface

UTM Universal Transverse Mercator

UTMS Universal Traffic Management System

WGS84 World Geodetic System 84

WP Way Point

MPE Monocular Pose Estimator

FOV Field of View

1 Introduction

Logistics has always been a problem in communities. For example, when doing long range transportation, big trucks and container ships are used which can be very efficient. However, small deliveries such as blood samples from a medical clinic in the outskirts of a country, it is usually delivered by truck [1].

The transportation takes a big toll on the environment, considering that a low amount of blood samples are taken at these clinics while still having a high priority. It is also very cost ineffective to use delivery trucks with a driver to deliver the blood samples.

Numbers in this report [1] suggests that on Funen alone, 1.5 mio DKK is used yearly on transport. If an automated system involving UAVs could provide blood sample delivery, no driver would be required and the transportation could happen at a much faster rate at any time during the day/night.

Such a system should be able to handle any situation that might occur during loading, takeoff, shipment, landing and offloading. A flowchart describing such a system can be seen in figure 1. The human operand would only be responsible for loading the blood samples and choosing which the UAVs delivery destination.

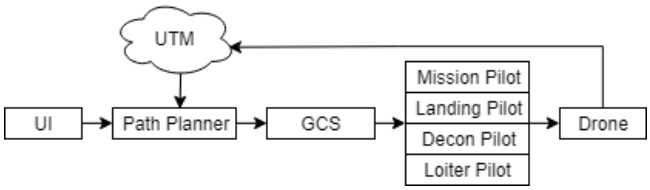


Figure 1: *A simplified flowchart describing an autonomous system that could deliver small blood samples.*

This project presents a system which transports blood samples via the use of UAVs. It was out of the scope of this project to develop a state of the art system, but the work described in this report lays the foundation for such a system.

2 Materials and Methods

The material and methods section describes the building blocks of the system and also the how and why these decisions were made described separately.

2.1 Full system overview

The overall system was supposed to connect the user to an autonomous system which could load/unload a drone that then flies autonomously between different locations.

Figure 2 shows how a drone is to fly between two different locations carrying a small payload, all done in an autonomous manner. It should also be clear from figure 2 that the system is to be controlled from a website and the website should only return essential information to the user. The essential information would be a map where the user could see the route, *No-Fly Zones*(NFZ) and the

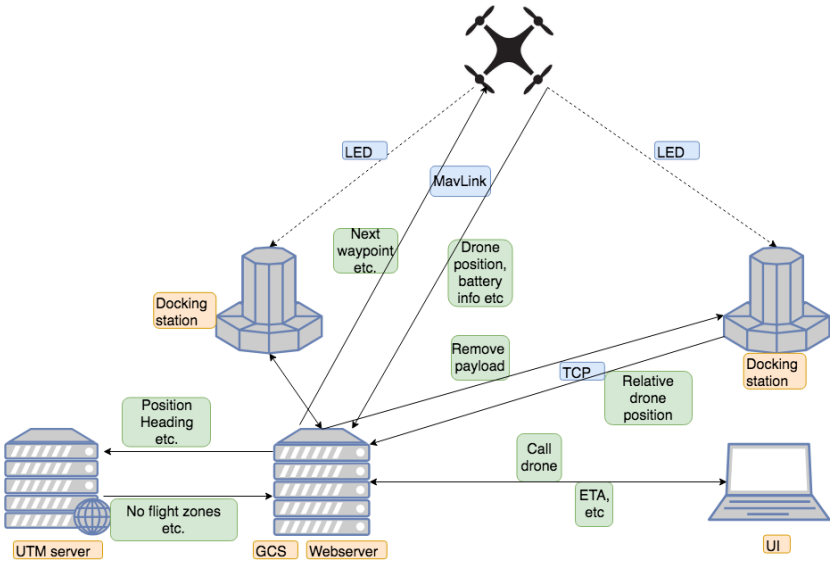


Figure 2: *This figure shows the overall system. Orange boxes are the names of the item, green boxes explains the information being send and the blue boxes is the protocol used.*

Estimated Time of Arrival(ETA). The system was designed to have one server running the web-service for the *User Interface* (UI) and acting as a *Ground Control System* (GCS) for the system. Each destination/take off site would need a docking station for loading/unloading the payload and one drone to do the deliveries.

2.1.1 Specifications of the project

This sections contains a quick overview of the specifications in this project.
Drone

- Mass: 2 kg
- Body: 40x40x15 cm
- Power: 4 cell LiPo battery
- Propulsion: 6 BLDC motors
- Propeller: 20 cm
- Flying speed: 10 m/s

Ground control station

- Operation System: Ubuntu 16.04
- Robot Operating System (ROS) version: Kinetic

Docking station

- Processing: Raspberry Pi 3 model B
- Video: Pi NoIR Camera
- Filter: IR lens 700-1100 nm
- Sliders: Stepper motor NEMA 17
- Size: 50cm by 75cm

2.2 Ground Control Station

This section contains the part of the system which is to be located on the GCS sever. The chapter is subdivided in interfaces within the GCS and each explained in detail.

2.2.1 User Interface

In order to make it easier for the users to interact with the system, a "simple" *Graphical User Interface*(GUI) has been designed and implemented. The GUI is available via a web page so that it can easily be accessed across multiple devices such as phones, laptops etc. The web page is by no means a full system control interface, but it gives the user the ability to order the drone to predefined destinations. Furthermore, the page shows the planned path, the drones current location and an ETA; the web page can be seen in figure 3.

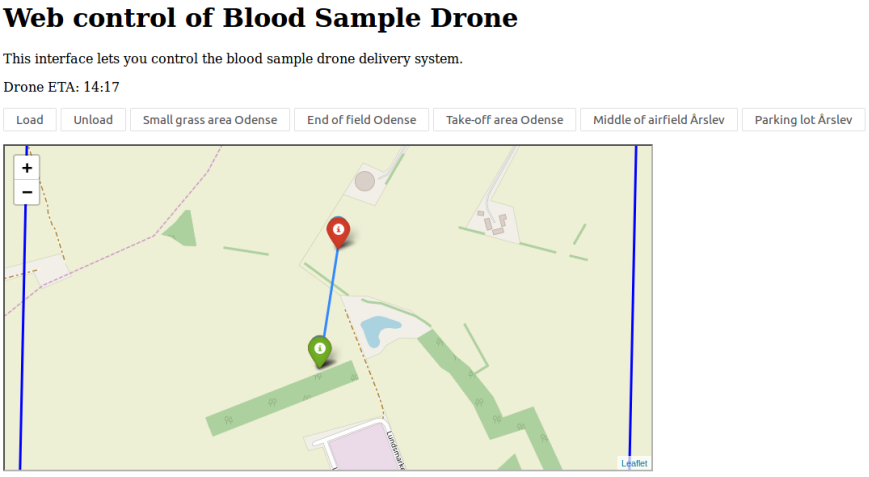


Figure 3: *This shows the layout of the web page with basic buttons, ETA and a planned path.*

The web page communicates with ROS using the ROS bridge suite[1]. The ROS bridge provides a web socket for which the web page can connect to, with the web page being hosted using an apache2 web server[2]. The web page subscribes to a single topic which contains the ETA of the drone. This is updated every time a new path for the drone is calculated. Each time a button on the web

[1] `http://wiki.ros.org/rosbridge_suite`
[2] `https://help.ubuntu.com/lts/serverguide/httpd.html`

page has been pressed a message is published. This message contains a number from 0 to 6 which corresponds to the button pressed. The seven buttons on the web page allows the operator to load and unload the drone and to send it to 5 different locations. The web page shows the user a map of the planned path and detected NFZs, outputting it as a HTML file generated by the path planner.

2.2.2 Main State Handler - Central GCS

When creating a large system as required for this project, it is important to have as high a level of reliability as possible. This can be achieved by using a state machine to control how the system is reacting in certain conditions. A ROS node was created that listens to the different parts of the system, reacting and changing state according to figures 4 and 5. The node then hands out permission for controlling the *Unmanned Ariel Vehicle*(UAV) to the respective pilots as seen in figure 1.

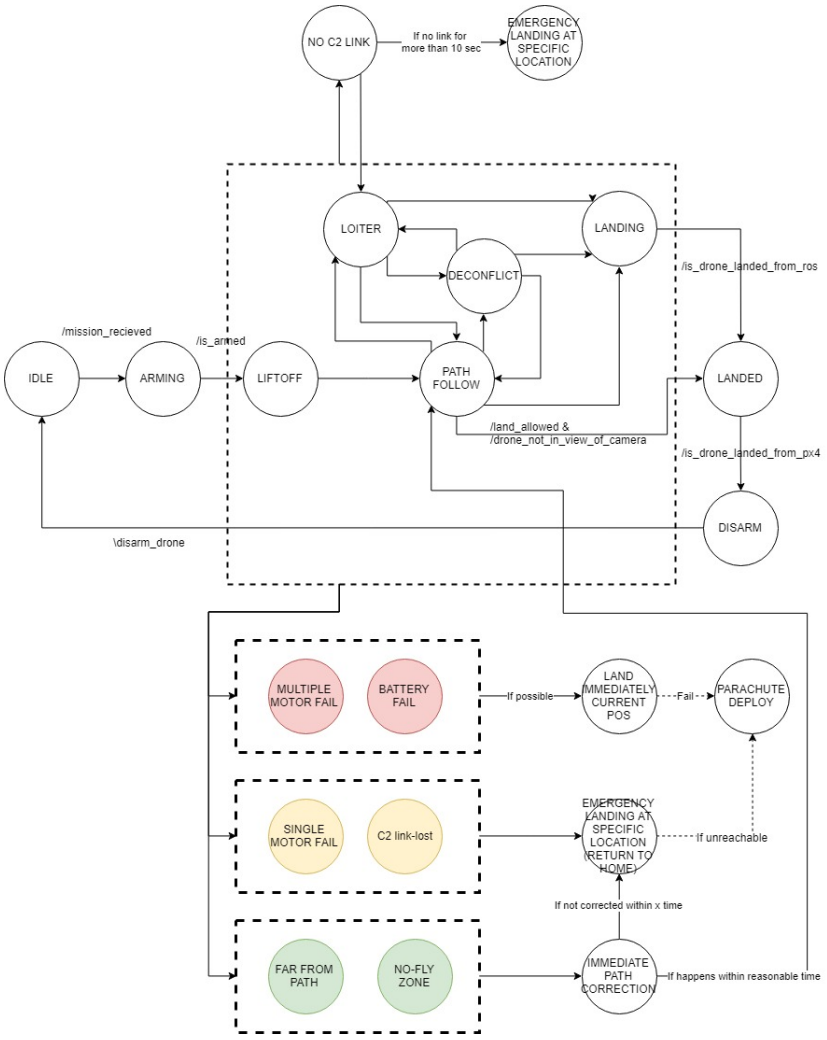


Figure 4: *Complete state diagram of the State handler, with message types in between. The dotted box encapsulates the states where any risk is involved and the actions taken if a failsafe measure is enacted.*

During operation the drone can be defined as working from a set of nine states, changing according to what is published on specific topics in the network. The flow generally follows from figure 1 in the introduction.

The general flow from the state handler goes from IDLE to DISARM, but as faults are possible safety measures have to be taken. The dotted box is placed around the states where these errors has an impact. If any of the errors occur, they are handled either by the code provided by the team, or the failsafe implemented by the PX4 flight-stack.

The dotted box in figure 4 is expanded in figure 5 which shows the states where the highest risk is associated. This means that most consideration has to be taken here and is why the state diagram is more intricate. When performing a normal mission, the state is PATH_FOLLOW and it is the mission pilot that

has control. The mission pilot continues to have control until it arrives at the last *Waypoint*(WP). Here a timer is started and depending on if the docking station camera sees the UAV or not, it is either the *Landing Pilot*(LAP) or the automatic landing that takes control with the latter happening after a 20 second timeout.

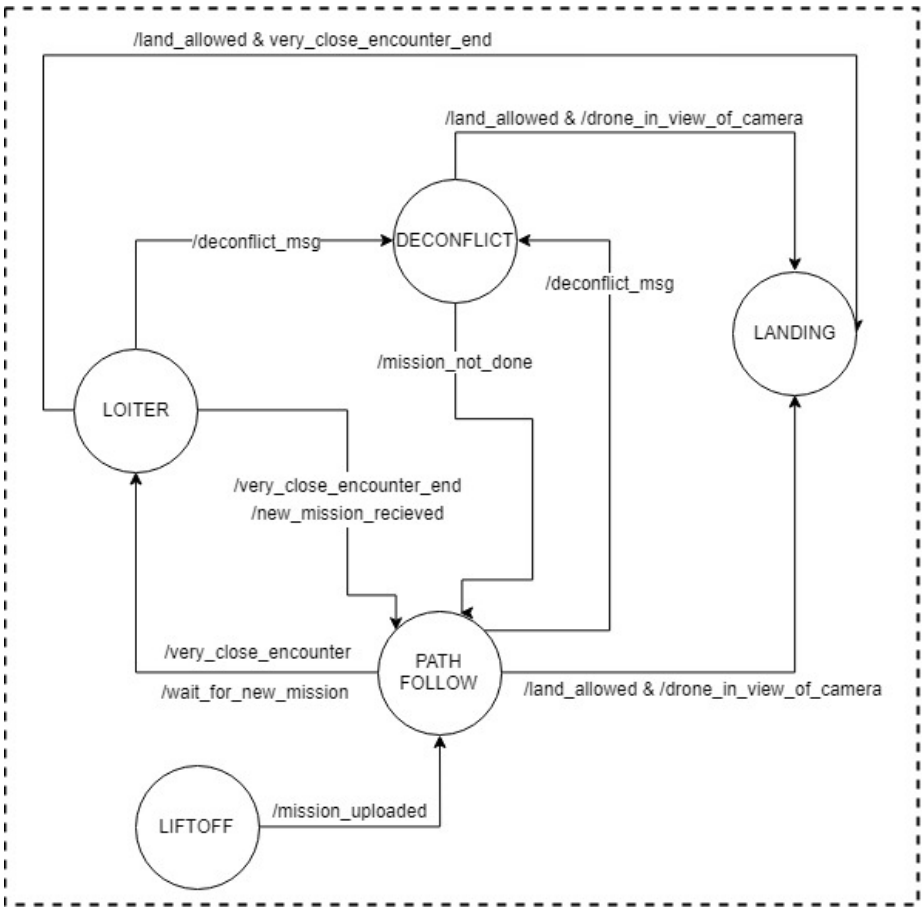


Figure 5: *This is the part of the state machine which is associated with risk. This is an expansion of figure 4 and gives more insight in how the pilot handling occurs.*

In the scenario when another UAV approaches, a ROS node calculates if a deconflicting action is required as described in section 2.2.4. If the approaching UAV is too close to perform a deconflict then the *Universal Traffic Management System*(UTMS) node will resolve this. Depending on which, either the *Deconflict Pilot*(DP) or the *Loiter Pilot*(LOP) takes control and waits for the instance to be resolved.

The topic with a current state is published only once to not clutter the network when changing states. An example of this can be seen in figure 6. This code shows what happens in the LIFTOFF state. Each method is similar, although some have more checks than others. The state machine waits for a message from the network, after which it continues through the states.

```
def liftoff(self):
    self.drone_landed.data = False # the UAV is no longer landed
    if self.mission_uploaded.data == True:
        self.curr_mode.data = MISSION_MODE_ACTIVE
        self.curr_mode_pub.publish(self.curr_mode)
        self.state = PATH_FOLLOW
    return "Rdy to liftoff"
```

Figure 6: *Code snippet of LIFTOFF state and how the state machine proceeds*

2.2.2.1 Terminal GUI It is necessary to monitor the drone and the GCS at all times. A terminal GUI was developed for this project, with a starting point in a minor example from the MavlinkLora library. The library was developed by Kjeld Jensen. The displayed data is both from the drone and from the developed GCS and is shown in figure 7.

The drone is transmitting data to the GCS, such as battery voltage, through various ROS topics. Most drone information is readily available, as part of the MAVLinkLora serial library, which utilizes the MAVLink protocol[3] to communicate with the Pixhawk *Flight Controller*(FC). The use of the MAVLink protocol is further described in section 2.2.7.

Other data used internally as part of the GCS, such as the current pilot or mission state, is gathered from the subsystems running in parallel as ROS nodes, which publishes data to the ROS topics. In the ROS node running the Terminal GUI, all these topics are subscribed to and their information is displayed in real-time into the terminal for debugging.

Since there is a vast amount of information available for the drone, only important data was selected to be shown via the terminal GUI. Alongside this, all ROS messages are logged for later inspection.

[3] `https://mavlink.io/en/messages/common.html`

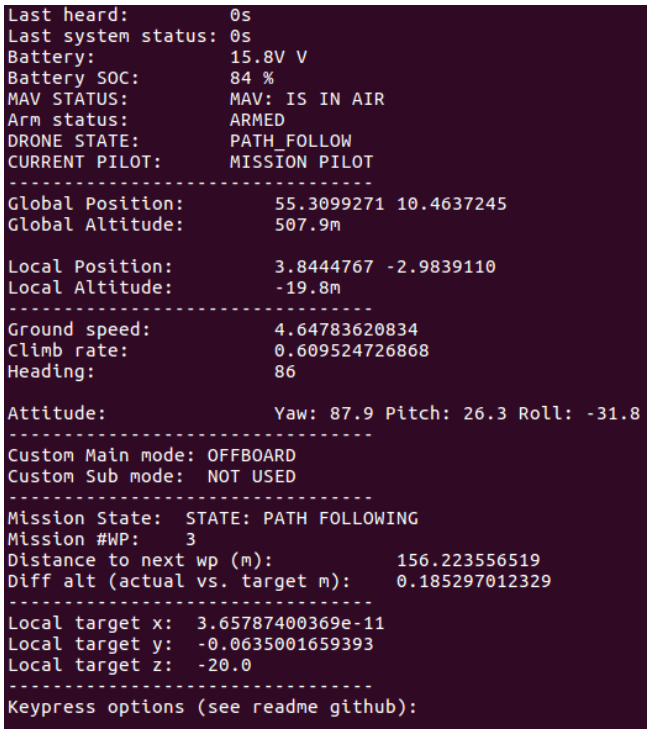


Figure 7: *A snapshot of current information during an operation, as displayed on the GUI. It is possible to monitor the drones position, states and other critical information during the whole flight.*

The terminal node was also responsible for sending heartbeat messages to the drone. This heartbeat signal was used to notify the drone of an active GCS.

2.2.2.2 Health Monitor When having a large system with lots of intricate connections, it is important to keep an overview of everything to quickly solve possible problems.

To achieve this, every ROS node was required to publish their operational frequency during every cycle (known as a heartbeat). The health monitor then compares the supposed frequency of the heartbeat against the actual rate at which these nodes run.

This is defined as the nodes health and is displayed in a terminal window. If the health of the node falls below 90%, it is considered "dead" and the user is made aware.

A subscriber is set up for every node, and a callback for every subscriber is made according to the pseudo code in algorithm 1. This status is then used to create a compact table that is easy to read.

Algorithm 1 Health node monitor

Require: update_rate_msg
1: now = current_time
2: $target\_rate$ = update_rate_msg
3: $actual\_rate = \frac{1}{now - target\_rate}$
4: **if** $actual\_rate$ <= target_rate $* 0.9$ **then**
5: **return** $status \leftarrow$ Dead
6: **else**
7: **return** $status \leftarrow$ Alive
8: **end if**

Due to the complexity of the system, it may be difficult to observe when errors occur visually through the terminal output. Therefore, it is advantageous to also have some audible indication besides the health table to indicate the system health. This was achieved through a provided Busylight [4] which is able to play a tune and has a RGB LED at the end. The Busylight can be interfaced with python via the PyBusylight library[5], where it causes the Busylight to flash red and play a tune if a single node is "dead".

2.2.3 Universal Traffic Management System

In order for the drone to navigate through the airspace in a safe manner, the path planner needs information regarding obstacles and hazards within its operational area - these can include other active UAVs and NFZs. A Universal Traffic Management system (UTMS) was provided which could generate server responses to traffic and zone requests through the internet.

To be granted access to the UTMS, the UAV must first be registered on the server by providing technical vehicle information such as weight, maximum speed, endurance and GDPR compliancy. Contact details of the drone operator must also be included in case of emergencies. In return, the UTMS generates an identification number and authentication key which can be used sending information on the drones position and status. The UAV for this project was designated ID 3011, which can be seen from a technical request in figure 8

```
oschofield@obsLaptop:~/SDUSem3/RMUASD/RMUASD-Team3-2018/utmService$ ./regUAV.py
[utmServer] - Technical details for UAV: 3011
Drone_ID 3011,   Drone_name SDU RM-UASDE18 Group 3
Drone_Weight (Kg) 2.00, Drone_Max_Velocity (m/s): 10.00
Drone_Max_Endurance (s) 1800
```

Figure 8: *Technical Detail Request of the project UAV*

To send tracking data to the UTMS, the node must poll information from various topics on the GCS to ensure that the data is accurate for other users of the UTMS. A typical tracking submission includes the UAV's current position, speed and heading along with the position of the next waypoint. The current operational status is determined by monitoring messages sent by the GCS state handler and converting these to predetermined codes as specified in the UTMS documentation. It is worth noting that the current implementation does not

[4] https://www.busylight.com/en/kuando-busylight-uc-alpha
[5] https://github.com/ericpulvino/pyBusylight

have any way to discern whether a payload is mounted to the UAV, other than the user clicking the on-/offload buttons on the web interface.

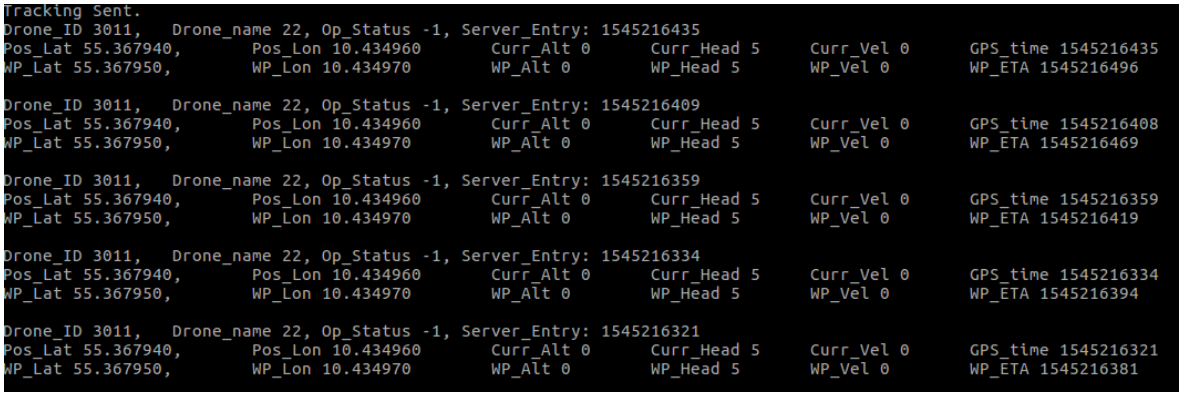


Figure 9: *Screenshot showing multiple tracking updates for the same UAV.*

On receiving tracking data, it can be seen in figure 9 that the server response can be populated with multiple reports from the same drone, resulting in a lot of unwanted information that needed to be reduced. The UTMS handler reduces this data down to the most recent submission and publishes it to a topic for the path planner to use.

Prior to implementing the deconfliction node, the UTMS handler was also responsible for checking and monitoring the proximity of traffic. If it was found that another UAV came within a proximity of 50 meters, a critical alarm would be sent through a topic to the state handler - this would loiter the drone until the hazard had passed.

In addition to providing real-time tracking data for airborne traffic, the UTMS is also capable of providing data for both static and dynamic NFZs. The UTMS handler within the system is capable of requesting for this data in a similar fashion to the tracking data, and parsing it into a desired format.

During operation, the dynamic NFZ parser is constantly polling for information from the UTMS and updating the handlers memory, dependent on whether any of the information has changed. This list would only published to a topic when it received a list request from the path planner, being implemented as a 1 sent on the '/UTMrequest' topic.

2.2.4 Deconflict

The UAV in this operation is supposed to fly *Beyond Visual Line Of Sight* (BLVOS) without any visual "detect and avoid system", this means that some other avoidance system had to be implemented. As the operation is taking place inside an atypical airspace with only known UAVs, a common avoidance system was agreed upon. This avoidance approach is called "deconfliction" and is implemented in this project through a Deconflict Pilot (DP), which controls the UAV in accordance with the state machine in section 2.2.2. The deconfliction strategy was agreed upon by the three groups each developing different UAV systems but with the same deconfliction node. Kasper Høj Lorenzen wrote

the deconfliction module as a ROS node which subscribes to two topics, and publishes one topic like below:

- Subscriber
  - Drone ID
  - UTMS information on the UAVs
- Publisher
  - Global GPS coordinates linked to and ID

The DP then uses the drone command handler to relay these coordinates into local coordinates for the UAV to adjust its position.

The deconfliction system tries to predict possible collisions up to two minutes into the future, by looking at its current heading and velocity for each drone registered in the system. If two drones are predicted to crash, a course correction is transmitted to both to adjust their operational altitudes. The corrections are calculated as follows:

- The drone with the highest numerical heading increases its altitude while the one with the lowest numerical heading lowers its altitude.
- If both drones has the same heading the drone with the highest drone ID increases its altitude while the other lowers its altitude.

2.2.5 Path Planner

This section describes the thought process used in designing and creating the path planner. It will also describe the different challenges the authors imagined the path planner would have to face and how they are dealt with.

The path planner is responsible for planning a route from the UAVs current position to a docking station at a known location. The path may be obstructed by static and dynamic NFZs. The planner is also responsible to plan around these at a safe distance. Furthermore, the planner should be able to handle different special cases such as:

- Not taking off in a NFZ.
- Plan out of a NFZ if it suddenly occurs at the drones position
- Handle if there is a NFZ at the goal position.

,

The path planner was written as a standalone ROS node and uses the input from the web page to get a goal for the route it is planning.

The path planner uses the UTMS handler to obtain information regarding NFZs. Where dynamic NFZs are published regularly whilst static NFZs are only published once. Both static and dynamic NFZ are taken into account during route planning. The path planner starts to create a path as soon as a button on the web page is pressed. The message from the UI is an integer between 0-4 which corresponds the destination button from the webpage; the path planner has five predefined locations which it can use in planning process.

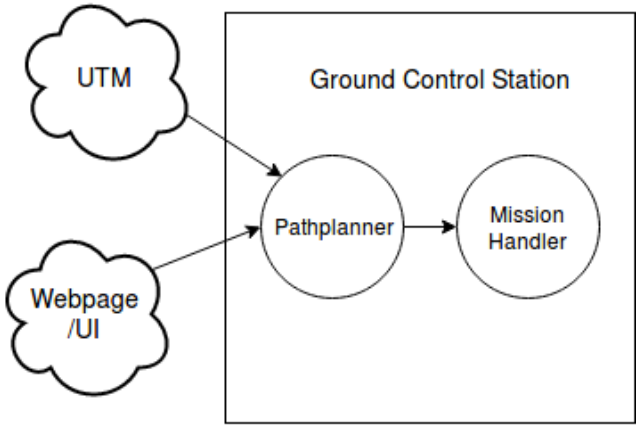


Figure 10: *This image shows a simplified information flow required during the initial path planning.*

The path planner starts by loading every known static NFZ from a file, previously created based on information from the UTMS. It is assumed that this file is updated regularly by the end user of the system and it is also assumed that the static NFZ does not change frequently - as implied by the name. After the static NFZs are loaded, the program waits for the destination to be sent from the UI.

On receiving the destination from the UI, a message is published on the *UTMrequest* topic which commands the UTMS handler to publish data regarding dynamic NFZs.

In order to find a path from the drones current position to the docking station, a graph data structure is required. A structure called a visibility graph was chosen because it was estimated that it would lower the number of edges and vertices of the graph - compared to using a grid with a fixed distance between the vertices. To generate the visibility graph, the python package pyvisgraph[6] is used. The graph building time is based on the number of vertices that is why its important to keep the number of vertices as low as possible.

2.2.5.1 Prepossessing of the polygons The NFZs are imported as list of zones, where each zone consists as a set of coordinates in the WGS84 datum format. To allow for the zones to be easily worked with, the coordinates were converted to *Universal Transverse Mercator*(UTM) coordinates using the code provided by Kjeld Jensen. In order to ensure that the NFZs are not violated, they are buffered with an additional distance of 50 meters. The reason for this is to prevent the case of overshoot due to GPS errors.

The drone is assumed to fly at a speed of 10m/s and the communication is assumed to be with 20Hz which means that the drone will have moved 0.5m pr.

[6] `https://github.com/TaipanRex/pyvisgraph`

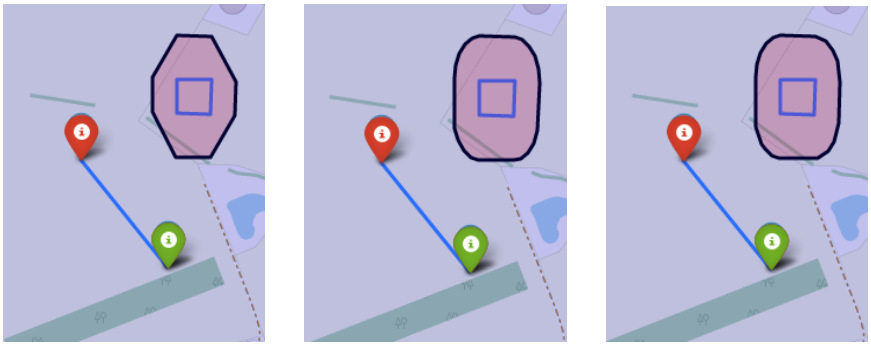
transmission as seen in equation 1.

$$\begin{aligned} Distance &= Speed * Time \\ Distance &= 10\frac{m}{s} * \frac{1}{20} = \frac{1}{2}m \end{aligned} \tag{1}$$

The GPS has an accuracy of 2m in horizontal position accuracy[7] [8] which is used to calculate a worst case scenario. This means that the drone will have between 4.8 and 5 seconds as seen from calculation 2 to react before entering the "real" NFZ.

$$\begin{aligned} Time &= \frac{Distance}{Speed} \\ Time &= \frac{50m}{10\frac{m}{s}} = 5s \\ Time &= \frac{50m - 4m}{10\frac{m}{s}} = 4,8s \end{aligned} \tag{2}$$

The polygons are buffered using the python package Shapely[9]. The ideal buffer algorithm creates an arc at each corner. To reduce the number of points in the polygon the group decided to buffer the polygons with a square. Examples of buffering with a square with different resolution can be seen in figure 11. This was also done using the Shapely package.



(a) *The image show an original NFZ marked with blue and the buffered NFZ with black and red fill with the resolution set to 1.*

(b) *The image show an original NFZ marked with blue and the buffered NFZ with black and red fill with the resolution set to 5.*

(c) *The image show an original NFZ marked with blue and the buffered NFZ with black and red fill with the resolution set to 10.*

Figure 11: *Examples of buffering a NFZ with different resolutions of the buffer circle. The resolutions describes how many additional points are created when buffering. This is why figure 11c and 11b are more rounded than 11a.*

If two buffered polygons are overlapping they must be joined otherwise the path planner will see the points inside the polygons as legal vertices. These

[7] http://www.proficnc.com/all-products/152-gps-module.html

[8] https://www.u-blox.com/sites/default/files/products/documents/NEO-M8_ProductSummary_%28UBX-16000345%29.pdf

[9] https://pypi.org/project/Shapely/

vertices will be used if it is the shortest path. This could create a situation were the path planner plans a route directly through two NFZs. Joining the polygons also has the benefits of reducing the amount of vertices in the graph.

Since NFZs are present all over Denmark, the number of NFZs can be significantly reduced by only considering those which are within the vicinity of the drone and docking station. This is done by creating a bounding box, where all NFZs which overlap the bounding box are kept, while the rest are discarded. The bounding box is defined by starting position of the drone and the desired destination and an additional buffer of 500m. This buffer was is arbitrarily chosen to give a more full picture of the neighbourhood around the destination and initial start point.

2.2.5.2 Planning the path The shortest path was determined using Dijkstra's algorithm. The path is defined as a list of waypoints where the first waypoint is the drones position and the last is the docking stations location. The path is then ready to be published.

Besides publishing the shortest path, the path planner node also publishes an ETA, which is based on the distance between the waypoints and the estimated speed of the drone. To give a better estimate for the time of arrival, 20 seconds for landing and take off is added. The calculated ETA is then published as a string to the UI.

The path planning node also generates an HTML file with the planned route, NFZs and the bounding box. This is all plotted on a map using the python package Folium[10]. An example of such a map can be seen in figure 12.

[10]https://pypi.org/project/folium/

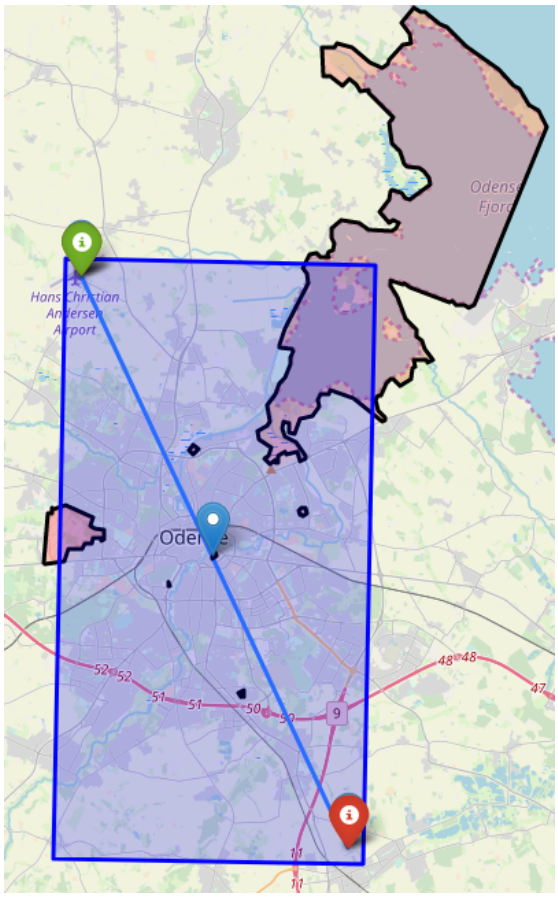


Figure 12: *This figures show the route from a point in HCA airport to Aarslev model airplane site and all the NFZ in the bounding box encapsulating the start and destination.*

2.2.5.3 Special cases The path planner should be able to handle several special cases, in particular:

- Takeoff in a NFZ
- Landing in a NFZ
- NFZ appearing at the drones position.

In order to make a more reliable system these three cases would have to be dealt with. To ensure the UAV did not takeoff whilst situated in a NFZ, a check was made as seen in pseudo code 2 line 1. If it was found that the UAV was in a NFZ, the route was not published and the system prints a ROS *logfatal* to notify the user (2 line 18)

If the takeoff site is clear, the second line checks whether or not the destination site is within a NFZ. If the destination site is found to be within a NFZ, the destination waypoint is removed from the path plan and the path is being published with a message to the drone to loiter replaces it (2 line 5). The next

check determines whether or not the current position is inside a NFZ, in the scenario where a dynamic NFZ suddenly appears. If so, the strategy is then to find the nearest exit of the NFZ, fly there and recalculate the a path from that point. Then the path from the current position to the nearest exit and following shortest path is published to the drone which can be seen from lines 8-12.

Algorithm 2 Path planner - NFZ checks

Require: ShortestPath

```
1:  if TakeOfSite! = Obstructed then
2:      if LandingSite == Obstructed then
3:          shortestPath.pop()
4:          Systeminfo ← Destination blocked
5:          PathPublisher() ← (shortestPath, Loiter)
6:      else                                          ▷ Check for dynamic NFZ
7:          if CurrentPosition == pointInPolygon then
8:              Systeminfo ← Inside NFZ
9:              ClosestPointOnNFZ ← FindClosestPoint()
10:             NewShortestPath ← CurrentPosition
11:             NewShortestPath = shortestPath(PointOnNFZ, Dest)
12:             PathPublisher() ← (shortestPath, Land)
13:         else
14:             PathPublisher() ← (shortestPath, Land)
15:         end if
16:     end if
17: else
18:     Systeminfo ← TakeOffSite is in NFZ
19: end if
```

2.2.6 Mission Handler

This project uses the so-called Offboard mode, which is a developer flight mode that allows fine-grained control over the drone. The use of Offboard mode is further described in section 2.2.7.

With the selection of the developer Offboard flight mode, it is not possible to use Pixhawks own mission handler. A simple mission handler was thus implemented in this project and the overall specification of the constructed mission handler is:

- Simple; only have takeoff, loiter, waypoint and landing as compatible commands.
- Mission loop will be constructed as state machine.
- Run mission loop with a rate of 10 Hz.
- Two helper classes; Mission list and mission items.
- Use thresholds for distance and altitude to traverse through mission list.

The flow of the overall mission handler node is shown in figure 13 and the main loop in the mission handler is shown as pseudo code in algorithm 3.

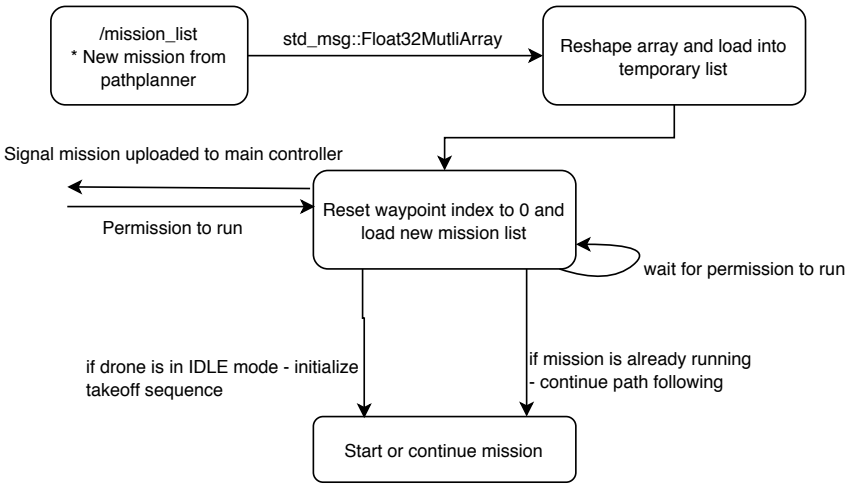


Figure 13: *The general flow for the mission handler, from the mission list is received to the mission is started.*

Only a few central commands like takeoff or WP are used in the current implementation. Further improvements could include additional commands, such as timed loitering. However, the current commands would suffice for this project.

The distance and altitude threshold is used as conditions to increment the mission list. The initial thresholds were chosen as:

- Distance to next WP before increment mission list ≤ 10 meters.
- Whilst moving along the horizontal plane, the difference between target and current altitude must be ≤ 2 meters.
- Distance to landing WP before handing over control to landing pilot ≤ 3 meters.

A threshold distance to the next WP of 10 meters was chosen, so that the drone would never stop at a WP, but rather continue on in a smooth manner. If the drone is travelling at 10 m/s, this will allow 1 second for the algorithm to send the new WP through the system and telemetry to the drone. The actual time for a message to transmit through the whole system was not tested, since it can be visually observed that if a 10-meter threshold was too low, it could be experimentally adjusted if required.

The altitude difference was chosen as to always be close to the target altitude before moving in the horizontal plane. If the drone suddenly receives a mission at different altitude the drone, adjusts its height before moving towards the next WP.

The requirement of less than 3 meters in case of a landing WP is based on the limited FOV of the camera on the docking station. When the drone reaches the final WP, it would initialize the landing sequence by signalling to the main state handler that the last WP has been reached.

The FOV of the camera is $12m$ by $7.5m$ meters[11][12] at a distance of 10

[11] `https://www.3dflow.net/ground-sampling-distance-calculator/`
[12] `https://elinux.org/Rpi_Camera_Module#Technical_Parameters_.28v.2_board.29`

meters from the camera. This allows for a GPS error and a WP acceptance of $\pm 3.75m$ in the worst case. The aforementioned WP acceptance of $3m$ is therefore decreased to $1.75m$.

The mission handler node is running with an update interval of 10 Hz and the mission loop is run at every update. A distance check to the WP is performed based on the global position from the drone, which is received with approximately 4 Hz when using the stock MavlinkLora library. Based on this, the update rate of 10 Hz was considered fast enough. In the case of ROS bandwidth problems, the update rate could be reduced.

Since the mission handler was developed much earlier than the main state handler, the use of drone states inside this node is redundant. It was kept because the mission handler was already tested and working. The states would be published as mission states rather than drone states. This provided a useful debugging tool within the terminal GUI. The internal states used are:

1. IDLE
2. ARMED
3. TAKEOFF
4. TAKEOFF DONE
5. PATH FOLLOWING
6. INITIALIZE LANDING

Algorithm 3 Mission Handler

Require: Mission Pilot permission

Mission State = Idle
if Drone = Armed **then**
    Mission State = Armed
    Load target altitude from first WP
    Mission State = Takeoff
    Local takeoff to target altitude

    if Δ(WP altitude, Drone altitude) $<$ 2 m **then**
        Mission State = Takeoff completed
        Load latitude, longitude and altitude from first WP in mission list
        Mission State = Path following
Path Follower:
        Command drone to fly to global target
        while Distance to target $>$ 10 m **do**
            while Altitude difference between current and target $>$ 2 m **do**
                Adjust altitude
            end while
            Move closer to target WP
        end while

        if Drone is within threshold distance to target **then**
            Load new WP
            if Next WP is normal WP **then**
                goto *Path Follower*.
            end if
            if Next WP is landing WP **then**
                while Distance to target $>$ 3 m **do**
                    Move closer to target WP
                end while
                Send signal mission completed
            end if
        end if
        if Next WP is Loiter WP **then**
            Send signal to initiate loiter
        end if
    end if
end if

To test the mission handler *QGroundControl* (QGC) was used to plan missions and save those mission to a plan file. These files were loaded into a test ROS node, which could decode the JSON format from the file and send the mission list to the mission handler. In this way, the test script could substitute the pathplanner - which was not developed at this point.

The mission list consisted of an encoded list, using the ROS standard message *Float32MultiArray*, and this was decoded in the mission handler. The MultiArray transmit a 1D list along with the dimensions of the original matrix, which is used for decoding.

2.2.7 Drone Command Handler

The command handler is used as a wrapper for the MAVLinkLora library. This command handler will subscribe to various ROS topics, such as */drone_command_arm*, which uses standard Boolean message type to arm or disarm the drone. The overall design in the drone command handler is shown in figure 14.

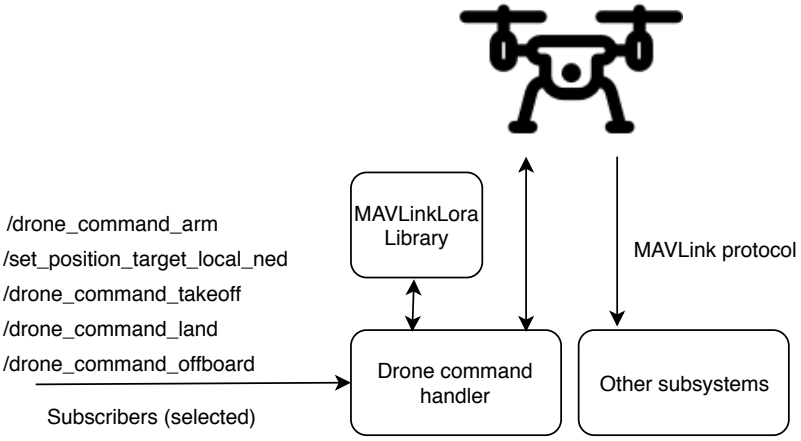


Figure 14: *Generalized overview of the drone command handler implementation. The drone command handler is the only way to send commands to the drone, while all subsystems has direct access to the information received from the drone. The only way to use the command handler is by publishing the correct message to one of the topics.*

There are numerous functions within the MAVLinkLora wrapper, but only a few of them were used in the project. The functions used are summarized here:

- Set local *North-East-Down* (NED)[13] target position
- Arm drone
- Takeoff
- Land
- Activate Offboard mode
- Send heartbeat

2.2.7.1 MAVLinkLora The MAVLinkLora library is provided by Kjeld Jensen as part of this project and was used for the handling of communications. This library implements the MAVLink protocol, which is used for both internal messages on the drone and communication to the GCS. The reason for

[13] `https://en.wikipedia.org/wiki/Local_tangent_plane_coordinates`

using the MAVLinkLora implementation is because of the minimum overhead in the data communication with the drone, as multiple data streams are blocked or limited by the author.

The MAVLink protocol contains a large variety of messages, whereof the most central messages and drone information is already being published to different MAVLink topics when the serial MAVLinkLora library is being run as a ROS node. The main topics used are */mavlink_rx* and */mavlink_tx* with the RX topic being used to grab information from the drone. The most used message in this project for controlling the drone is the MAV_COMMAND_LONG, which is comprised of approximately 100 sub-messages, which give you authority over the the drone (at least what is currently supported in the firmware). An example of using this message is shown in Listing 1.

```
    def set_mav_mode_offboard(self):
        self.msg.header.stamp = rospy.Time.now()
        self.msg.msg_id = MAVLINK_MSG_ID_COMMAND_LONG
        self.msg.payload_len = MAVLINK_MSG_ID_COMMAND_LONG_LEN
        self.msg.payload = struct.pack('<7fHBBB',
                                       True, #param1
                                       0, #param2
                                       0, #param3
                                       0, #param4
                                       0, #param5
                                       0, #param6
                                       0, #param7
                                       MAV_CMD_NAV_GUIDED_ENABLE,
                                       target_system,
                                       target_component,
                                       1 #confirmation
                                       )
        self.mavlink_msg_pub.publish(self.msg)
```

Listing 1: *The COMMAND_LONG message requires 7 parameters. These depend on the desired action for instance, whether it activates offboard or commands the drone to a specific coordinate. In this example, it is used to activate offboard mode and only uses the first parameter. The eighth argument denotes a MAV command, specifying the submessage to be used. The last 3 arguments are system-related.*

2.2.7.2 Offboard The Offboard mission mode plays a large role in this project. An alternative to this is AUTO mode, which is implemented within the PixHawk firmware. The AUTO mode is an autonomous flight mode, which allow to fly pre-programmed missions without the need for communication with the GCS. At the beginning of the project, the pros and cons of each mode were considered. Based on table 1 and 2, Offboard was chosen at the beginning of the project.

Pros	Cons
Offboard mode has already been tested and used in previous courses.	Need to create our implementation of mission handler.
Less time is required to implement Offboard mode, thus making it faster to start flying with.	
Will only required one MAVLink message to change target - no need for uploading full missions.	
The Supervisors have strongly emphasized that telemetry will not be an issue. Therefore, the likelihood of communication loss was minimal.	

Table 1: *Pros and cons of the Offboard mode.*

Pros	Cons
Can use already existing and tested functionality in the Pixhawk FC.	Uncertainty about upload times.
	Reports from previous years students of undesirable in-air motion when uploading a new missions.
	Will take more time to get started with, since it requires modules to be implemented before testing can be done. For example, a mission upload handler would need to be made first.

Table 2: *Pros and cons of the AUTO mode.*

Initially, activating the Offboard mode caused some challenges. However, using the MAVROS[14] example in the footnotes[15], the Offboard mode was successfully enabled. The problem occurred with the rate of setpoints sent to the drone - according to the documentation, the setpoints needed to be transmitted with a rate of at least 2 Hz. However, a higher rate was actually needed to enable and maintain offboard mode. Furthermore, testing in simulation showed that at least 4 Hz was needed to ensure activation of Offboard mode every time. To account for delays and lost packages in the transmission link, the setpoint rate was set to 10 Hz.

From the first flight tests the Offboard mode seemed like a challenge, because the drone would go into fallback mode at random times. The fallback occurred

[14] `http://wiki.ros.org/mavros`
[15] `https://dev.px4.io/en/ros/mavros_offboard.html`

when the setpoint rate is not high enough and this fallback would switch the drone into *Position Hold* and stay at the current position. If the Offboard mode was lost for more than 3 consecutive seconds, the drone was programmed to land at the current position. Alternative failsafes such as "return to home" were considered for future use - but as a preliminary precaution, landing at current position was used. The flight distances at this stage were in relatively short (sub 100 meters), so this was not considered a problem.

Switching between flight modes and arming the drone caused a response if the message was received and accepted. Timeouts to resend commands were not implemented, as the responsibility for arming and Offboard mode was initially handed to the main state handler. To create a more robust solution, one could include timeouts and retries before the drone reported arming failed.

In late October, some of the first major issues with Offboard were experienced, most likely caused by shortcomings in the provided telemetry link. with time to deadlines reducing fast, it was necessary to have group discussion about using Offboard or whether to switch to another solution. Some of the information available when the preliminary decision was made had changed and new information was now available. A new set of pros and cons were considered, which is listed in table 3.

Pros	Cons
Supervisors reported that the final test would be in a highly dynamic environment with shifting NFZs and drones flying over each others paths. Offboard mode can react quickly to changes, as it only required a single setpoint to start the path change on the drone. Fast reactions is required to fly in close proximity with other drones at speeds exceeding 10 m/s.	Telemetry seems to be causing problems with sending continuous WPs at a certain rate.
In Offboard mode the drone do not need to wait for a whole new mission.	The initial long (15-20 seconds) mission upload times in AUTO mode in the provided library was caused by a bug. The new upload times was only a few seconds (reported by group 1 and 2).
The chosen solution for precision landing requires Offboard mode anyway.	Have to implement own mission handler, when PX4 have a thoroughly tested mission handler.
Full control over the drone at all times.	The drone can be put in position hold if it is very critical.
The rumours for the final acceptance day test will be solved better with Offboard, because of the quick reactions.	Offboard mode requires at least 4-10 Hz of setpoints transmitted to the drone to keep it in Offboard mode.
It will be possible to deconflict instantaneous without setting the drone to a position hold.	If the drone for some reason reboots in air, the global origin, which is used for local setpoints can be wrong and the drone will navigate wrongly.
The offboard control was already implemented at this point and tested to be functional to some extend.	In the real scenario, there will probably not be many WPs for the mission.
	In the real scenario, very quick changes are probably not likely to happen (and these will probably be deconflicted by an on-board collision avoidance system).
	If Offboard mode is skipped, this will require new implementation to use non-implemented functionality for missions.

Table 3: *Pros and cons of the Offboard mode with newest intel.*

The decision at this point was to continue using Offboard. However, there were multiple reasons to consider going in another direction. Some of the important factors for choosing the Offboard, was to optimize the drone with quick reactions for the final acceptance day and that the current implementation supported Offboard and Offboard missions.

2.2.7.3 Global2Local All targets used in this project - both the GCS mission plans and the UI, were WGS84 coordinates. As shown earlier in figure 14, the setpoint target topic subscribed by the command handler is using local NED. Global targets are not supported in the current MAVLink receiver module in the PX4 firmware[16] - only local setpoints are currently supported.

The conversion was achieved with the pymap3d library[17]. To use the local frame, an origin was required. This was set to the WGS84 coordinate situated at the position where the drone boots up. MAVLink supports setting the origin, in addition listening to changes to the origin - this could be used in the scenario of the PixHawk rebooting in another place when the ROS system is already running. Despite MAVLink supporting the origin change, the Pixhawk FC does not. If the GCS is restarted, it will load the current WP as the new reference. This could be inaccurate because of the difference in internal origin on the Pixhawk (caused to the problems mentioned above). To overcome this problem a simple solution was chosen; the drone would always be rebooted if the ROS system was restarted. Several solutions were considered, such as saving the boot position to a memory location, which could be loaded into the ROS system. However this was not implemented due to the complexities involved.

2.2.7.4 Loiter and Deconflict pilot The pilots in the project was shown in figure 1 and consist of Mission, Landing, Deconflict and Loiter pilot. The main state handler gives the different pilots clearance to operate the drone (allowing the individual pilots to send target setpoints). The mission pilot was described in section 2.2.6 and the Landing pilot is described in section 2.2.8. The Loiter and Deconflict pilots are very simple and similar in operation and is described in section 2.2.4.

2.2.7.5 Setpoint Repeater The problems with having the Offboard mode go into failsafe caused more and more concern, as multiple flight tests showed this failure. The results from the general simulation did not show these telemetry issues. This fact is obvious, since the simulated telemetry in the simulation have characteristics which resembles a wired connection, rather than a radio link.

Since the telemetry link caused severe problems for the Offboard mode, several solutions were considered. To mitigate the problems the drones antenna was adjustment and an extended ground antenna was created as described in section 2.3.

A test with improved antenna conditions was performed as shown in figure 60 and is further described in section 3.5. The results of the test showed that the time between 2 consecutive messages, was below the desired rate of 10Hz - even dropping below the required 2 Hz needed to maintain offboard. One solution could be to integrate a custom PX4 module as an internal repeater.

However, the project specifications stated that the stock firmware was to be used, since it had been thoroughly tested tested by many users. Therefore, the decision was made to use a hardware repeater. A RPi was mounted on the drone with the wire setup shown in figure 15. This would allow the RPi to listen to all incoming radio messages from the telemetry link on the GCS.

[16] `https://github.com/PX4/Firmware/blob/master/src/modules/mavlink/mavlink_receiver.cpp#L247`

[17] `https://pypi.org/project/pymap3d/`

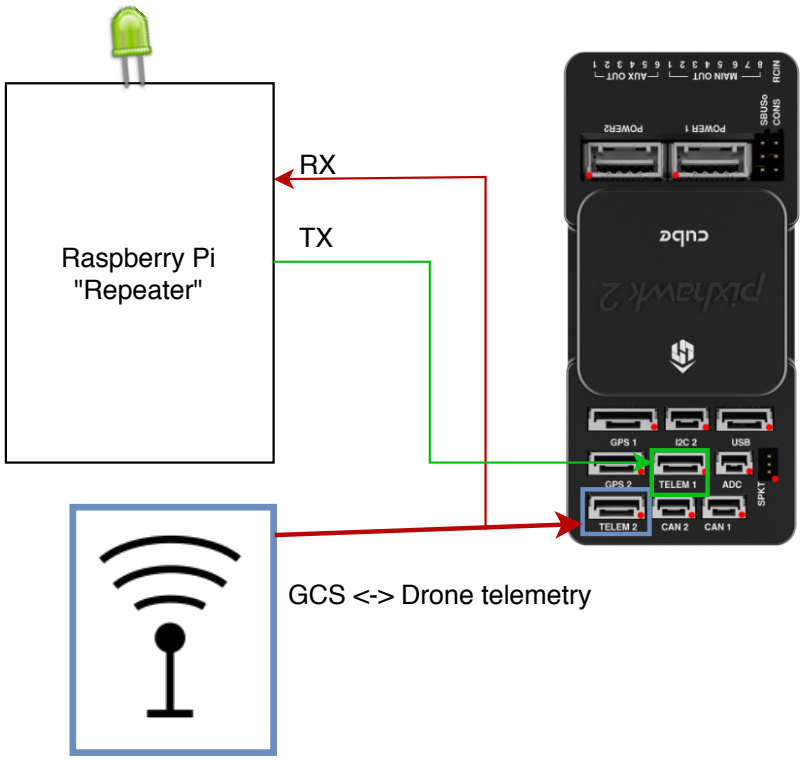


Figure 15: *The setup was very simple. Breakout cables was soldered onto the existing telemetry cables, which allowed the RX (FC RX) cable information to be sniffed by the RPi. The Pixhawk carrier board already had a breakout for two telemetry connections, which allowed easy access to plug in a secondary telemetry input. A LED was attached for visual debugging. When the ROS system was booted on the RPi - the LED would light constantly. When the repeater was active the LED would flash in the rate, which the target messages was received in.*

A ROS node on the RPi decodes all MAVLink messages received, and then repeats setpoints directly to the drone. This setup would allow for a high repeat rate which would always use the latest setpoint transmitted from the GCS. A safety measure had to be implemented, so in case the drone would loose connection to the GCS, the RPi would not keep repeating the old setpoint infinitely. The way this was implemented was to always check the time between two setpoint messages received from the GCS. If there was more than 3 seconds between two consecutive messages the repeater would come to a halt. This would cause the normal failsafe because the Offboard mode was lost. The 3 seconds was an initial estimate, and could cause the drone to continue travel for approximately 30 meters if the Offboard mode was lost. This was not considered a problem as air-buffer zones would take this into account.

This setup would act like a companion computer, which required the PixHawk to be setup using a companion link, and choosing the same baudrate used on the RPi serial connection. The repeater node would repeat the setpoint with

a rate of 20 Hz. As the onboard rate was set quite high, it would not affect the current telemetry connection. Additionally, It was not believed to affect the internal system, as it is geared to handle a higher rate of communication. This is only limited by the PixHawks internal hardware and operating system.

2.2.8 Precision landing

The precision landing system is based on a vision pose estimating algorithm and the embodiment of the docking station. The combined effects of both solutions reduce the position error of a drone during the landing process, each solution corrects the error on a different phase of the process. Five different stages of landing can be differentiated, which has been presented in the figure 16.

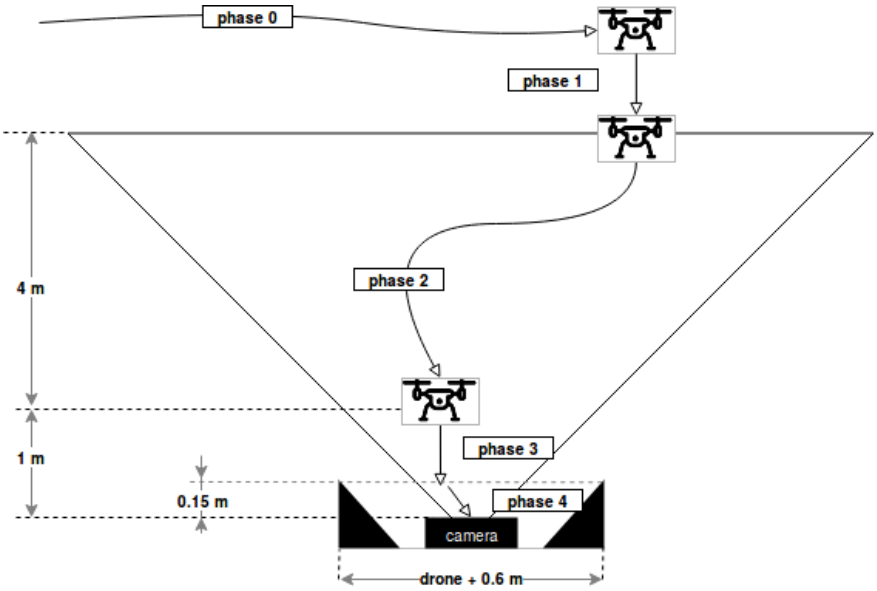


Figure 16: *The five phases of the precision landing.*

- **Phase 0, Arriving.** During this phase, the drone's X and Y target position is based on the GPS coordinate of the docking station. Once the position has been reached, the drone progresses to the next phase.
- **Phase 1, Descending.** The drone altitude is reduced in constant decrements. The Z-coordinate of the target position is changed each time an update of MAVLink local position arrives. The same descending speed is kept through the whole landing.
- **Phase 2, Vision guiding.** The drone enters this phase when the pose estimator starts detecting the drone in the incoming camera images. Usually this phase would start from around 5 meters and last until 1 meter above the camera plane.
- **Phase 3, Intermediate descending.** The pose estimator has difficulties with tracking the drone from approx. 1 - 0.8 meters and below (Figure 17). As a result, there is a short phase where the drone descends without

any adjustments to X and Y target position. This continues until drones landing legs come into contact with the the frame of the docking station.

- **Phase 4, Embodied alignment.** In the last phase of the landing process the drone's X and Y target position stays unchanged yet its position is adjusted by the morphology of the docking station. The funnel-like shape of the docking station allows the drone to keep on descending towards the desired location.

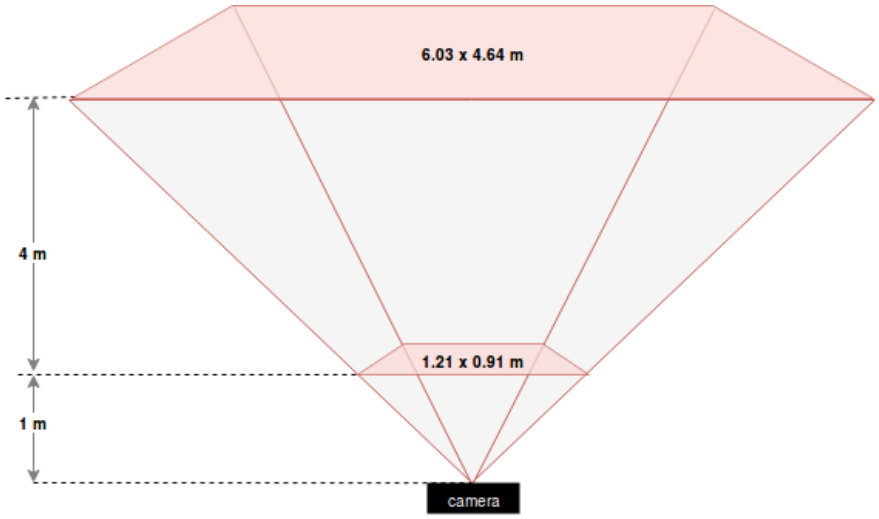


Figure 17: *The FOV of Monocular Pose Estimator (MPE) in 1 and 5 meters from the camera plane. In order to successfully transit through the phases 1 (Descending), 2 (Vision guiding) and 3 (Intermediate descending), the drone has to be located within the FOV. The FOV have been calculated based on the angle of view from the Sony IMX219 camera sensor (table 4).*

The following frames of reference will be addressed later in this section:

- **Global.** Expressed in the longitude and latitude coordinates.
- **Local.** Expressed in X and Y coordinates, with the origin located in the position where the drone has been initiated.
- **Drone.** Expressed using local coordinate system, located in the center of the drone, with the positive X-axis pointing towards the front of the drone and Z-axis pointing upwards.
- **Camera.** Expressed in X and Y coordinates, located in the center of the camera's view, with Z-axis pointing downwards.

The precision landing system is based on a computer vision algorithm, detecting and tracking a set of markers. The position of the markers are defined in the drones *Frame of Reference* (FoR). An active marking system, consisting of four bright infrared LEDs, has been used to help differentiate the markers from the noisy image. A narrow bandpass filter was used, only allowing wavelengths corresponding to the LEDs to pass through, removing all undesirable

wavelengths from the image. The final output of the pose estimator is a pose in 6 DOF (x, y, z, roll, pitch and yaw) given in the camera FoR.

The precision landing can be divided into a ground and drone part. Heavy components of the system, such as the camera and and *Single Board Computer* (SBC), were positioned on the docking station in order to minimize the weight impact on the drone. On the other side, the lightweight LED system was mounted on the drone.

A PID controller has been implemented to control the drone through the landing process. A Kalman filter was used to fuse the velocity from the drones IMU with the pose estimations from MPE, which would reduce the influence of noise on the pose estimation.

Sensor type	Sony IMX219PQ (CMOS)
Pixel count	3280 x 2464 active pixels
Angle of view	62.2 x 48.4 degrees
Video modes	3240x2464, 15 fps 1640x1232, 40 fps 720P, 40-90 fps

Table 4: *The camera specification.*

2.2.8.1 Hardware-based filtering and segmentation

The hardware-based segmentation of the image is done with a combination of a camera, a filter and an active marker system operating in similar wavelength range. As shown on figure 18, active pixel sensors of all three colors of the camera are responsive to the light from around **700 to 900 nm**, with around 30% relative response for the wavelength of 850 nm. The light emitted by chosen LEDs spans from **780 nm to 880 nm**, with a peak of the relative spectral power around 850 nm (figure 20). All the light below **700 nm** has been filtered by the narrow-bandpass filter, with a transmittance of 0% for this range, and a ideal transmittance of 90% for the light of 850 nm, the peak of LEDs output (figure 19).

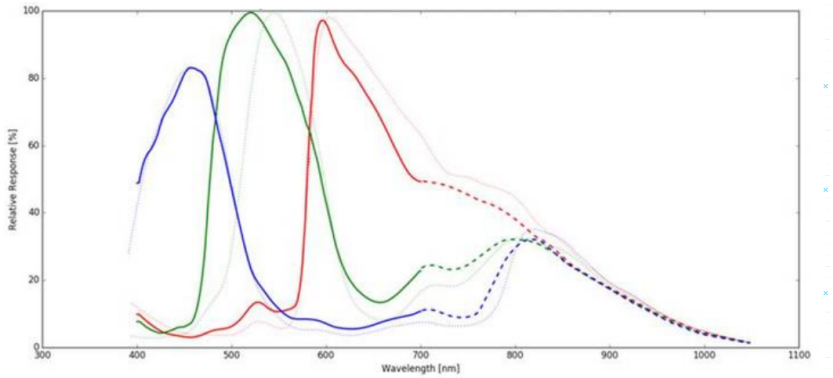


Figure 18: *Sony IMX219 camera spectral response.*

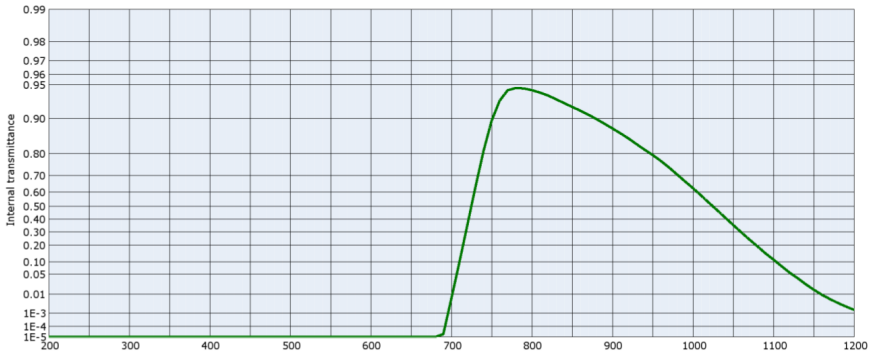


Figure 19: *RG-9 filter internal transmittance.*

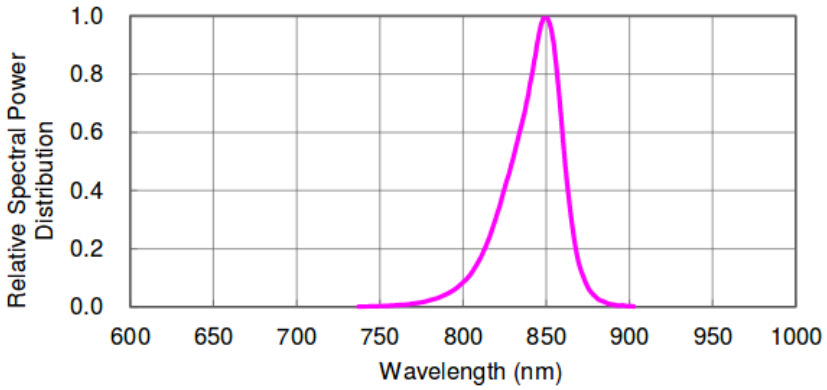
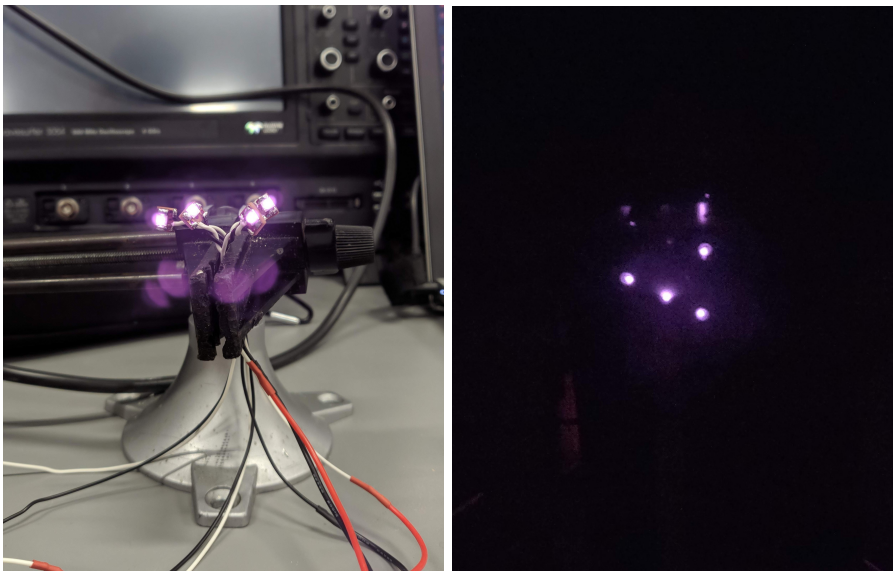


Figure 20: *PK2N spectral power distribution.*

The comparison of the view with and without the bandpass filter can be seen in figure 21. A shot of the sky with the drone in view has been presented in the figure 22. In order to detect the LEDs on the image for tracking, the remaining parts (like clouds, reflections on the drone etc.) have been removed through thresholding, which produced a black and black-and-white binary image as the output.



(a) Without the filter. *(b) With the filter.*

Figure 21: *A comparison of image quality for detection with and without a bandpass filter.*

Figure 22: *The performance of hardware-based segmentation. Most the undesirable parts of the images have been filtered out using the bandpass filter. No additional image processing has been applied to these shots.*

2.2.8.2 LED Driver Circuit For the precision landing to work, the LEDs need to correspond with the bandpass filter frequency and have enough power to be visible during the day. From the data sheet [7], it can be seen that the these LEDs run at an optimal level of 1.6V at around 350mA.

To power the LEDs, they can be placed in a series or parallel with pros and cons for each configuration. If placed in parallel, additional power regulation would be required to reduce the six volt input from the BEC to 1.5V, potentially resulting in a high amount of power loss through heat. Additionally, the current requirements would need around 1.4A to power the LEDS with optimal brightness. On the other hand, the system would be more robust as a single failure of the LEDS would not affect the others, but would cause the MPE not to be able to detect the and calculate the pose - as 4 LEDs are needed for the minimum requirements.

The benefits of a series configuration is that it would be easier to implement; the BEC can be configured to 6V - resulting in four series LEDs being able to fit on a single loop. The downside is that a loss of even a single LED would render the circuit broken and not operational. That being said, the series configuration was chosen as of its lower complexity compared to the extra hardware required for the parallel circuits.

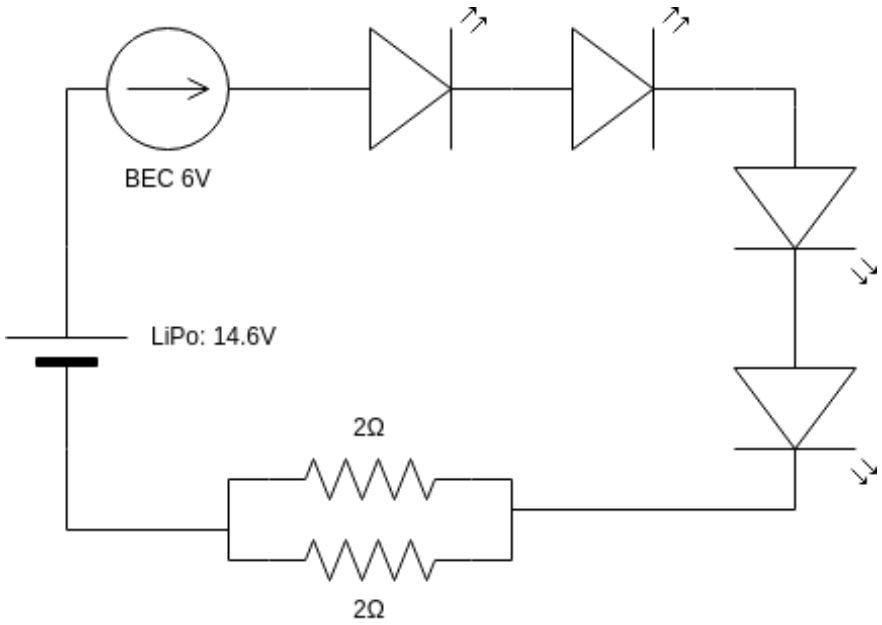


Figure 23: *LED Driver Circuit Diagram*

When designing the LED driver circuit, it was intended that the main circuitry would be mounted on the centre hub of the drone, whilst the LEDS were mounted to different locations on the frame using wire extensions. It was also noted that whilst the LEDS could withstand a current of 1A, they would require significant thermal cooling to remain operational in an 'open loop' setup. Therefore, a current limiting resistor was required to regulate the power consumption.

$$V = I.R \tag{3}$$

To calculate the resistance for the total circuit and current limiting resistor, Ohms law in equation 3 was applied to obtain a total resistance of 17Ω within the circuit. The internal resistance of each LED is approximately 4.3Ω(350mA at 1.5V), resulting in a total resistance of 17.2Ωover four LEDs. As this exceeds the calculations for optimal current to the LEDs, the smallest value resistor of one Ohm was used - Recalculating the total current in the system to around 330mA. To test the calculated values against the real characteristics, LEDs were connected in series to a breadboard and different resistor values were tested. The results can be seen in table 5

Resistor Value (Ω)	Resistor ΔV	Total Circuit Current (mA)
1	3.4	330
2	4	200
3	4.45	150
4	0.4	115

Table 5: *Experiment results for current-limiting resistors*

It can be seen that the results of the experiments accurately reflect the calculations mentioned above. To avoid heat dissipation concerns, the one Ohm of resistance was achieved by putting two 1Ω resistors in parallel.

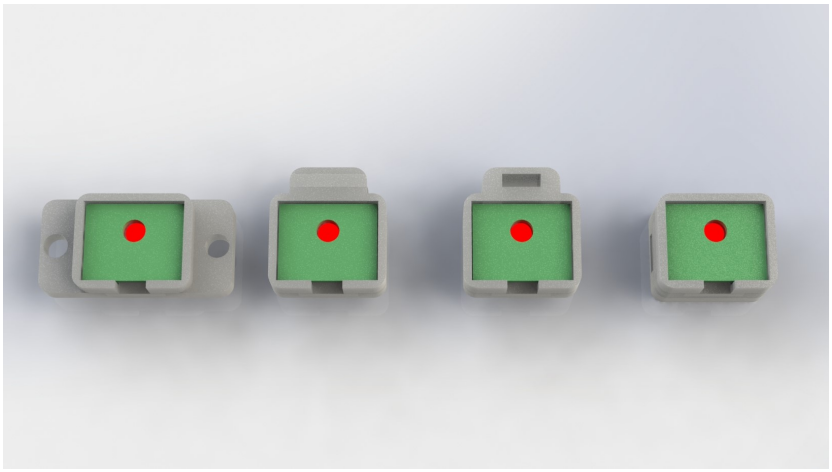


Figure 24: *CAD render of the four different types of LED Cases, All of which were 3D printed in Black PolyLactic Acid (PLA)*

2.2.8.3 LED Mounts Once the LED system had been implemented, casings had to be designed which would allow for flexibility when mounting on the drone. Four variants of the mount were designed, all of which using cable ties to mount them on the drone. These can be seen in figure 24

2.2.8.4 The software overview On figure 25 the overview of the precision landing implemented in ROS has been presented. All the components have been running on the GCS in order to increase the fps of the MPE. The processing power of the Raspberry Pi was not enough to deliver the estimations with a satisfactory rate.

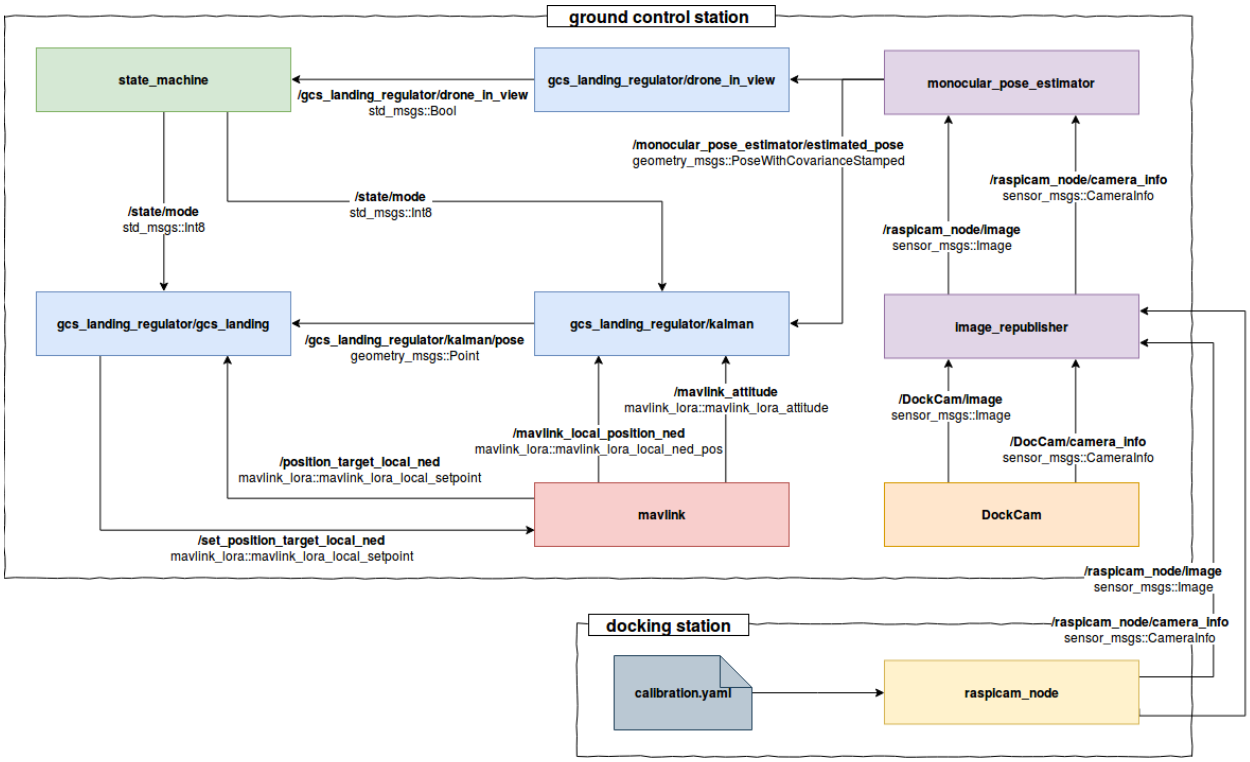


Figure 25: *The overview of the precision landing system.*

2.2.8.5 Monocular Pose Estimator Instead of developing a detector and tracker for the infrared LED system from the scratch, an existing solution was chosen. The selected algorithm has been recommended by the course supervisors as a reliable solution. More information about MPE algorithm can be found in the related publication[2], and a github repository shall be visited for code references.[3]

The configuration of the Pose Estimator proved to be essential for its effective detection and tracking. An in-built ROS tool for dynamic parameters configuration has been used to adjust the parameters (figure 26).

- **Threshold value.** Highly dependent on the lighting conditions. A high threshold (around 200-240) would be usually chosen due to an outstanding brightness of the LEDs in the image.
- **Min and max blob area.** The developers of the estimator limited the upper range to 1000 pixels of area. A bigger area was needed meaning that it has been increased to 6000 pixels. This was to allow the pose estimator to function even close to the ground. The blobs, coming from the LED glar when the drone was situated close to the ground. These exceeded the 1000 cap set by the developers, preventing a detection when below approx. 2 meters from the ground.
- **Max circular distortion.** Some on the detected blobs coming from the background, like clouds, could be easily eliminated by setting a high

threshold on the circularity.

- **Certainty threshold** together with **Valid correspondence threshold** were the most difficult parameters to determine. When set to a high value, sometimes the detector would not accept a true positive pose estimation due to lack of certainty. When the values were lowered, a false positive pose occasionally was tracked. As the success of a precision landing depends on the proper pose, these false positives would often results in the UAV flying out of frame

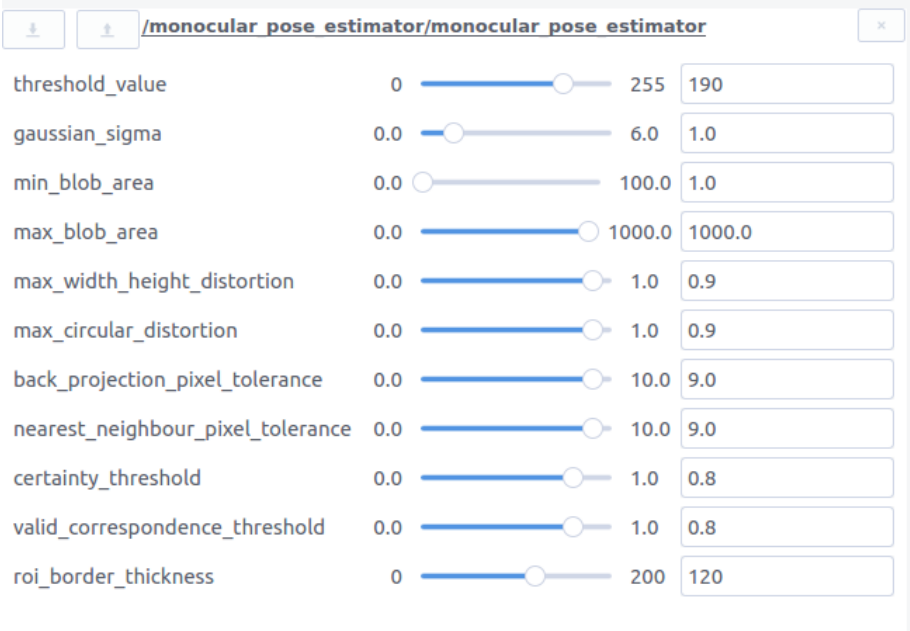


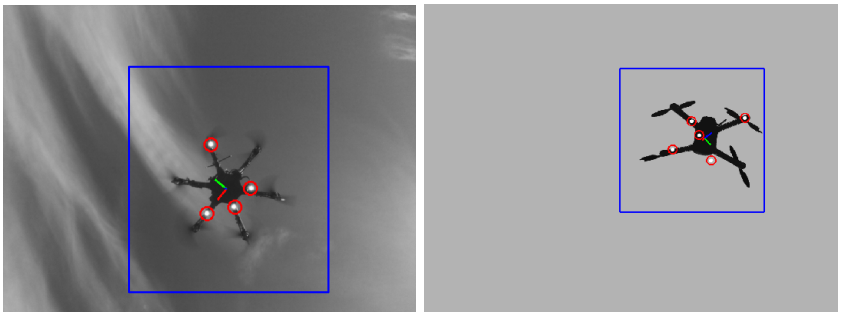
Figure 26: *The configuration panel of MPE package in rqt_reconfigure ROS tool.*

Integration of the pose estimator into the system required a series of pre-emptive steps:

- **Setting up markers** The different LEDs cannot be distinguished from each other in the image, therefore a correspondence between detected blobs in the image and the target object had to be found. The Pose Estimator uses a brute-force approach for matching the points for the first time, namely the P3P (Perspective-n-Point), and then the position is predicted for each consecutive frame. This is tracked within a ROI. In the simulation, a five-LED configuration has been used (figure 27b). CAD software was been used to obtain the precise mounting positions with the respect to the drone frame. A few different configurations were tested for the real-life application, resulting the configuration presented in figure 27a. The considerations for placing the LEDs was as follows:
  - in a non-symmetric configuration

  - each diode lying in the different plane
  - maximizing the volume covered by the LEDs
  - wide angle visibility from the ground perspective
  - satisfying requirement of the minimum 4 LEDs

- **Camera calibration** Matching the points requires an image to be as undistorted as possible, which shows the significance of the camera calibration in relation to the algorithms performance. Two different calibration boards were tested (in A4 and A3 format), with an issue occurring with calibration in a distance higher than 2-3 meters from the camera plane on both. Therefore, all the calibration points have been collected within a relatively close range (considering the typical heights for the drone missions).



(a) *4 LEDs on the physical drone.* ***(b)*** *5 LEDs on the simulated drone.*

Figure 27: *The marker placement on the drone in real-life and in the simulation.*

2.2.8.6 Camera driver The resolution of the acquired images has been reduced to **640x480 px** in order to improve the processing speed of MPE. In comparison to acquisition in **1280x960 px**, the average rate of pose messages improved from **1.2 Hz** to **4.7 Hz**.

2.2.8.7 Image re-publisher The *raspicam_node*, is responsible for image acquisition during the real-life missions, whilst the *DockCam* node is used for the same purpose but in simulation. Both publish messages under two separate topic names. In order to keep the MPE package relatively unmodified, an *image_republisher* node was made.

2.2.8.8 PID controller The PID controller has been implemented in order to regulate the target position of the drone, based on the estimations from the Kalman filter. The integration of both solutions has been presented in figure 28.

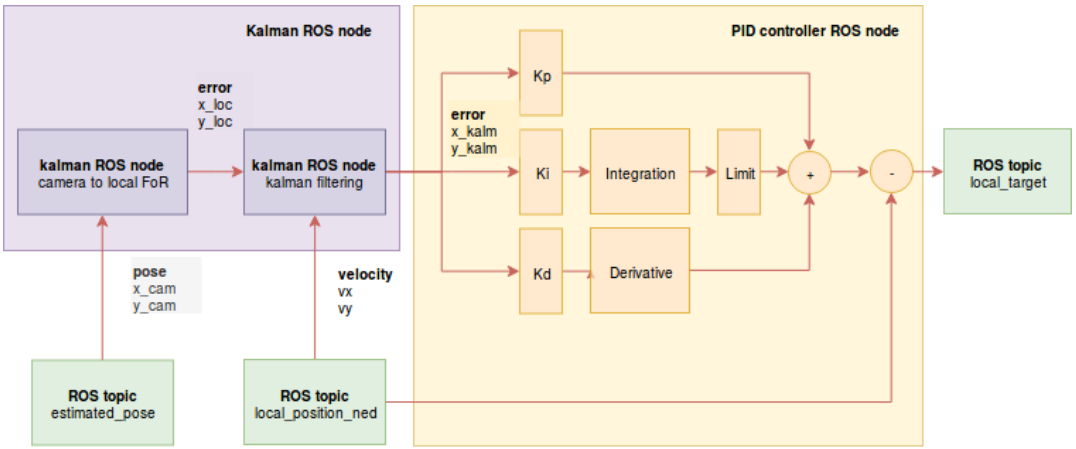


Figure 28: *The integration between the implemented PID controller and Kalman filter.*

The gains and saturation value of the integrator have been chosen empirically and independently for the simulation and real-life tests. The chosen values for each can be seen at table 6.

	simulation	**real-life**
Kp	1.12	0.85
Ki	0.04	0.005
Kd	0.41	0.14
I_sat	0.18	0.2

Table 6: *The gains and saturation value for the PID controller.*

2.2.8.9 Kalman filter For sensor fusion, a first order system had to be designed, incorporating both velocity and position measurements to estimate a pose. An estimation is performed each time a measurement is acquired from either of the sensors, which has been visualized in figure 29. Since the error is defined as a position with respect to the camera FoR, the model of the system can be treated as a tracking problem and implemented solution is a first order system supplied with measurements of position and velocity with two separate sensors (a pose from MPE, and (x,y) velocities from the pixhawk's IMU).

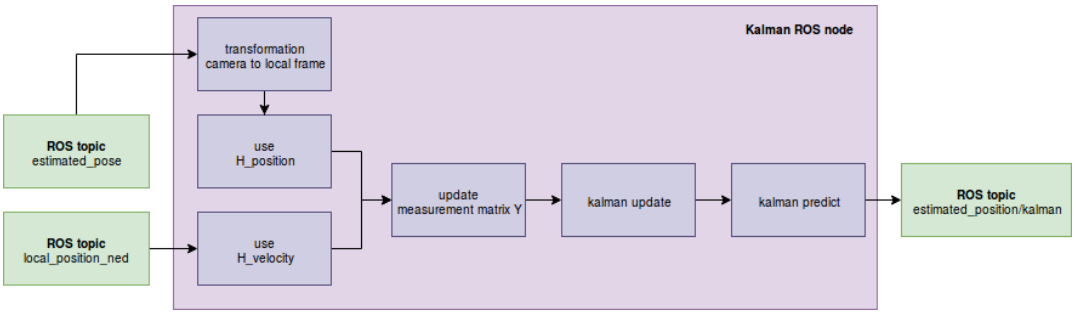


Figure 29: *The flowchart of the kalman implementation.*

A design of the state variable **X** should incorporate the information about both the position as well as velocity. Since the system is used for tracking in two dimensions, the state of the regulation process can be expressed as:

$$X = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \tag{4}$$

As a next step, the state transition function **F** has been defined.

$$F = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The expected change of the model between steps is described by the process noise matrix **Q**, and has been defined as below:

$$Q = \begin{bmatrix} Q_{std}^2 & Q_{std}^2 & 0 & 0 \\ Q_{std}^2 & Q_{std}^2 & 0 & 0 \\ 0 & 0 & Q_{std}^2 & Q_{std}^2 \\ 0 & 0 & Q_{std}^2 & Q_{std}^2 \end{bmatrix} \wedge Q_{std} = 0.95 \tag{6}$$

The measurements of incorporated variables come from two different source:

- Each time the pose estimator detects the drone in a frame, a new pose message is being published in */monocular_pose_estimator/estimated_pose*.
- MAVLink lora publishes a new */mavlink_local_position_ned* with velocities in the local frame.

A measurement function has been implemented separately for each of the sensors.

$$H_{position} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

$$H_{velocity} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The magnitude of the noise in the measurements proved to be hard to estimate. Even if the noise of the sensor is known, other factors can have an unpredictable impact on the measurements - such as the latency in the transmitted data. The reliability of the readings has been chosen based on simulation of the precision landing in Gazebo. The system presented an overall better performance when the readings of velocity were considered noisy, and the pose estimations highly reliable. The pose was read at an approximate rate of 4 to 5 Hz in simulation, where the local positions were three times as frequent. When low error was assigned to the velocity measurements, an occasional peak of readings had a destabilizing effect on the regulation.

$$R = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{9}$$

Each time the */gcs_landing_regulator* ROS package is re-initialized, or the GCS state machine authorize the node as a pilot, the state variables $\mathbf{X}$ are zeroed and the diagonal of the covariance matrix $\mathbf{P}$ is filled with high values in order to make the first measurements highly reliable (or in other words, to tag the initial prediction as highly unreliable).

2.2.8.10 Quaternion to euler transformation The message type used by MPE uses the quaternion system for the pose orientation. But */mavlink_attitude* are given in radians. In order to combine the data, the angles has to be expressed in a common unit. The pseudo-code 4 explains how the conversion from quaternions to the euler system was performed.

Algorithm 4 Quaternions to Euler

Require:

q_x, q_y, q_z, q_w

Yaw calculation

$siny\_cosp = 2.0 * (q_w * q_z + q_y * q_x)$
$cosy\_cosp = 1.0 - 2.0 * (q_z * q_z + q_y * q_y)$
$yaw = atan2(siny\_cosp, cosy\_cosp)$

Roll calculation

$sinp = 2.0 * (q_w * q_y - q_z * q_w)$
if $fabs(sinpy) >= 1$ **then**
    $pitch = copysign(pi/2, sinp)$
else
    $pitch = asin(sinp)$
end if

Pitch calculation

$sinr\_cosp = 2.0 * (q_w * q_x + q_y * q_z)$
$cosr\_cosp = 1.0 - 2.0 * (q_x * q_x + q_y * q_y)$
$roll = atan2(sinr\_cosp, cosr\_cosp)$

return$(roll, pitch, yaw)$

2.2.8.11 Camera to Local FoR transformation Before the position can be corrected, the error has to be translated into the same coordinate system.

As shown in figure 30, the camera FoR is a mirror image of the other, which means that a standard rotation matrix which would transform one into the other does not exist. A non-standard rotation matrix was necessary.

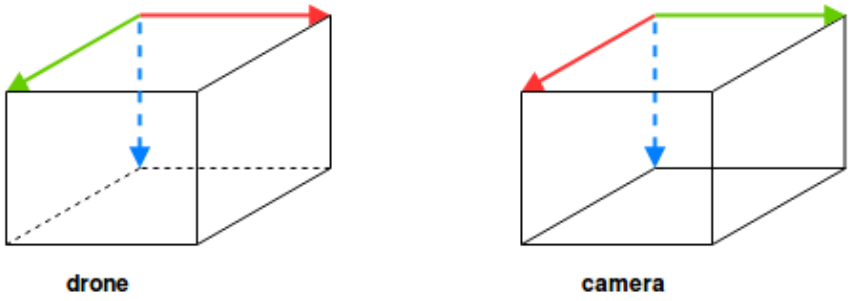


Figure 30: *The problem of transformation between the local and camera FoR.*

The camera frame has to be reflected by around the Y-axis, which can be done by multiplying the **rotation matrix R** by a mirroring matrix.

$$R'^{local}_{cam}(\theta) = R^{local}_{cam}(\theta)M = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} -cos(\theta) & sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{10}$$

To rotate by a certain angle θ, which is a sum of the orientation detected in the camera FoR and the global heading, the coordinates in camera FoR are multiplied with the R_{cam}^{local} matrix, as follows:

$$\begin{bmatrix} x_{local} \\ y_{local} \end{bmatrix} = R_{cam}^{local}(\theta) * \begin{bmatrix} x_{cam} \\ y_{cam} \end{bmatrix} \tag{11}$$

2.2.8.12 Drone detection Before the precision landing system is authorized as a pilot, the state machine has to be informed that the drone is in a view of the docking station. The information about tracking status is forwarded to the */gcs_landing_regulator/drone_in_view* topic, as explained on the architecture diagram in figure 25.

For the tracking to proceed, a timeout of **3 seconds** can not be exceeded between two consecutive pose messages.

In the final tests, this node was disabled due to irregular message acquisition resulting in constant timeouts. Additionally, a more dynamic approach was taken where the landing pilot keeps descending the drone at a target position. The landing pilot expects corrections from the pose estimator, but not depending on them to land the drone.

2.2.9 Simulation Platform

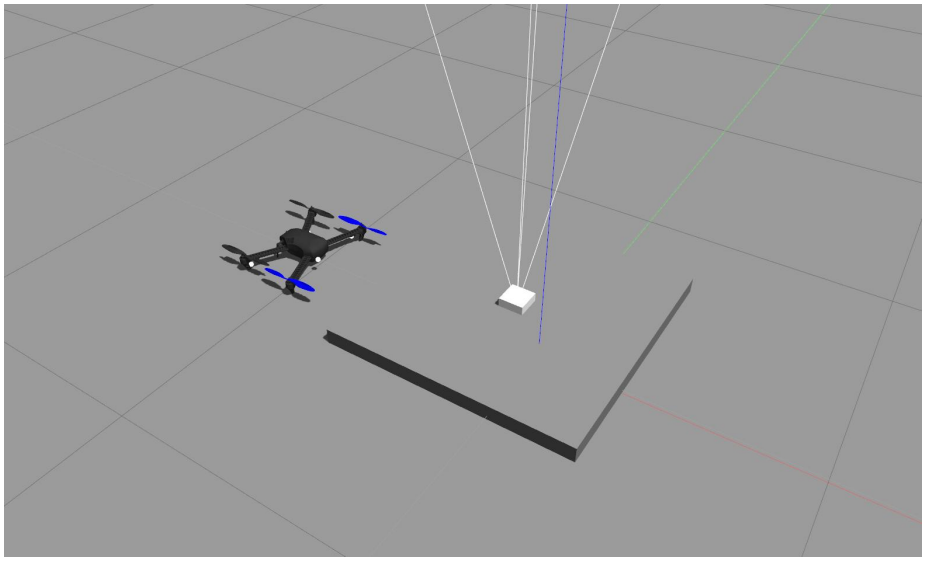


Figure 31: *Virtual Simulation environment within Gazebo-sim.*

During the testing phase, a simulation environment was required to ensure that the ground control station and its respective subsystems functioned as intended without any errors. It also gave team members a chance to experiment with new features without causing damage to the UAV. Gazebo-Sim by the Open Source Robotics Foundation (OSRF) was chosen as the Simulation software of choice, mainly due to its support and integration with both ROS and the PX4 firmware running on the UAV.

Within Gazebo, a simulation environment was implemented that included the basic elements needed for testing the system, such as a functioning UAV and a docking camera. It was intended to use a hexacopter platform available within the OSRF repositories called the Typhoon 580, however it was found during initial testing to be a very complex platform with many integrated features not present on the physical hexacopter used for this project. As a result, it was decided to use the default "Iris" model supplied by the PX4 team and modify the appearance to suit the requirements. For instance, the precision landing required four LEDs to be placed on the drone at specific places in order to be detected. This meant placing uniformly-coloured spheres on the drone as analogs.

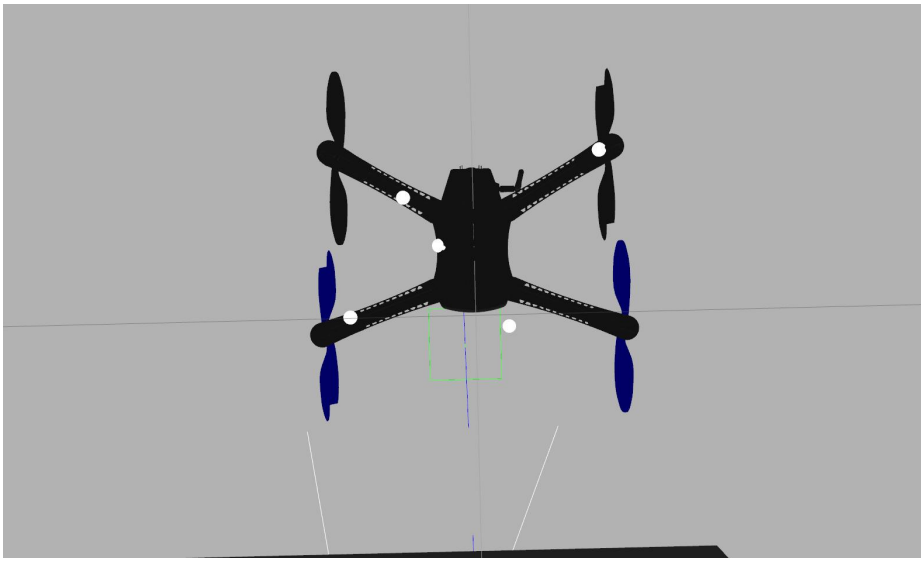


Figure 32: *Simulated drone with white spheres as LED analogs*

The Docking station was also required with the simulation environment, but in order to reduce the model complexity, only the camera module was implemented. This was done using the Gazebo camera plugins, where the properties of the camera were set to accurately represent the No-IR raspberry pi camera's resolution and focal length.

It was observed that a normal laptop could cause multiple GPS errors in the simulation because of the limited computer resources, causing the laptop to not run it properly. Removing the Gazebo GUI and not having the QGC running along would help remove the majority of these problems

Another observation during the use of the simulator is, that it was possible to simulate a "weak" telemetry link. To use Gazebo with the current implementation of MavlinkLora, a link between the UDP ports and COM ports had to be setup. This requires a set of virtual COM ports with a chosen baud rate. Here it was observed, that because of a mistake in setting the baud rate for the virtual ports, the serial connection would loose a random number of packets, but not all of them. Thus, this caused the serial connection to be comparable with a telemetry link with higher packet loss, than a cabled serial connection.

Since this was quite random and could not be readily verified or controlled, no actions was used to investigate this further. The baud rate for this actual case was 57600 and 115200.

2.2.9.1 GCS debug terminal. Prior to testing, a terminal node was created which would allow for data being sent and received by the UAV to be easily monitored and used for bug fixing. Part of this monitor included the creation of a keypress monitor which could be used to inject commands into the system, letting the tester to effectively control the UAV position, attitude and operational mode (pilot/mission handler) with ease.

The GCS terminal proved to be very useful when testing the precision landing system. The terminal keypress monitor was extended with landing pilot authorization later on, which accelerated testing for the tracking capabilities of the Monocular Pose Estimator 2.2.8.5. The tests could observe the influence of different gains on the position regulation in the PID controller whilst verifying that Kalman predictions were accurate.

2.3 Drone Platform and Docking Station

In the project description, it was stated that the blood transportation system must to be able to accommodate for the autonomous loading and unloading of the sample payload, both for ease of use and safety implications for the end user. A Hardware solutions was designed and built and iterated in order to fulfill the project requirements. The payload was described as a box with sides 100mm in length, width and height and is an accurate analog of medical transport boxes. To remove the complexity of mounting the payload, rare-earth neodymium magnets were embedded into the box which could be used for easy snap on/off of the system.

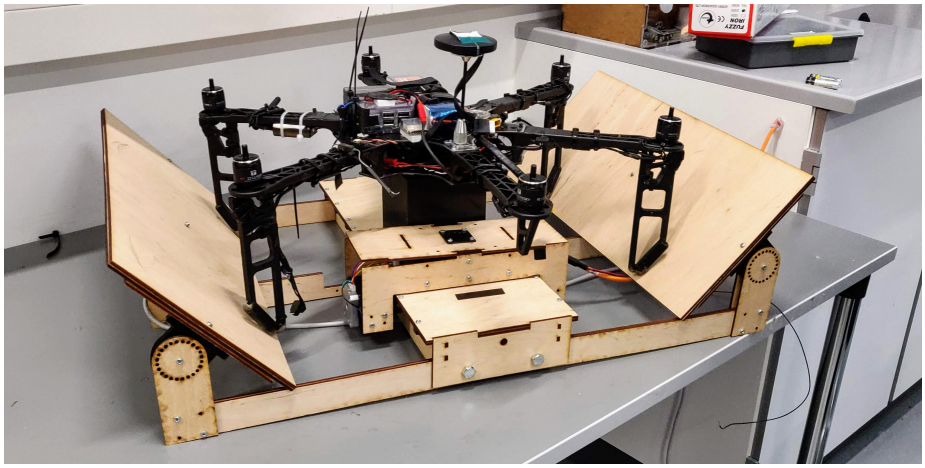


Figure 33: *Final Design and manufacture of the UAV platform and Docking Station*

2.3.1 Docking Station

As part of the requirements, the docking station must be able to precisely land the drone at a specific location, whilst also having the ability remove the payload autonomously. A number of ideas were proposed on how to detachment the payload from the UAV - either through a horizontal "pushing" force or rotational "twisting" force. For further information about these ideas see appendix D.

There were a number of factors that had to be taken into consideration whilst designing the docking station. For example, our chosen method of precision landing meant that a camera would need to be placed centrally on the docking station for optimal results. At the same time, the payload would also need to be placed centrally on the drone to keep it balanced during flight, resulting in a problem that the camera would either be obscured when performing a collection or on landing. As a solution, it was decided that the camera would be placed on an actuated slider, capable of adjusting its position during the landing process.

The design of the Docking station was split up into two parts: the physical embodiment of the dock and the actuation control electronics. The design was developed through the use of Solidworks and a Version Control System called Grabcad [18]

The physical embodiment of the docking station was designed to provide a certain level of flexibility when the drone performs a landing attempt, as it can be assumed that a number of factors such as lighting/weather conditions can cause the drone not to land in exactly the same manner every time. For instance, the sides of the docking station were designed to have a funnel-like shape to guide the UAV into the correct position over the slider gangway. A benefit of using this approach means that the need for a very precise algorithm to be implemented was reduced, as the dock assumes that the stability of the drone and force of gravity will position the UAV in the right place.

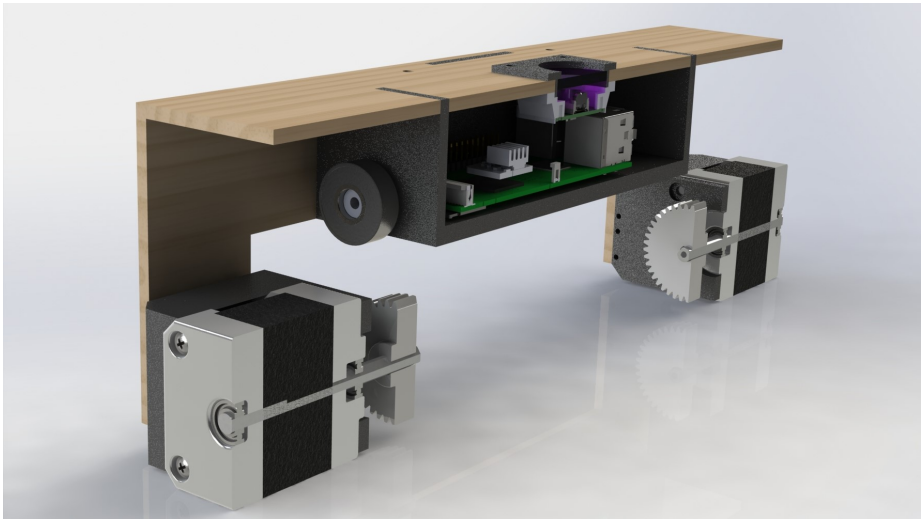


Figure 34: *CAD Rendering of the slider with a Cutaway for internals*

The Slider of the docking station (figure 34) has been designed to house the

[18] `https://grabcad.com/workbench`

electronics and components required for the precision landing system, using a Raspberry Pi 3 with the NoIR camera and 870nm optical band-pass filter to guide the drone into the station - this is explained in more detail in section 2.2.8 The Slider also has the ability to move in a linear motion to aid the precision landing, where it can adjust its field-of-view at lower altitudes (if required). The slider can also perform a payload detachment, where it can gently apply a horizontal force to the blood sample box to detach it from the UAV. The motion of the slider is controlled through the use of a rack and pinion system mounted to the underneath of the gangway, where the pinion gears are attached to two NEMA17 stepper motors for fast and precise motion during operation. In turn, the stepper motors are driven by an A4988 driver IC and are controlled by the General Purpose Input-Output (GPIO) pins placed on the Raspberry pi. To determine its the exact position, the slider had micro switches mounted on one side, adjacent to the stepper motors. When these switches came into contact with the end of the gangway, the origin point could be determined. Like the stepper driver, the micro switches are also connected to the GPIO. The operation of the slider was controlled by a node running locally on the Raspberry Pi, which in turn listens to the '*/topic_ui*' for the loading/unloading commands. When initializing the node, the slider goes through the process of homing and waits until the micro switch is triggered.

Figure 35: *CAD design with the additions made to the UAV platform*

2.3.2 Drone modifications

By default, the SK550 hexacopter frame is not designed to carry a payload, with the landing legs being integrated into the frame arms and leaving little clearance between the base-plate and ground. For the payload to be mounted to the base-plate of the frame, this meant that the drone frame would need to elevated through the use of longer legs. To reduce friction between the docking station arms and the extended legs, 3D printed "bumpers" were attached and

mounted as seen in figure 35. These allowed the legs to run more smoothly during the landing phase. Whilst testing the camera system, it was found that non-black surfaces would reflect IR light whilst the UAV was airborne, particularly in sunny conditions. The Pose estimation detector is fairly robust in compensating for small reflections of light, but false-positives were sometimes detected when large areas of light are reflected into the camera. To resolve this, the drone was dismantled and the frame was covered with black spray paint, with all all wires during reassembly being covered in a black insulation tape to minimize reflections.

2.3.3 Extended antenna

To have a good indication of the datalink between the GCS and the UAV, the Fresnel zones can calculated and observed. The first Fresnel zone was looked at, as it is considered the zone with greatest impact on the communication link. According to Wikipedia[19], it is recommended that no more than 20% of the zone can be obstructed, and a maximum of 40% before serious problems occur. The zones can be calculated by equation 12. The operation is done at fairly close range at 100 and 200 meters set as home and goal.

$$F_1 = \sqrt{\frac{n * \lambda * d_1 * d_2}{d_1 + d_2}} \tag{12}$$

This equation, with the plots in figure 36 and 37 shows that better connection can be obtained by extending the GCS antenna 4 meters up into the air. The antenna was built as shown in figure 38. In an ideal situation, it would have been better to create a taller antenna, but this also would result in being more cumbersome to set up and transport.

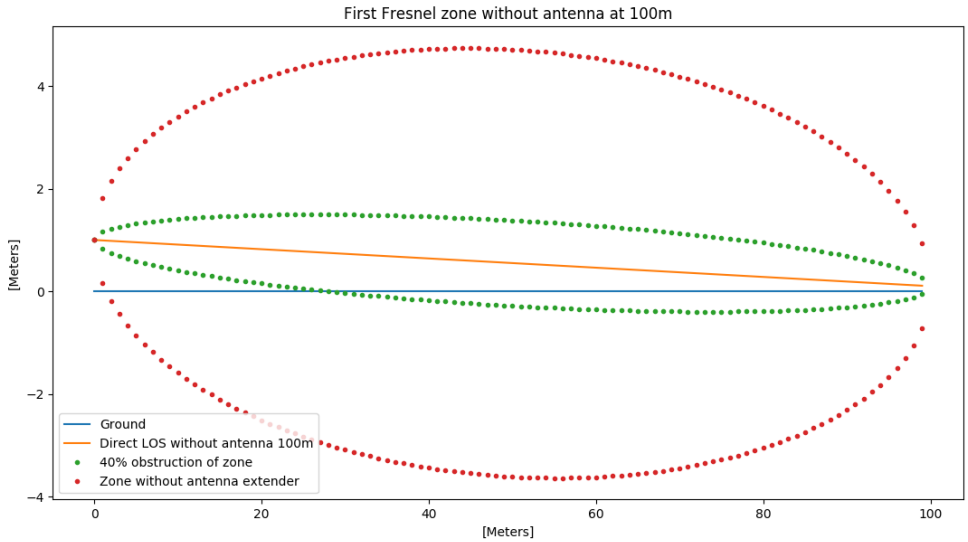
As the team had problems with the telemetry link, the placement of the antenna was tested at different places on the drone. The most successful antenna placement turned out to be below the drone, which was rather surprising but not investigated further.

2.3.4 Radio Link Budget

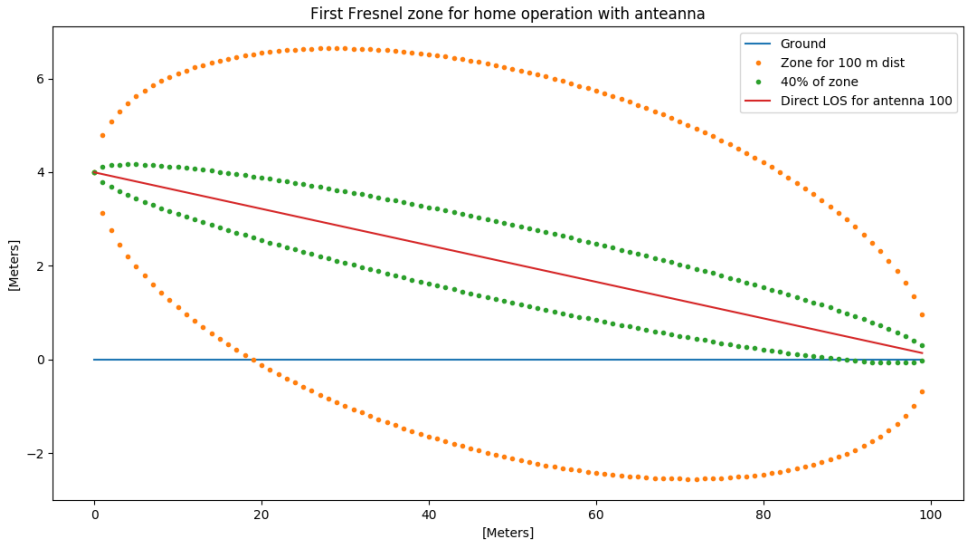
The radio link used in this project consists of two 433 MHz telemetry links called: mRo SiK Telemetry Radio V2 433 Mhz. It is important to make a budget for the project so that it can quickly be assessed whether better hardware is required. The budget can be seen in table 7, and after all gains and losses are taken into account, a margin of 62.7 dB remains for additional noise on the link.

The information used for the radio link budget was found at multiple sources. References can be found in the reference section. The general structure of the budget follows the template in this article [4].

[19] `https://en.wikipedia.org/wiki/Fresnel_zone`

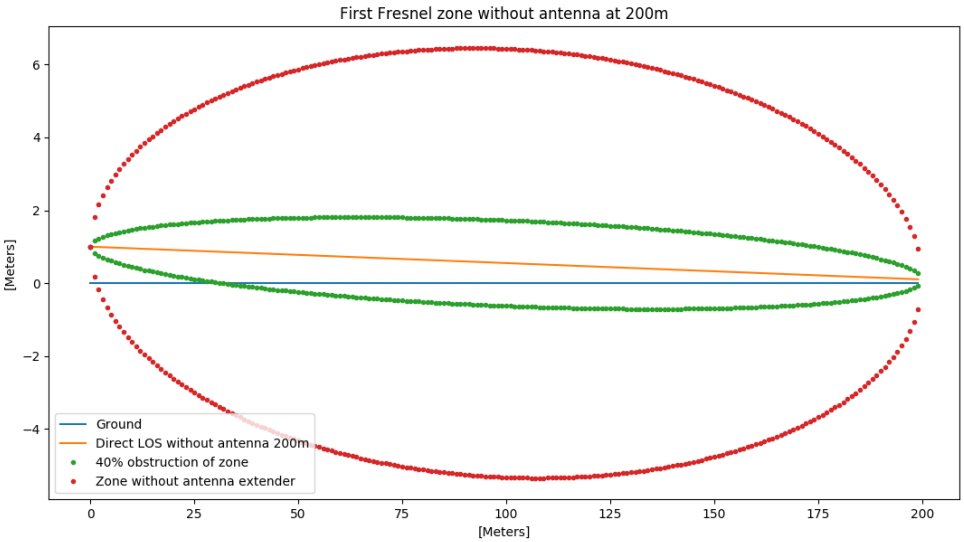


(a) *Fresnel zones at 100 meters out, without an antenna extender. This figure shows that the 40% obstruction is met and retained quite early on.*

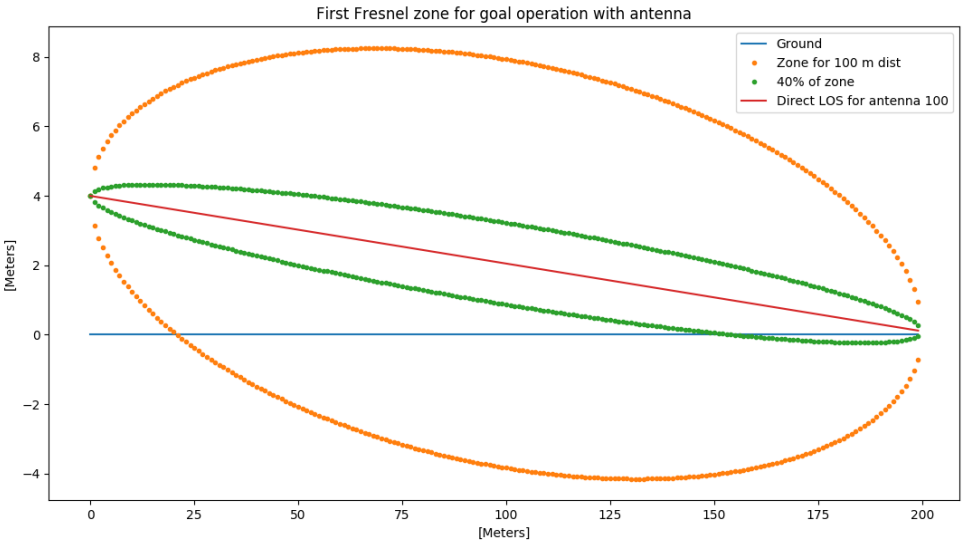


(b) *Fresnel zones at 100 meters out, with an antenna. This figure shows that the 40% obstruction is met only at the end which is marginally better than without the antenna extender.*

Figure 36: *This showcase the difference in the Fresnel zones with having an antenna vs not having an antenna extender.*



(a) *Fresnel zones at 200 meters out, without an antenna extender. This figure shows that the 40% obstruction is met and retained quite early on.*



(b) *Fresnel zones at 200 meters out, with an antenna extender. This figure shows that the 40% obstruction is met only at the end which is marginally better than without the antenna extender.*

Figure 37: *This showcase the difference in the Fresnel zones with having an antenna extender vs not having an antenna extender.*

Figure 38: *Setting up the big antenna for the first time.*

Radio link budget for drone operation		
Transmitter		
Transmitter output power [W]	0.10	
Transmission line loss [dB]	-0.50	2 connectors and 5 feet cable [4]
Antenna gain [dBi]	3.00	[6]
Aperture antenna efficiency	0.70	Source[20]
Transmitter output power [dBm]	**20.00**	
Equivalent Isotropic Radiated Power (EIRP) [dBm]	**20.95**	
Maximum radiated power [W]	**0.12**	
Path loss		
Frequency [MHz]	433	
Distance [m]	200	
Free space attenuation [dB]	**-71.20**	Omitting near field obstacles, fresnel zone, polarization and doppler shift losses
Receiver		
Receiver sensibility [dBm]	-117.00	[5]
Transmission line loss [dB]	-0.50	2 connectors and 0.1 feet cable [4]
Antenna gain [dBi]	3.00	[6]
Aperture antenna efficiency	0.70	
Required signal for minimum sensibility [dBm]	**-112.95**	
Budget		
Equivalent Isotropic Radiated Power (EIRP) [dBm]	20.95	
Free space attenuation [dB]	-71.20	
Required signal for minimum receiver sensibility [dBm]	-112.95	
Margin [dB]	**62.70**	Antenna alignment, cross-polarization, fading, interference, weather etc.

Table 7: *Radio link budget for the mRo SiK Telemetry Radio set.*

The calculated results from the Fresnel zones and from the radio link budget was taken into consideration. Except for minor adjustment of the drone antenna and constructions of an antenna extender, this information was never used directly, because only the provided telemetry module was readily available during this project.

2.4 Project Management

In this project, it was decided to use a modified version of SCRUM, an agile methodology for project management. One of the reasons for choosing an agile approach, is that project delays are very common and sometimes these delays are uncontrollable such as delayed hardware orders in a short-spanned project. Other delays caused by development challenges will also benefit from the agile management, which allows the resources to be reassigned fast to avoid wasting time. The techniques used from Scrum included using a product backlog and Burndown chart, consisting of a list of all considered project tasks - and a chart that shows the accumulated work hours respectively.

The choice of tools for the project management were discussed, taking into account that the project management would need be documented in the report later on. This requirement was vaguely specified in the report requirements, which led the team to initially choose a visually good online tool Trello[21] for organizing the project management. The overall idea with Trello is shown in figure 39. It was also suggested to use a psychical Kanban board, but due to the fact that the group work would move around between the airport facilities and campus, this was not considered a good solution.

[21] `Trello.com`

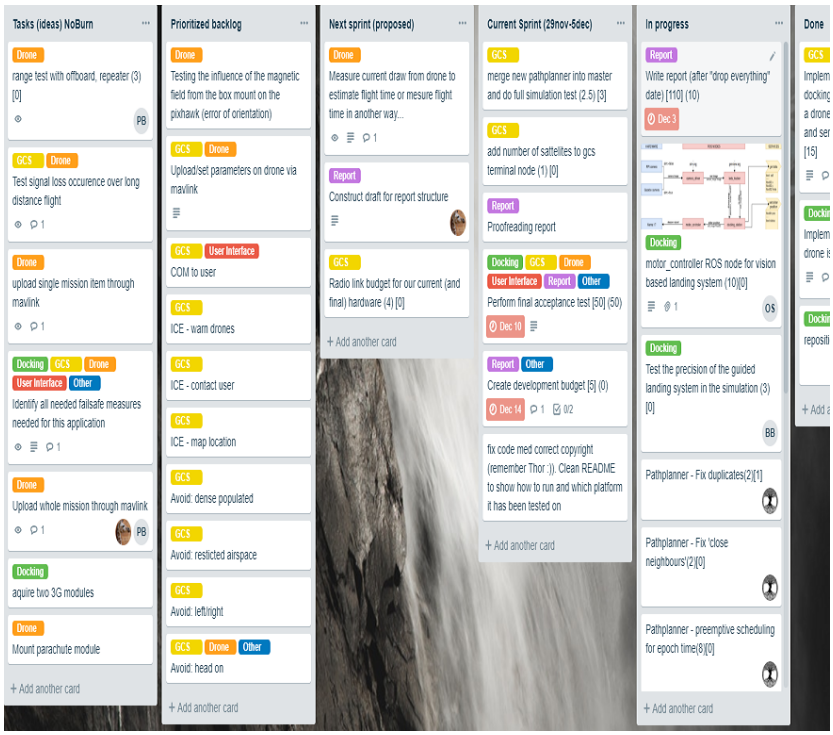


Figure 39: *The actual text is not important in this image, as it purely serves to show the organization in the Trello tool. The board is organized in such way, that 6 lists is the main part of the board. The lists are named Product Baclog and Current Sprint amongst others. Each list will have several cards and these cards represent individual tasks.*

Individual tasks are represented as cards on the Trello board lists. An example of a card is shown in Fig. 40. These cards can hold all the relevant information in a smart way and will allow to look back at tasks, when the report is due.

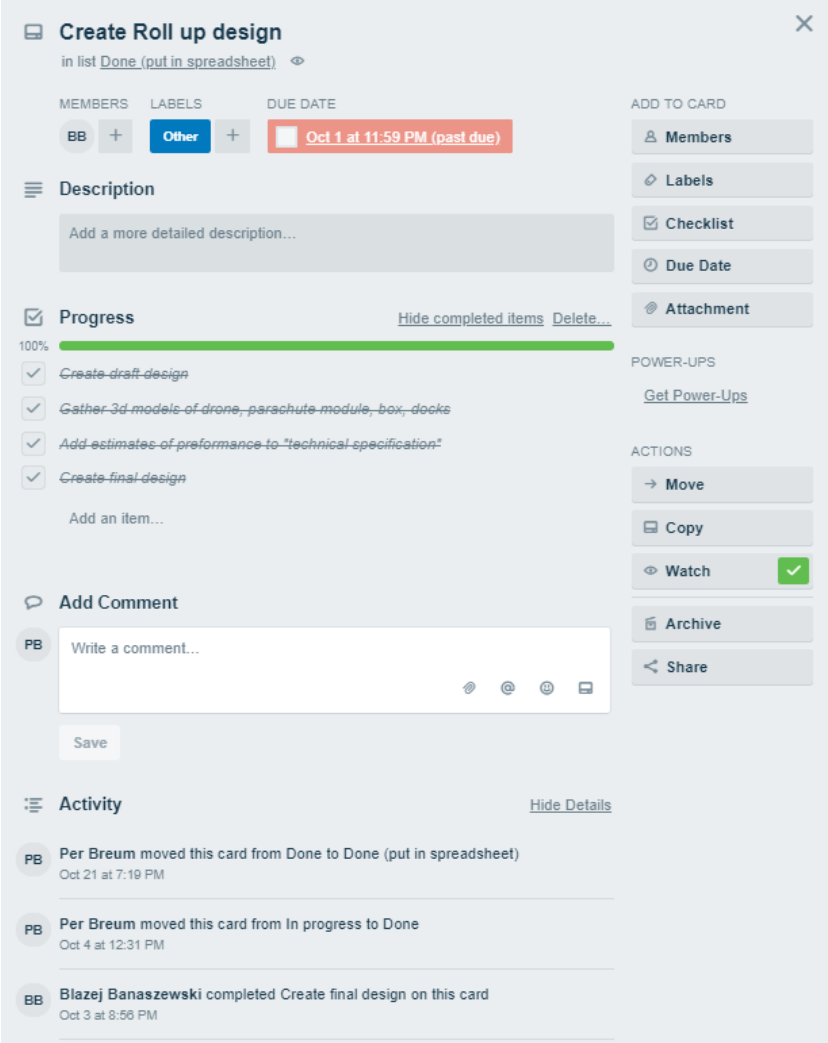


Figure 40: *The card would hold all the necessary information, required for every individual task. It is possible to add checklists and monitor its progress, in addition to adding descriptions and viewing a card/task around is history. Other things, assigning people to a task and setting deadline dates is also possible.*

Some weeks into the project, the stakeholders stressed that there only one interpretation of the project specification existed, effectively voiding the intended use of the selected project management tool for the report. This led to a discussion on how to alter the project management in a way that would fit the new requirements without losing the flexibility that Trello provided. The solution involved continuing to use Trello for the daily management, while time was used to duplicate all tasks into a spreadsheet and maintaining this in parallel. This spreadsheet is depicted in appendix F. The spreadsheet was intended to hold additional task descriptions, but this was not prioritized since this information was readily available at Trello for the developers.

The additional part of Trello is the use of plugins, for which a Burndown

chart plugin was used, and is depicted in figure 41. Each team member was responsible for logging their own hours, and this was often forgotten, which resulted in larger lumps of work hours at the same time, since Trello could only update from the date, which it was typed in.

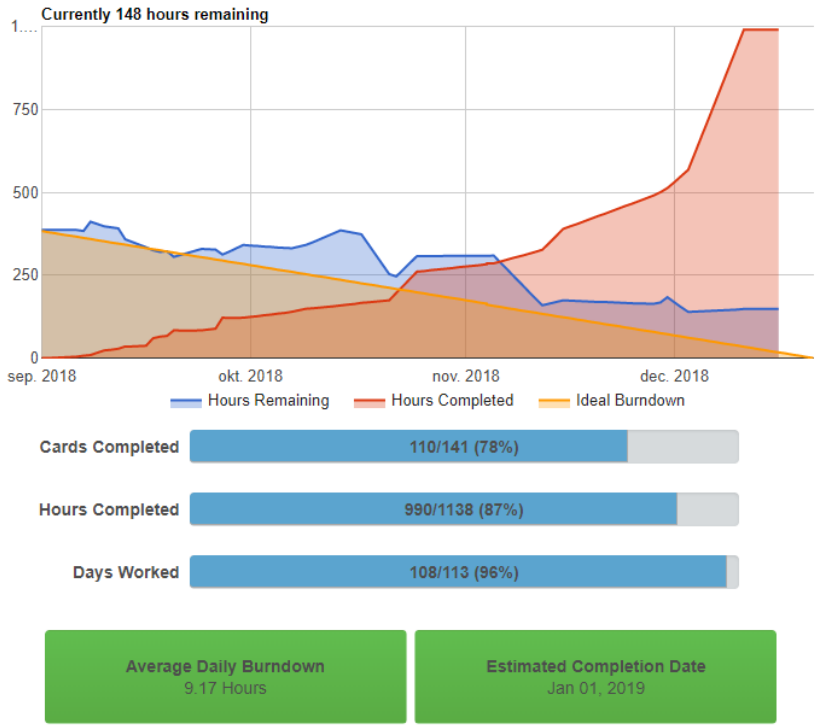


Figure 41: *The burndown chart used in this project. With every task all team members would specify an expected time for the task and continuously add the hours already worked on the individual task. The number of expected hours was approx. 400 according to the chart and the actual worked hours was approx. 1000. The ideal burndown with the expected time, and current tasks, would have the project finalized the 1st of January 2019. It is notable to see how the amount of work hours increased rapidly in the last part of the project. The obvious reason is of course, that more hours was actually put into the project in the final part, but another reason is the maintenance of logging hours are most times not prioritized or simply forgotten.*

Using the burndown chart might seem obscure or irrelevant, but it is considered a good practice to enforce the logging work hours on separate tasks - since it is an expected requirement in a normal developer job. Another significant fact, is that it was agreed to log work hours to use for the development budget.

Because of the lumps of added hours, especially in the end the curve does not reflect the true progress. Moreover, the expected curve would still look similar to this - as previous project experience shows that increased hours of work are often performed just before deadline. The project manager is responsible for enforcing the practice with logging the work hours, but one of the challenges

in the project work is limited to a few days every week. Sometimes individual group members would shift the workload between weeks and made it difficult to track without daily feedback on the work process. Notwithstanding, a more experienced project manager would possibly add to a higher success of using these tools. In hindsight the Burnout chart is still a good tool to use because of the ability visualise progress.

In the beginning of the project, the amount of expected work hours was specified in the group contract to be 16 hours on average per week. this can seen in appendix B. A value of 16 was decided after hearing a briefing from supervisors towards the beginning in the project. However the actual number might be a considered a bit higher, since the course has a 0.167 full-time equivalent, which gives approx. 18 hours per week, under the assumption that the year have 1924 work hours and the project spans 4 month. The number from the report specification states 22.5 hours pr. week, but this is, obviously, highly depended on the method for calculating this.

Using the initial agreed of 16-hour expectation and the calculated 18 hours, this difference of 2 hours every week will add up to approx 200 work hours. Despite the discrepancy, it was not taken into account when the group contract was made. The workload was high in the beginning, although this is not perfectly reflected on the burndown chart, and it is believed, that the last work hours was obtained in the end of the project with the relatively high work load.

A final reflection on this; no matter which method is used for calculating the amount of work hours, the total number of accumulated hours and expected completely from the real numbers. This emphasizes the need for more experience for all team members in the way of estimating and logging hours.

The assumptions made in the group regarding work hours, rules and regulations within the project group was decided upon the first meeting together and cemented into a contract which is seen in appendix B.

2.4.1 Development Budget

A budget has been constructed for the development performed in the current project period. The budget is divided into two parts, the materials costs are depicted in figure 42 with the administrative costs and summarized development costs, in figure 43.

As the previous section stated, logging the time used was difficult and the number of hours used for the budget is the nominal hours for 10 ECTS workload for each person in the group.

Material costs

Description	Additional information/Comments	Amount	Price pr. item	Subtotal DKK
Visual guided landing subsystem				
RPi NoIR camera	Visual guading landing system	2	144,00	288,00
Mini BEC V2 Voltage Regulator (3.3V,5V,12V)	Powering LEDs from the battery on the drone	1	16,80	16,80
IR LED 870nm 780mW	LED system on the drone	10	9,60	96,00
Narrow band 883nm filter	Filtering out everything but light from LEDs on the drone	2	249,60	499,20
Docking Station				
Nema 17 stepper motor	Attaching and disattaching a package from the drone	4	111,20	444,80
A4988 stepper driver	Driving stepper motors	5	36,00	180,00
BEC 5V out, 12 V in	For powering the RPis in docking stations from 12v input	2	88,00	176,00
bearings 10mm 3mm	For the slider pushing the load out or in the drone's magnets	4	14,40	57,60
GroundControlStation				
Strømforsyning for Raspberry Pi	Powering the VPN RPi	1	47,20	47,20
GSM module	Comunnication between GCS and docking stations	2	268,00	536,00
Raspberry Pi B 3+	VPN server	1	219,20	219,20
Droneplatform				
Frame	(price for S550 listing used)	1	320,00	320,00
Batteries	5800 mAH	4	464,00	1.856,00
Battery charger	4S compatible	1	200,00	200,00
Pixhawk/GPS combo	2.1 model	1	2.240,00	2.240,00
Spektrum receiver	SPM4648	1	180,00	180,00
DX6 transmitter	G3 model	1	1.200,00	1.200,00
Telemetry kit	mRO kit	1	304,00	304,00
Motor/ESC combo	T-Motor Air Gear 350 Combo	2	624,00	1.248,00
3D printing material	(assume it is not needed to buy 3D printer)	1	200,00	200,00
Raspberry Pi 3		1	250,00	250,00
Additional				
Additional	Added 20 % for budget overhead to buy e.g cables or for repairs	1	1.931,76	1.931,76
Subtotal				12.490,56

Figure 42: *The complete costs of materials used during this project. Note: minor items (e.g screws and heatshrink) have been omitted and included in a budget overhead.*

Adminstrative costs

Description	Additional information/Comments	Hours	Hour rate	Subtotal DKK
Test facilities				
Test area and lab access	Estimated hourly rate	30	1.200,00	36.000,00
Developers				
Per Breum		321	250,00	80.250,00
Oscar Schofield		321	250,00	80.250,00
Błażej Banaszewski		321	250,00	80.250,00
Thor Gunnlaugsson Jensen		321	250,00	80.250,00
Mathais Witten Madsen		321	250,00	80.250,00
Mads Tilgaard Jensen		321	250,00	80.250,00
External Consultancy				
Kjeld Jensen		25	275,00	6.875,00
Tobias Lundby		25	275,00	6.875,00
Vincent Klyverts Tofterup		25	275,00	6.875,00
Kristian Husum Terkildsen		25	275,00	6.875,00
Subtotal				545.000,00

Total costs

Description	Additional information/Comments		Subtotal DKK	% of total budget
Materials			12.490,56	2,2%
Administrative costs			545.000,00	97,8%
Total DKK				**557.490,56**

Figure 43: *The hours noted for the external consultancy and costs of test facilities is rough estimates, since they are only considered a minor expense in the total budget for the current development.*

This development budget reflects the current stage of the development. *Technology Readiness Levels* (TRL) can be used to put the current development into a larger perspective. The TRL level consist of 9 levels with the development stages going from basic principles to the full implemented and proven system into the intended operational environment.

If the interpretation of the TRL from here[22] is used, then this project has reached level 4. The reason for this level is because the project is considered beyond a proof-of-concept, but is still residing in the small prototype state with the need for more validation and development of the subsystems inside the lab.

To state a concrete number for the final budget, if it was desired to continue from this point on and reach TRL 9, is very hard with the lack of financial experience within the team, in such scale. Completing the final level requires a fully consumer-ready product, which is believed to require a long-running large scale test of multiple drone system running in parallel, to obtain the permission for autonomous flight all over Denmark. The hardware costs in this case would be a wild guess, as the drones used in this case would be much more expensive, as they will require more advanced functionality, like collision avoidance system.

With this in mind, a guess for reaching a TRL 8, which is the level for a complete and qualified system, would be the current budget with a factor of 20. This would give a development budget of approx. 11 mio DKK for reaching level 8.

Earlier in section 1 it was stated, that on Funen alone the costs for blood transport was 1.5 mio DKK, so there is potential for creating a product, which could be a good investment for the society. It is emphasized, that this comes

[22] `https://serkanbolat.com/2014/11/03/technology-readiness-level-trl-math-for-innovative-smes/`

with a reservation, that there is a high uncertainty for the price of the full completion of the system, as the technologies for actually achieving this is still being developed, such as the UAS traffic management.

2.5 Specific Operations Risk Assesment (SORA)

UAV operations are divided into 3 categories:

- **Open:** Operations that follows the general law about flight and does not need special permission.
- **Specific:** An operation that needs permission from the authorities to do a very well defined task.
- **Certified:** Is used when the risk becomes more severe, akin to a commercial aircraft, normally multiple certifications needs to be provided.

The SORA is a document that is used to help companies that wish to conduct an operation within the specific class, and the authorities to monitor any active drone operations.

The assignment given is native to the specific category, and thus a SORA documenting the mission and organization behind the operation needs to be conducted. The SORA has 9 steps which need to be conducted according to JARUS-WG-6 guidelines. These steps includes a complete description of the whole operation, ground and air risk classification, which results in a SAIL (specific insurance and integrity level). It also must specify ways to mitigate the possible risks.

The complete SORA can be found under appendix H and the key elements are described here. First Annex A from JARUS has been followed in describing the operators, testing conditions, the complete UAS, and the ground support equipment.

The initial ground risk is found to be 3 as the kinematic energy at terminal velocity is 1190J, this risk is then reduced as a parachute module is to be installed on the UAV before flight.

An air risk is also derived from a flowchart provided by JARUS. As the airspace is restricted this puts the operation into the 'a' class which is the safest.

A SAIL is then determined from the air and ground risk. Considering these the final SAIL value is 1 and the Operational Safety Objectives (OSO) can now be looked at. These are divided into Technical issues, Deterioration of external systems, Human errors and Adverse operating conditions. Under each subject, multiple points has to be taken into account, making the operation more robust and safe.

2.6 Rollup

The design of the rollup has been done primarily in Adobe Photoshop. Adobe Illustrator has been used for the vector graphics and minor fonts adjustments, and SolidWorks CAD software for the 3D modeling. The final design has been presented on figure 44.

2.6.1 Design considerations

The target audience for the project concept are the engineers, but also people lacking the relevant technical knowledge - such as potential investors. Therefore, the wording and terminology had to be adjusted to be comprehensible for all of the targeted groups.

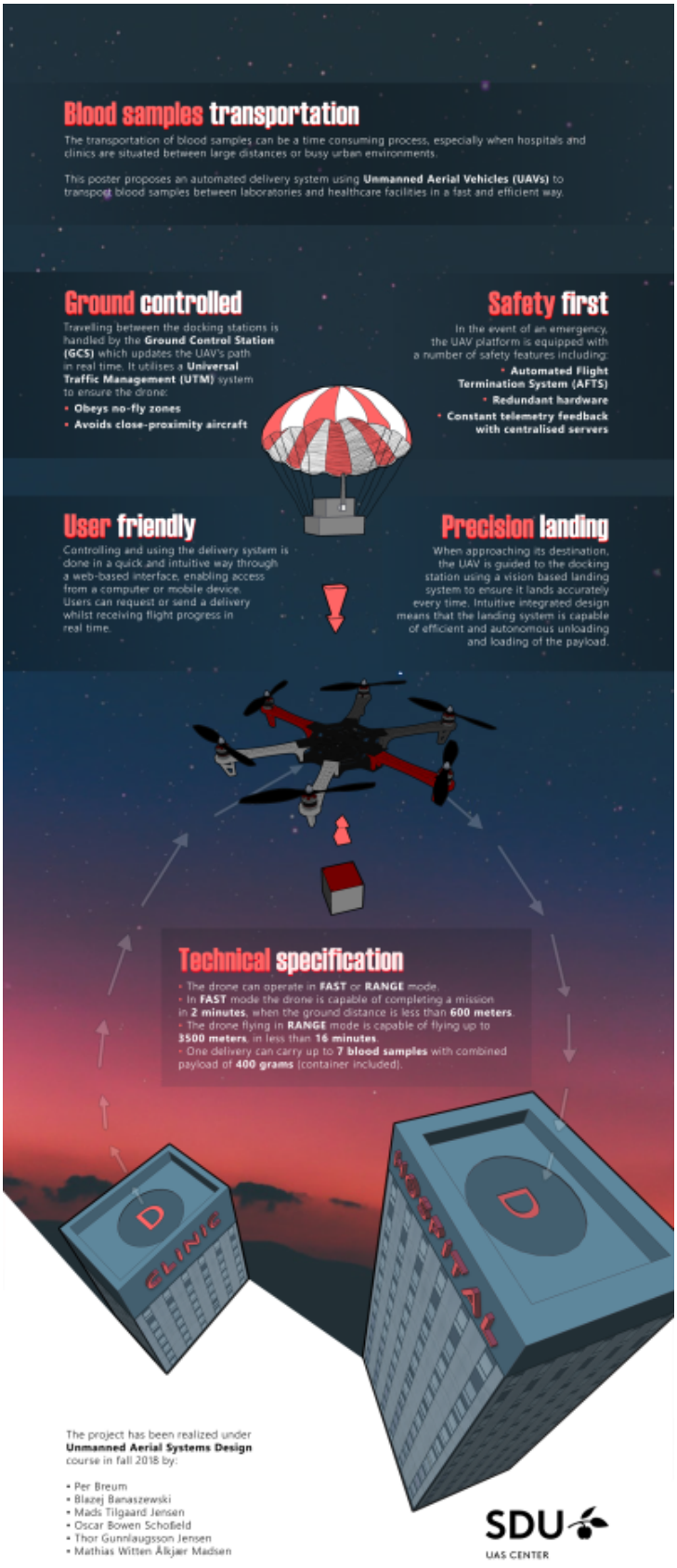


Figure 44: *The final version of the rollup*

3 Tests and Results

This section will include a description on how the tests were performed in this project, along with checklist and procedures for the real life test. The checklist used when testing can be seen in appendix H, page 14.

Since this project aims to be agile, a testing list was not developed with the required tests and accompanying deadlines. The Product Backlog and the current tasks was prioritized in a way to fulfill the desired goals for the project and when a task/subsystem was completed or ready to be tested, this would be done at that point. This allows for a highly agile test planning. The goal is for each subsystem to first be tested to an acceptable working level in simulation, wherever possible. Shortly hereafter, a real life test would be conducted to check if everything operated as expected in a less perfect environment.

During the whole project, all flight data is logged by the internal Pixhawk system and these logs can be retrieved for inspection, after each flight if necessary. All flights are also logged with timestamp, flight time, drone operator, location and notes about the flight. This is typed into a spreadsheet and can viewed in appendix A. When there was more than one test or multiple observations in a single day, the flight report was summarized in a document on Google drive and linked in the flight notes.

As the project is based on ROS, it is also possible to log the complete ROS system by recording a rosbag. A specific strategy for when to record was not agreed upon, but when the test operator deemed it valuable it was expected to be done.

3.1 UTM test

The testing of the UTMS handler was minimal, as this was mainly grabbing data from different parts of the system and collating it within a single place. A number of tests however were outlined in the testing plan (appendix E) to ensure that the handler would behave as expected during operation. These tests included:

- Sending UAV Position
- Receiving UAV Positions
- Receiving Dynamic NFZ

To avoid endangering the UAV or others during these tests, the team were provided with the details for a test ID within the UTMS. This test ID would appear in the responses for tracking data, but would be ignored by the visual front-end website.

3.1.1 Sending Data

The main focus of this test is to ensure that the data was packaged correctly for the UTMS to accept it, as the system would reject the data and throw an error status if it was incorrect. During simulated operation, it was important to see that the data being sent to the UTMS accurately represented the UAV's current status, in particular its current operational status and GPS data. Figure 45 shows a verbose response to sending tracking data, with the drones information being shown on the UTMS visual website.

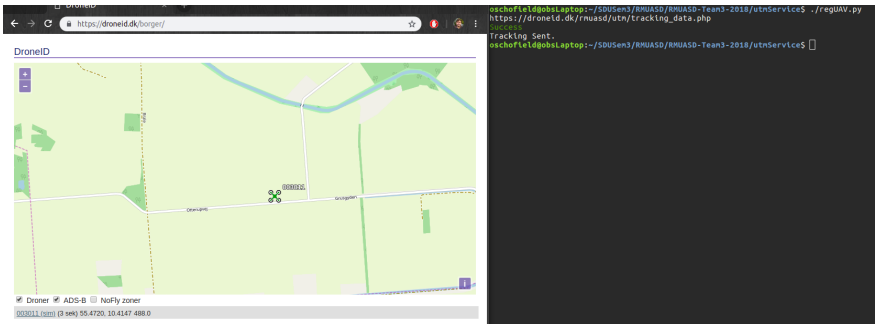


Figure 45: *UTMS Confirmation and Website Visualisation*

3.1.2 Receiving Data

The process of receiving tracking data is split up into two sections: Firstly, the process of requesting tracking data, then parsing it and filtering wherever necessary. Because of this, the tests were more focused on the parsing and filtering of data, where the complexities in coding lie. This approach applied to both the UAV tracking data and the dynamic NFZ.

The tracking data for traffic was checked through visually displaying the results to the command line. Figure 46 shows that the data was coming through as expected. This was then filtered down to only show the most recent tracking data sent to the UTMS, and was confirmed by comparing the size of the list pre/post-filtering.

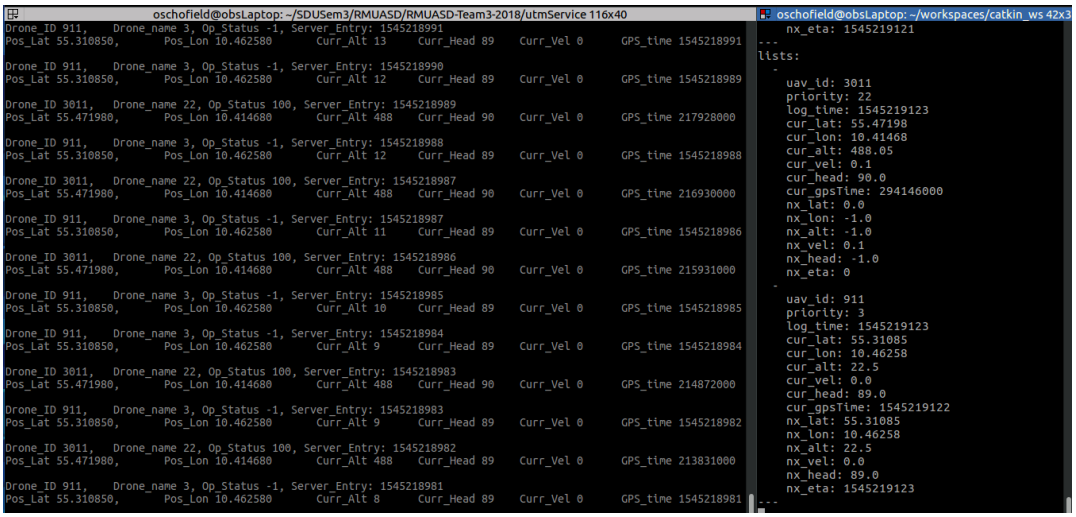


Figure 46: *(left) Unfiltered Data (right) The filtered list of recent updates.*

The next step was to check the traffic was being published to the */Proximity_Traffic* topic whenever there was traffic within the proximity radius of the UAV. This was done by using the test id to simulate a drone performing a square holding pattern near our UAV, where the pattern crosses the proximity boundary. The Result of this can be seen in 47.

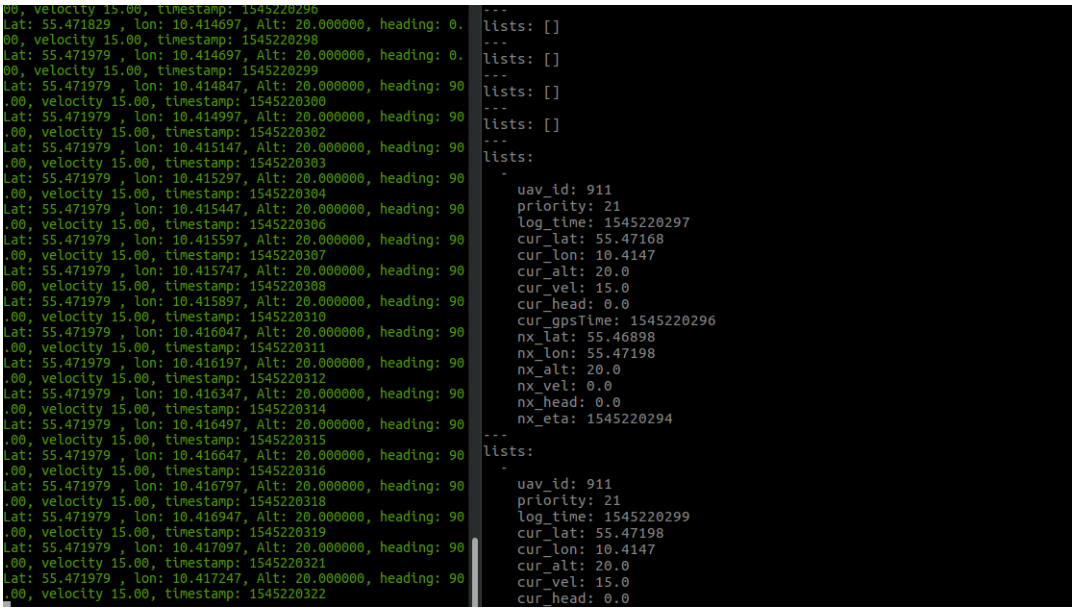


Figure 47: *(left) Simulated traffic flying in a square pattern, (right) simulated traffic crossing the proximity boundary of the UAV.*

Additionally, the critical alarm was tested by statically positioning the test ID drone adjacent to the UAV, which caused the critical alarm to be published over the */state/close_encounter* topic.

3.1.3 Dynamic NFZ Parsing

The data for the Dynamic NFZ was parsed in a different way compared to tracking, and this was down to the structuring and intended use of the data. The testing for the parser was initially undertaken by only considering circular NFZ, as they are structured in a simpler form compared to their polygon counterparts. The parser was then checked by visually inspecting the compiled list, as shown in figure 48.

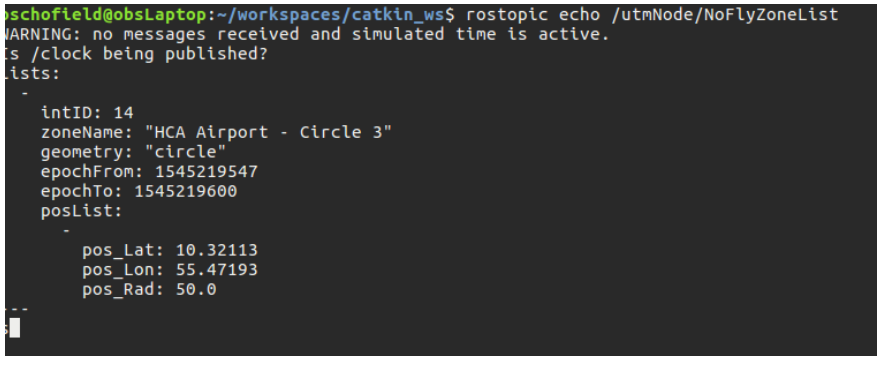


Figure 48: *List showing the UTMS Dynamic NFZ*

3.1.4 Error and Exception Handling

Whilst the previous tests focused more on the data handling and processing, it was important to ensure that the UTMS handler could deal with issues in communication and exceptions. These are primarily split up into two kinds: HTTP exceptions and Requests exceptions, where the former occurs when the get/send/post request is successful in communicating with server, but receives an unexpected response in return (such as a 404 error). The Requests exceptions occur when the UTMS handler tries to communicate with the server, but due to an unstable internet connection is unable to complete the response within a specified time frame.

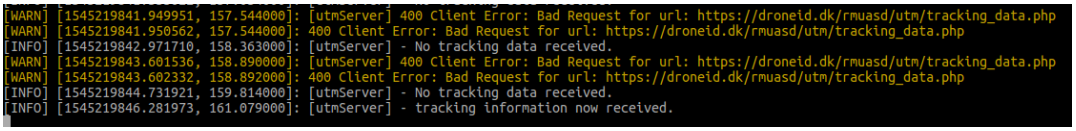


Figure 49: *Screenshot showing a HTTP error, where user is notified after error is resolved.*

During Simulation testing it was easy to debug HTTP exceptions, as these typically occurred when the system was trying to send or receive data before the nodes had fully initialised. For instance, figure 49 shows an error when the handler tries to receive tracking data from the UTMS, but no tracking data is available to be received. Depending on the severity of the exception, the handler will choose to notify the user using the ROS severity guidelines.[23]

Figure 50: *Screenshot showing the UTMS handler recovering after a timeout*

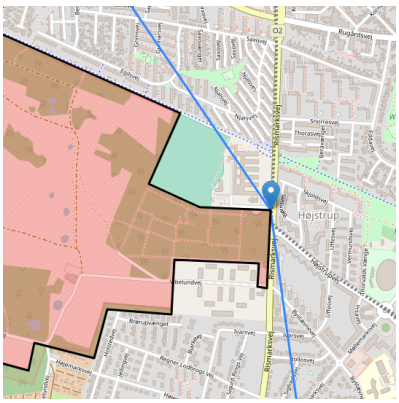
The Requests exceptions were a little harder to test, as these did not appear in simulation but during field tests. This could be caused by a lack of stable network connection because the most common exceptions that were raised were Request timeouts. To tackle the problem, the python try/except frameworks were incorporated, with the ability to retry a request three times, after which the UTMS handler would warn the user. Additionally, the system would notify if a retry was successful and the connection was restored. After implementation, it was found that the system would only need a single retry before the connection was restored and the request sent successfully, as show in figure 50.

3.2 Path planner test

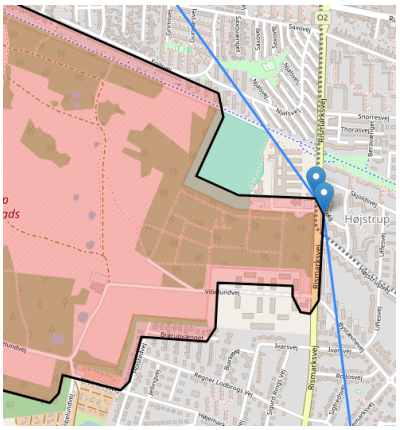
The pathplanner was first tested in simulation with predefined start and end destinations found from `www.google.com/maps` at various locations.

[23] `http://wiki.ros.org/Verbosity%20Levels`

This test here is to verify that the pathplanner can buffer the NFZs and plan around them. The regular NFZ is compared to the buffered NFZ. One NFZ before and after the buffering can be seen in figure 51.



(a) *This figure shows the static NFZ around "Hoejrup oeveplads". Note that the zone has not been buffed.*



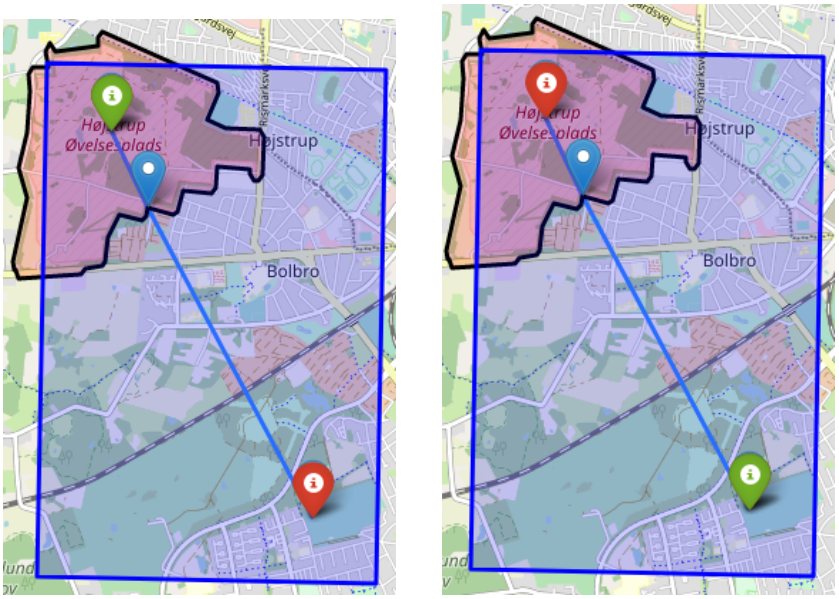
(b) *This figure show the static NFZ around "Hoejrup oeveplads". Note that the zone has been enlarged by 50m and the edges have been rounded.*

Figure 51: *The figures above shows how the buffer function enlarges the polygons that constitute the NFZ. The value for enlarging the NFZ has was chosen to be 50m.*

There were three major special cases which the pathplanner needed to deal with:

- Takeoff in NFZ
- Landing in NFZ
- Suddenly being in a dynamic NFZ

The pathplanner were tested with the takeoff in a NFZ to see whether the system could plan a route without having the drone executing it. The route planned can be seen in figure 52a. The pathplanner also had to account for having the desired landing site in a NFZ which can be seen in figure 52b.

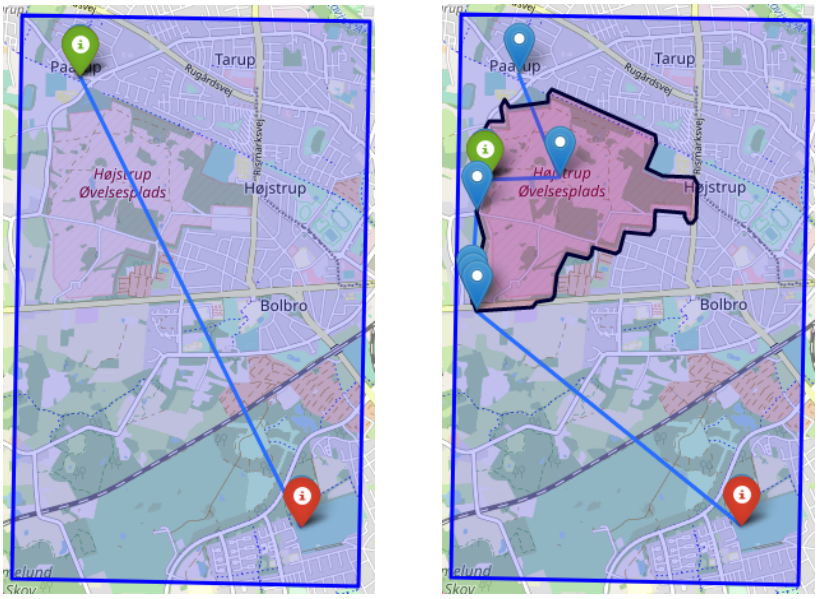


(a) This figure show that the starting position is located inside the NFZ.

(b) This figure show that the end location is inside the NFZ.

Figure 52: *The figures show two of the more special cases as landing and taking off in a NFZ. The figures uses the same coordinates but the end and start position was switched thus the same "entry" point in the NFZ.*

Another special case for the drone is to suddenly find itself inside a dynamic NFZ. This is shown in figure 53b. The top green marker is the initial start point and the red marker is the desired destination. The green marker inside the red marked area is the current position of the drone. Now the path planner has to exit the NFZ with the shortest possible route and then proceed to plan the shortest possible path to the destination from the point on the boundary of the NFZ.



(a) *This figures show the path planner planning from start to end with no NFZ taken into account.*

(b) *This figures show the same path as in figure 53a but with a sudden dynamic NFZ appearing.*

Figure 53: *The figures shows how the path planner planning a direct route and how it recalculates to the nearest point outside the NFZ when it suddenly appeared where the drone is.*

3.3 Offboard mode

The process of testing Offboard mode lasted throughout the project. Initially the results were promising, at least in shorter flight tests. In the development of the Drone Command Handler, described in section 2.2.7, the subsystem was tested both in simulation and in real life multiple times.

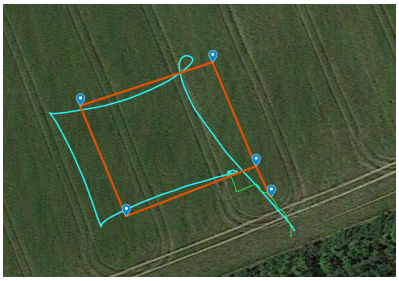
An overview of the different tests are provided in the following section with important observations. The dates used in the section names are for easier time line reference.

3.3.1 Initial tests

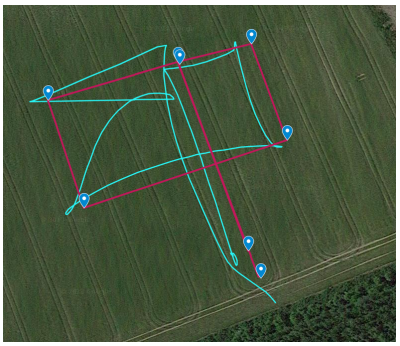
Initial tests outside simulation is depicted in figure 54. Simulation tests are omitted, as they showed expected flight pattern and behaviour, which was a requirement before advancing to real life tests.

These initial tests was performed using a minimum of the subsystems and using a test script to upload mission, based on a GCS mission plan as input. The fallback for loosing the Offboard mode is Position Hold for all tests.

This test was performed ultimo October, where the project had been running for approx. 2 months. The subsystem for performing this test had been ready for 2 weeks, but because of the Autumn break and the wish for pooling more tests together, it was tested for the first time late October.



(a) Small square flight test. The full test mission was completed



(b) Longer double square test. A GPS error was encountered, causing the drone to appear in another place than it actually was. The full test mission was completed

(c) Repeated test from figure 54b without showing waypoints. The mission was terminated just before it was completed. This is where the "no offboard" error was observed the first time.

Figure 54: *The red lines show the direct lines between WPs and the blue shows the travelled path in Offboard mode.*

The missions in figure 54 is performed with the antenna placed on the table and a maximum distance between the telemetry of 100 meters. The Offboard mode was lost several times during the last (just before it terminated) flight in figure 54c. Although the flight tests in 54a and 54b was performed without offboard problems. It was clear at this point, that this type of test must be repeated again several times. The changes considered for the next Offboard test was considered to be raising the setpoint publish rate. The GPS (internal EKF) error observed in figure 54b was not considered a problem at this point, but noted down in case of similar future occurrences.

3.3.2 7th November

This test is performed after the GPS problems was fixed. These problems is further described in section 3.6.

Two flights were conducted this day. The GCS antenna was placed 1 meter above the ground on a cardboard box for both flights. The setpoint rate for the drone was 20 Hz for this test.

The first test was operated in 40 meters altitude and the drone would stop and hold the position at the last waypoint. The same mission was repeatedly uploaded to the drone 5 times and the flight path is shown in figure 55.

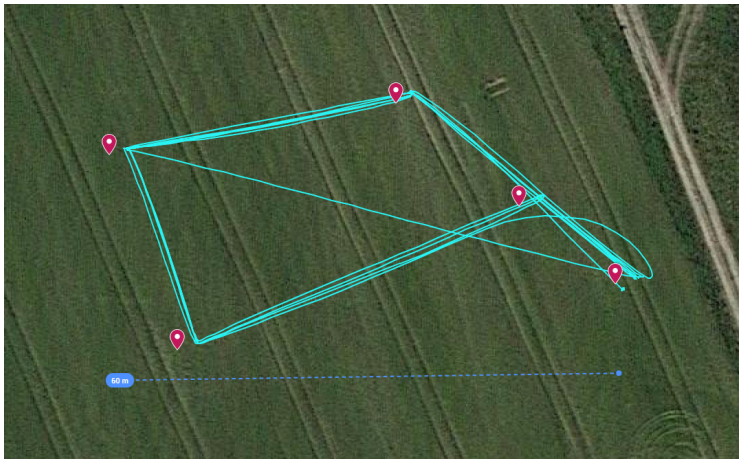


Figure 55: *The dotted blue line show 60 meter reference. The light blue show the repeated Offboard mode flight pattern. During these repeated flights, the drone is not casting a single Offboard error.*

Second flight that day was an attempt to challenge the telemetry with an increased distance. It has been observed in previous test, that increasing the distance could cause Offboard mode to go into failsafe mode.

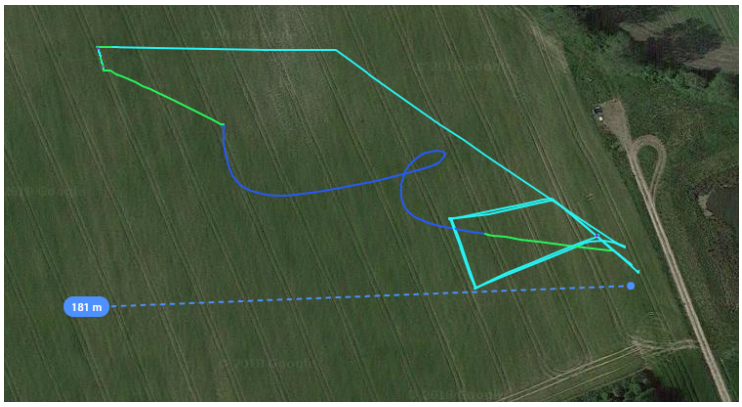


Figure 56: *The dotted line is 180 meter reference. This flight was a combination of a longer path and the path from figure 55. First the short path was conducted and hereafter the long path was attempted. This was, however not a success, and the drone was manually flown back to the starting point. Then the short mission was repeated again, to see if that still worked. The drone started loosing Offboard mode in the top left corner and went into Position Hold, which is marked with green color. The dark-blue color is manual flight.*

The second test pushed the limit with a longer distance and the drone went into failsafe Position Hold. This indicated that the telemetry might be causing the problems with the Offboard mode. The following improvements was considered; **1)** Create a higher ground antenna. **2)** Obtain a better antenna (patch antenna is suggested from the telemetry website). **3)** Try to improve the telemetry link by changing the settings on the radio kits. At this point several patch antennas were investigated and a longer antenna holder was constructed. In regards to improvement for parameters on the radio kits, there was reach-out for help from the supervisors but they were not aware of any concrete solutions to improve the settings at this time.

The team had a discussion about the parameters and it was suggested to try changing selected values and perform more tests, but increasing the antenna height first was prioritized.

Another observations during the last of the long tests, was that the RC signal was lost shortly after the Offboard mode went into failsafe. This could be because of the preliminary positioning of the RC receiver on the drone. Since the receiver is a diversity module, the antennas was placed in 90 degrees in relation to each other to improve the connection.

3.3.3 12th November

The testing done this day, was conducted in the model airfield club near the HCA airport. A new antenna extender had been constructed at this point and was ready for testing. Two flight test were performed, the first was a short route with a repeated pattern and the second test was a longer route. The two test are shown in figure 57 and 58. Both flights was operated in 20 meters altitude to stress test the system a bit more, as the signal should be worse the closer the drone were to the ground. A range test was performed to check what the message throughput was from the drone before the testing done this day.

The result of this test was, that approximately 95% of the expected messages was received at the drone - although the latency could cause problems for using Offboard. The range test is described in section 3.5.

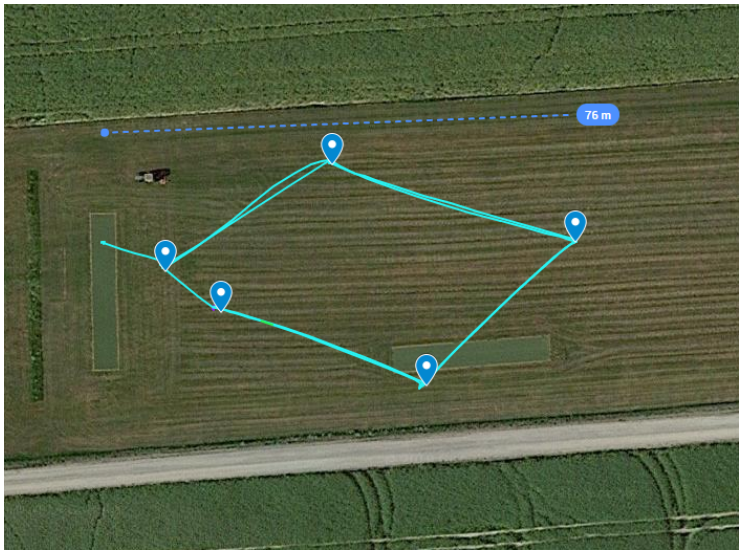


Figure 57: *The dotted line is a 75 meter reference. The flight pattern here is repeated two times around the clock. No Offboard problems observed in this test.*

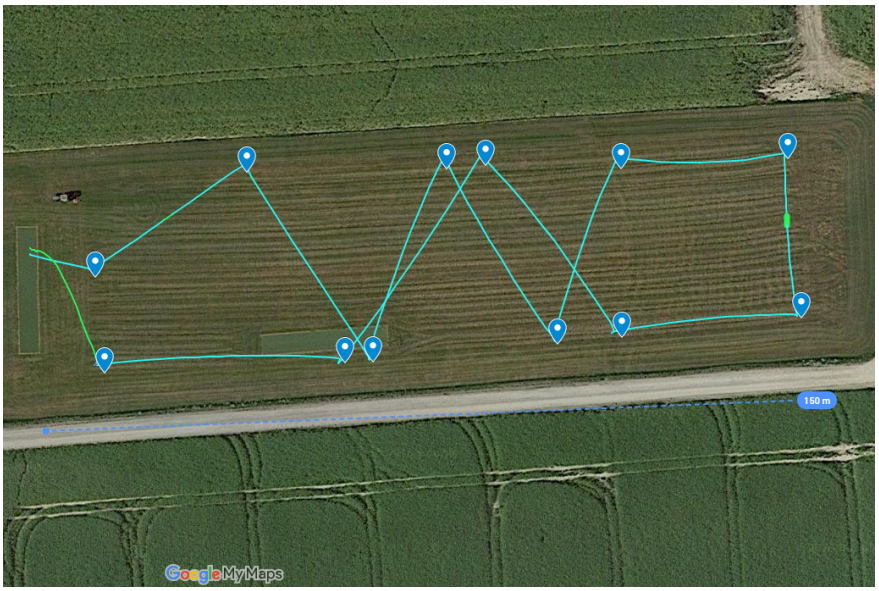


Figure 58: *The dotted line here is a 150 meter reference. Only once during this longer flight was the Offboard lost. This is visible in the right side of the image, where the green colored line has been increased. The green line in the left side of the image is from when the drone was manually brought back by the operator.*

The test done this day was proved promising for the use of the antenna extender, as there was very little visual observations from the drone stopping up in the far end. During the second test, the flight log showed Offboard failsafe 5 times during the whole flight, where most of these were in close proximity from the GCS antenna.

These results indicates, that the problems with Offboard is hard to get rid off. As described earlier, the drone must receive the setpoints fast meaning that latency can be influencing the momentary loss of Offboard. At this point the RPi repeater, described in section 2.2.7.5 was considered and implemented shortly after.

Another considerations about the weak telemetry link, is that during the drone flight, the drones is posing in a vast number of ways. The antennas on the telemetry link are linear polarized and certain positions of the drone can orient the drone antenna, in the worst case scenario, with 90 degrees shift between the GCS and drone antenna. This would in theory mean, that nothing is received from the transmitter.

During these tests, it was observed, that other groups were complaining about interference, because their drones were almost impossible to communicate with, when all the telemetry modules were transmitting.

The groups must from this point on agree upon the used (and allowed) frequency band, to avoid as much interference as possible.

3.3.4 26th November

This test was a very short test. The test consisted of trying the Offboard mode with the new RPi repeater onboard. Several subsystems was included in this test. Both the Main State Handler and path planner was included.

The test went as follows; the drone took off and flew directly into the table where the test crew were sitting with laptops.

Although this the test lasted a few seconds, there was several important observations. **1)** Always check without propellers first. **2)** Safety first, supervisors noted that safety glasses should be part of the test equipment for students, from this point on.

The intended test was tried in simulation first, but the drone behaved in a different way in real life. The failure was caused by the newly implemented Main State Handler, which activated the Offboard mode, before a mission was uploaded. This caused the drone to fly towards the bootup position, which was at the table.

3.3.5 28th November

The Offboard testing this day, was a test with the "whole" system, using a simulated precision landing. The "user" would click on the target coordinate and activate the drone and mission. The drone would fly towards the target and land with a simulated precision landing (just setting the altitude target lower and lower). This flight is shown in 3D path in figure 59. The operation altitude was 20 meters.

Figure 59: *The drone was in Offboard mode for the complete test from takeoff to touchdown.*

This drone was behaving as expected, showing that the main GCS node was working as it should after the problems in the last test. During this test the "no offboard" message" was thrown a few times, but the drone never went into failsafe.

The above test was repeated, but with small change: during the flight, a new mission was uploaded with a target at the takeoff point (approximately). The drone flew back and started landing as expected.

During the last flight, it was observed, that the drone lost height and at the same time, the battery remaining dropped almost to a critical level. A further investigation was needed for the battery calibration.

Observations from both tests was, that the "no offboard" messages was thrown at both flight. This indicated, that the repeater was not repeating as intended. The parameters for the companion link needed to be verified back at the lab before further flight tests could take place.

3.4 Expected battery life

When performing an operation with a self-built UAV, no datasheet can be provided for technical details such as flight time. Thus, the operational limits of the UAV must be tested to ensure that a 2km mission can be flown without crashing due to low battery. On 1st of October, a battery test was performed to determine the expected lifetime of a 4 cell LiPo battery with a capacity of 4250 mAh (35C), weighing 0.411 kg. This battery was tested at a range of speeds including 5m/s, 10m/s and 15m/s as seen in appendix.C The drone was able to fly approximately 2.18 km from a fully-charged state.

3.5 Range test

This test was to observe the frequency of the received messages to the drone. A RPi was mounted on the drone and cables were soldered onto the existing telemetry cable to sniff the packages. A rosbag was used to record the environment (/mavlinkz_rx topic was recorded).

The flight test was performed in Årslev with a maximum distance of 650 meters and the altitude was changed from 70 meters down to 20 meters during

the flight. The flight out was in approximately 60-70 meters of altitude and lowered to 20-40 meters when the maximum test distance was reached (limited by the test area). The drone was operated manually in Position hold mode.

To mitigate the negative effect from the antenna being close to the ground, a long stick was constructed to place the antenna approximately 5 meters above the ground. This construction was depicted in figure 38.

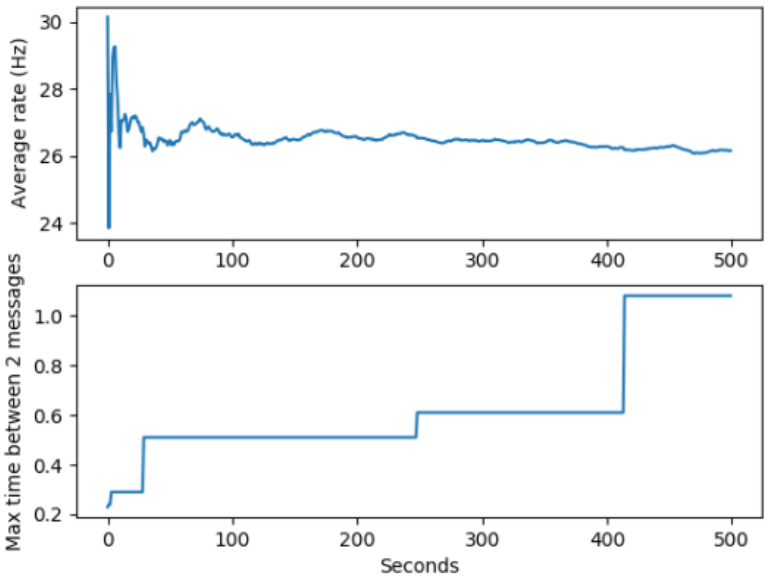


Figure 60: *This data shows the flight from a distance of 0-650 meters from the antenna. The average rate is shown in the top image. The mavlink_rx topic is almost what was expected (GCS is set to publish 20 Hz of setpoint targets and 8 Hz heartbeat messages, in total 28 Hz). The bottom image shows the maximum time between 2 messages. The maximum delay between setpoints must be less than 0.5 seconds (2 Hz). This limit is exceeded after a short amount of time. When the drone is flying away from the ground antenna the update rate was around 2 Hz at its slowest. 250 seconds into the test the drone was heading back and the altitude was adjusted to around 20-40 meters, which surpassed the previous worst case to around 1.6 Hz. Sometimes during the flight back the time between to messages was less than 1 Hz.*

The average rate in figure 60 also indicates, that the throughput should be good enough, since almost 95% of the expected messages are received at the drone side. This indicates, that the link should be working correct - also closer to the ground (in this case +20 meters).

As the purpose of this test was to get a rough idea about the receiving rate on the drone, only the mavlink topic was recorded. If the time allowed it and the test was performed again, then all topics could be recorded at both ends (both the GCS and repeater ROS system). This would require that the two logs was synchronized, but would allow to illustrate a better example of the throughput, altitude and distance and it could be plotted simultaneously. Further tests this day was interrupted by rain.

Another observation from this test is, that the telemetry link was terminated for a short time. The error message "ALL DATA LINK LOST" was logged multiple times. The conditions should be as ideal as possible for this project, with the given equipment. One of the many situations where the data link was lost is shown in figure 61.

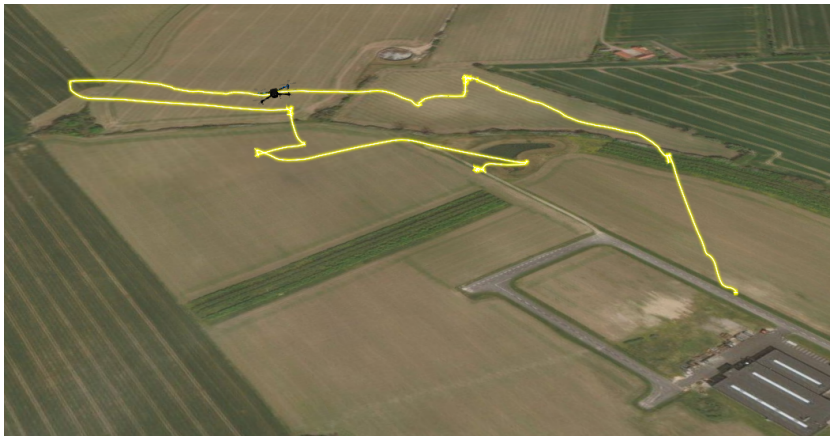


Figure 61: *3D path. The drone started in the right part of the image and ascended to approx 60 meters above ground. When the drone was approx 500 meters away in the same height the error loosing data link was observed.*

The observations about the average rate in figure 60 and the loss of data link connection in figure 61 was actually conflicting each other. How can the link be lost, even for a very short amount of time, without this is visible on the average rate? The abruption must be very close to zero if this error message is true. Or the actual case can be in the logging approach, since the "rostopic hz" method is by default using the average over the entire logging time, which could almost void any good interpretation, since the dropouts might be hidden in a long-time running average.

3.6 GPS error

The drone behaved odd when during a normal test of the Offboard mode and lost the GPS signal. Multiple flight tests were performed trying to locate this error. This required going back and forth from the test field each time after doing hardware changes, the debugging spanned over several days.

The error was observed the 1st of November and the first action was to test a programmed flight without using any of the developed subsystems, thus using the process of elimination to isolate the problem. The error casted by the Pixhawk was "No Global Position" and this error was repeated again in the programmed flight without any custom sub modules, which indicated a hardware or system settings problem. The drone had been operated with programmed flights multiple times before, without this error. A stepwise change of the hardware and parameters on the drone was performed in order to isolate the problem. Between each new flight test from this point on the drone sensors was recalibrated before each flight.

During a few repeated test this day with mission mode and loss of GPS, the drone to started drifting. The last try caused the drone to drift over a muddy area, which led to having the drone land directly into the mud. This required a thorough cleaning of the drone and one of the motors afterwards. As a reference of the test conditions this situation is shown in figure 62. In the other tests, the drone was switched into manual control when the drifting started, but in the last test, the drone operator experienced that this switching was ignored. This caused the drone to be killed in-air approximately 1 meter above the ground. The flight log showed, although, no sign of using the mode switcher, so this mud-landing might be caused by faulty operator use.

Figure 62: *The test area was very muddy, which was caused by rain and heavy-duty vehicles on-site. The drone landed (luckily) on top of the drive track, so only the bottom and one of the arms were affected by the mud.*

The 2nd of November the GPS was changed (Here1 to Here2 GPS was used as this was readily available from another project.) however the error was still present. Here1 and Here2 are GPS modules which interfaces great with the Pixhawk2 and is the to-go GPS choice when working with a Pixhawk.

A re-flash of stock firmware for the Pixhawk Cube was done for the test the 3rd of November. Also all the drone parameters were reset to default. The error was still present.

On the 5th of November, the Cube and GPS was replaced (now using a Here1 GPS again, after recommendation from the supervisors). All parameters

was set to default (except for the GPS position offset). Now the drone was able to perform programmed flights again without problems. This was tested with several missions.

Why this error occurred is a small mystery, since the drone did not crash in between the last working flight. Although, the drone suffered a hard crash in the very first flight test, which was (presumably) caused by a faulty soldering in the XT60 battery plug. This crash can have caused initial damage to the hardware, which led to the other problem.

3.7 Precision landing test

3.7.1 Simulation

The accuracy of the precision landing system has been put into repeated testing. In table 8 the results from 10 tests have been presented. The average error was calculated to be **29 cm** with a standard deviation of **13 cm**.

In the test, the drone takes off 6 meters, starting 23.4 cm away from the docking station. The takeoff position becomes an origin of the local coordinate system. Since the detector has issues with finding the correspondence above 5 meters, the precision landing system is authorized to pilot the drone while MPE is not tracking the target yet. The whole landing takes approx. 1 minute 20 seconds, and the known position of the docking station is subtracted from the final position of the drone in the local FoR, in order to get an error.

	errors [m]		
test	**x**	**y**	**total**
1	0.33	0.12	0.35
2	0.25	0.44	0.51
3	0.14	0.1	0.17
4	0.05	0.11	0.12
5	0.09	0.1	0.13
6	0.16	0.18	0.24
7	0.11	0.31	0.33
8	0.08	0.15	0.17
9	0.26	0.31	0.40
10	0.29	0.34	0.45

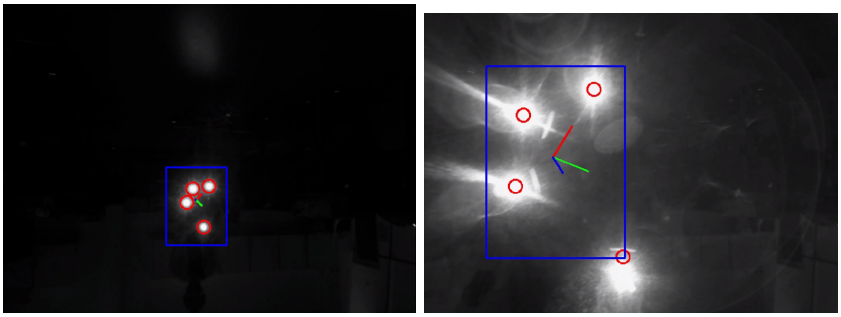
Table 8: *Results of the precision landing in Gazebo simulation.*

3.7.2 Real-life

3.7.2.1 28th November Four tests have been performed in order to verify the pose detection and tracking capabilities of MPE.

- **Indoor test 1.** The algorithm did good in a static tracking (slow, gradual re-positioning of the tracked object), detecting the correct pose for **74 out of 85 seconds**, in **87%** of the time. A few reflections have been detected as markers, yet in the correspondence finding phase they were correctly eliminated (figure 63a).

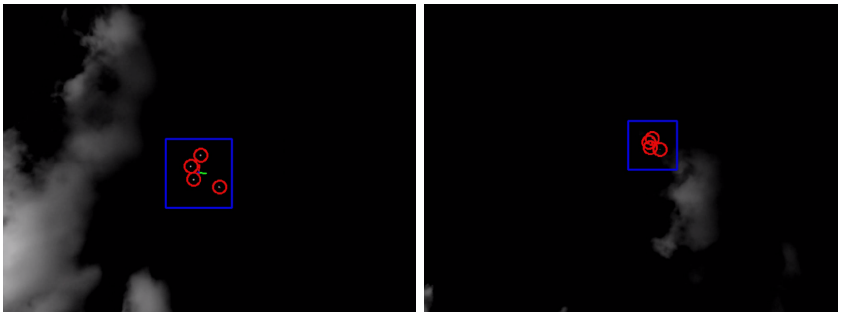
- **Indoor test 2.** Dynamic tracking turned out to be problematic when done inside. The markers were disappearing from the ROI, which caused the algorithm to switch between the brute-matching phase and tracking phase, slowing down the performance. The pose has been successfully estimated for **18 out of 34 seconds**, which is **53%**. An exemplary image seen by the detector has been shown on the figure 63b.



(a) Static indoor test. *(b) Dynamic indoor test.*

Figure 63: *The two tests performed inside on the dronelab in order to confirm that MPE is detecting poses for the new markers positions. The strong glare comes from the enclosed space and the LED light reflected back from the reflective interior accessories of the room.*

- **Outdoor test 1.** MPE was tested outside when the sun was obscured by the clouds, the conditions have been presented on the figure 64a. In no sun conditions it proved to track well, finding the pose about **60%** of the time, and tracking it up to 6-7 meters above the camera plane (figure 64b).



(a) Successful detection despite clouds being segmented from the image. *(b) Successful detection 6-7 meters above the ground.*

Figure 64: *The lighting conditions in the test performed outside. MPE precisely found the correct poses in both cases.*

A few false positives correspondences were found in the clouds, before the UAV was in the camera's view yet. The tracking of these poses was aborted

right away, showing that the MPE efficiently deals with the background noise (figure 65).

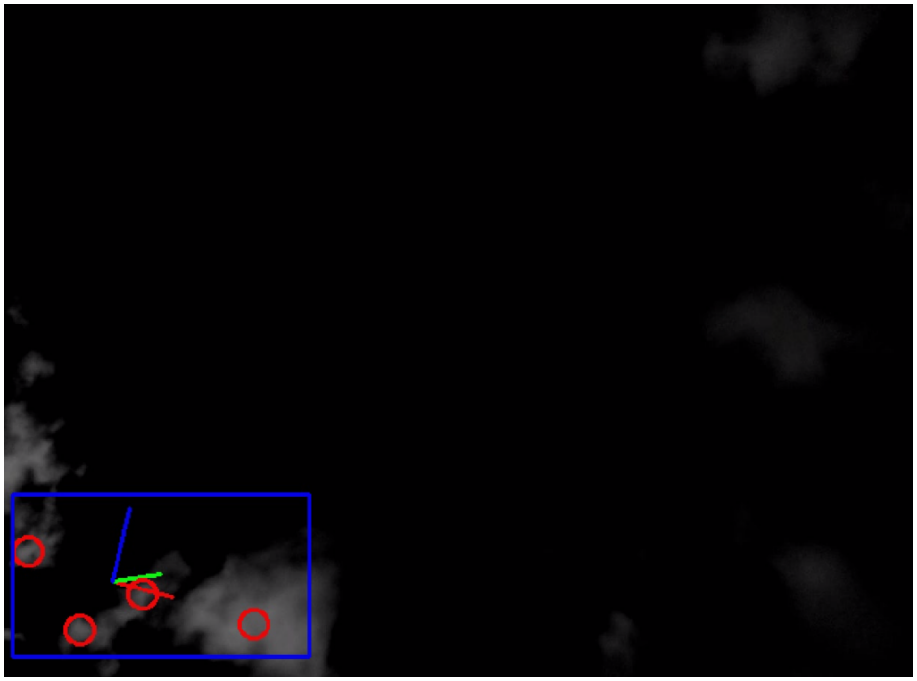
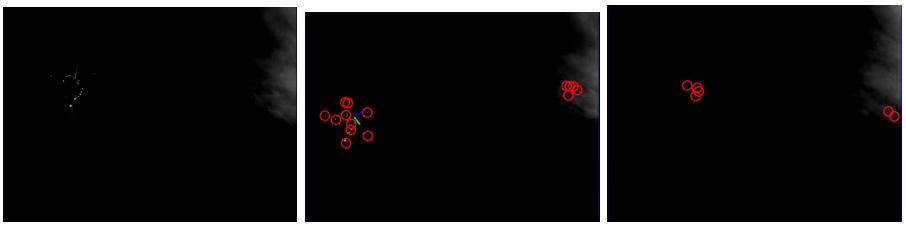


Figure 65: *The false positive found in the clouds. The tracking of it was aborted in the next frame.*

- **Outdoor test 2.** In the bright sunlight coming from the side, the tracker had difficulties with finding the right correspondence, finding correspondence at all, and narrowing the ROI for the tracking.
  - **Correspondence not found.** As shown on the figure 66a, the pose has not been found. The image has not been segmented properly for the detection, the reflections on the drone make it difficult to distinguish or find the LEDs in the image.
  - **Finding a wrong correspondence.** On the figure 66b, the wrong pose has been found, in which the drone is oriented vertically (the Z-axis, the blue, in parallel with a horizontal plane).
  - **Extended ROI.** On both figure 66b and 66c, the ROI has been extended to the size of the image. The essential idea behind using ROI is to reduce the computation time, thus increasing the throughput of the algorithm. When the size of ROI covers the whole image, the benefit in speed is non-existing.



(a) No detection. *(b) Wrong pose.* *(c) Extended ROI.*

Figure 66: *The problems recognized in the hard sunlight.*

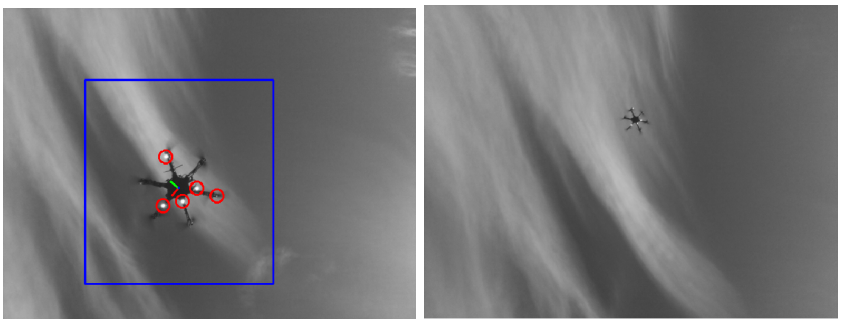
3.7.2.2 29th November The drone's frame has been painted black and all the reflective parts have been covered. The drone has been tested inside in order to find out if the detector finds the pose and the reflections still occur. Despite a strong artificial light source located below the drone, the bottom plate of the drone did not reflect anymore (figure 67).

Figure 67: *A frame for an indoor test to find out whether the reflections are present. The glare next to some of the LEDs is typical for indoor testing and has been considered negligible, since the system is meant to be used outside.*

3.7.2.3 3th December A short test in the model airfield reveled that placement of the LEDs has to be changed due to the lack of visibility of two of them in some configurations.

3.7.2.4 5th December A test outside has been performed to confirm the newly placed LEDs are properly tracked, and the precision landing regulator is

capable of centering the drone with respect to the camera FoR. The test revealed that the camera to local FoR translation have been wrongly implemented for the real-life regulation. MPE detection range has been found, the upper limit turned out to be lower than in the test on 28th November, approx. 4.5 meters, compared with the previous 6-7 meters (figure 68).



(a) *Successful tracking despite one false positive LED.*

(b) *Tracking has been lost after exceeding 4.5 meters above the camera plane.*

Figure 68: *The test of the regulated landing.*

3.7.2.5 9th December As a first step, the PID gains had to be adjusted because of a previous test, where drone was hovering and the regulator was centering it's position, some strong oscillations have been experienced. In this test, the PID gains were the same as used in the simulation, which resulted in an unstable system.

After lowering all three of the gains, the regulator became less aggressive. The issue has not been fully solved yet. When the hovering took too long, sometimes the would still start to oscillate around the center of the image. Due to the day coming to an end, the adjustments had to be stopped at this point and a slight change of approach was needed.

Instead of lowering the Z-component of the drone's position when a few consecutive error readings have been lower than a threshold, now the drone would keep on descending in the best known estimation of where the docking station is, and if MPE finds the pose, the target position would be accordingly corrected. Three successful landings have been performed using this approach, which proved to be more dynamic and faster than the previous one, but also more ignorant since in theory the drone could start landing outside of the camera's scope and it's target position wouldn't be corrected.

- **1st landing.** [24]
  - Test duration: 60.8 seconds
  - Landing duration: 8 seconds
  - Landing error: approx. 10 cm
- **2nd landing.** [25]

[24] `https://www.youtube.com/watch?v=DeFl2nVPnVg&feature=youtu.be`
[25] `https://www.youtube.com/watch?v=dS837BQr6zM&feature=youtu.be`

- Test duration: 45 seconds
- Landing duration: 6 seconds
- Landing error: approx. 40 cm

- **3rd landing.** [26]
  - Test duration: 64 seconds
  - Landing duration: 5 seconds
  - Landing error: approx. 25 cm

3.8 Full system test

As the maturity of the project is not at the expected level, the full system tests is several combined subsystem tests, rather than the complete set of subsystem.

The weather conditions, in the week up to the final acceptance test, contained a lot of rain, which led to less tests in the final days than expected.

3.8.1 7th December

First test using all subsystem, except precision landing controller, was performed early the 7th. In the moment the sun rose, the drone was launched with an attempt to use a "full" system test. A new implementation is in the Main State Handler was a timeout. If the docking station did not detect the drone within 20 seconds, the drone would send a land signal.

The test scenario was; 1) Takeoff from UI input and fly 80 meters away 2) When arrived at final waypoint, descend to 10 meters altitude to be visible for docking station 3) If docking station does not detect drone after 20 seconds, send land signal 4) After landing, send land completed signal to reset system for new mission. This test is illustrated in figure 69.

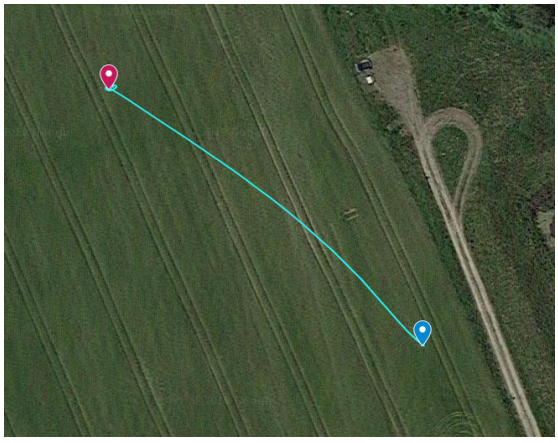


Figure 69: *The blue marker is the takeoff point and the red marker is the final waypoint.*

[26] `https://www.youtube.com/watch?v=C-FRhc3EKwo&feature=youtu.be`

After touchdown, the system was reset and ready for new mission. The caused problems in the range, since the drone could not takeoff again. The reason for this is most likely caused by the drone being on the ground - and the high wet grass and weeds did probably not help with the telemetry conditions. Although, the drone did not come back, it completed a one-way "full" mission test, in a very simple case, where the Pathplanner did not detect no-fly zones and the UTM node did not interfere either.

To try mitigate this range issue, a new mission was planned, where the drone would takeoff and land close by and then upload a new mission right after. This path is shown in figure 70.

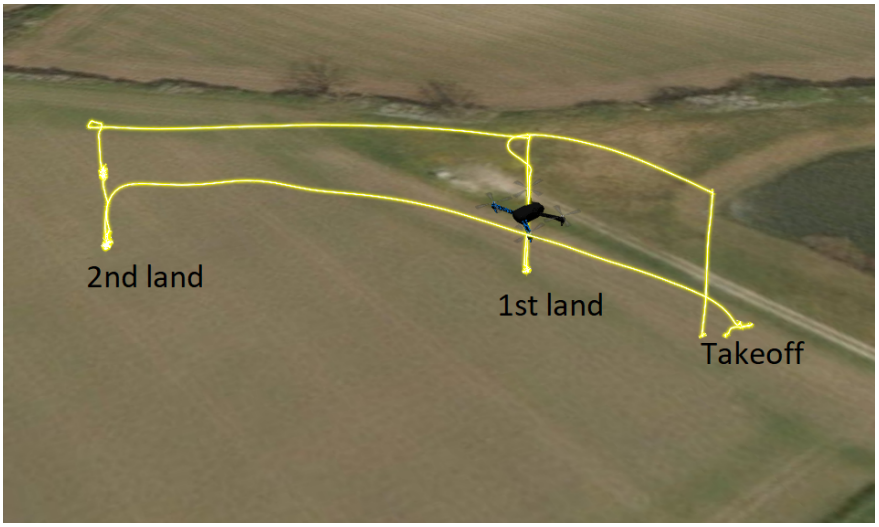


Figure 70: *Drone is taking off, flying a short distance and landing at first landing spot after timeout. After this, the drone is receiving a new mission with target at the 2nd landing spot.*

The second test worked better. Here the drone was able to takeoff and land, reset the system, receive a new mission and fly towards that target. After landing a the 2nd landing spot, the drone was in the same situation with no ability to takeoff because of the missing presence of telemetry messages.

A promising observation is, that the Offboard mode did not fallback at any point.

Further tests this day was not possible due to lack of drone operator.

3.8.2 9th December - day before final acceptance test

With heavy rain most the previous days, this was the final chance to get the full system tested.

The testfield was Årslev, close to SDU, if e.g. the batteries needed recharging. Today it was desired to focus on two main things - get the landing controller tested and do a full system test.

To prepare for the final acceptance test all checklist was prepared and it was decided who had the different responsibilities on the final day. All items was checked and rechecked before leaving campus, and all flight missions were

performed in the same way, as it was intended at the final acceptance day. This included verbal communication with e.g. launch signals from GCS operator to drone pilot.

Figure 71: *The setup for the last day of preparation.*

The first 12 flight tests was an attempt to get the landing controller to work properly.

One of the observation was that the gains in PID controller of the precision landing system were not adjusted properly. The test was repeated multiple times to find the best parameters.

Sundown came closer and closer this day, and it was now time for doing some full system tests before attempting more tests with the landing controller. The final tests was rushed with the darkness approaching and the two final tests is shown in figure 72 and 73.

Figure 72: *This test was similar to previous tests with takeoff and landing after timeout.*

This test intended to fly out and land at last waypoint, to see if problems with Offboard mode was experienced. During this flight the Offboard mode was only lost a single time during takeoff. The drone flew out in a small curve to the final waypoint and lowered to 10 meters and here the drone was taken down manually, due to a mistyping in one of the subsystems.

Figure 73: *Same test repeated, but this time the drone would reset and fly back using an uploaded mission.*

This test was performed almost perfectly (without the use of camera for landing). Very small problems with the Offboard was observed.

The sun was almost coming down at this point, so no further tests, since the drone was almost not visible in longer ranges. This was of course problematic to have so few tests, but it came down to prioritizing, since a few more tests and adjustments was desired with the landing controller before leaving the test area. These tests was performed in close range until it became too dark.

3.8.3 Final acceptance test

On the final day, the team arrived at the test area just before takeoff clearance.

The position of the landing waypoints was determined from the acceptance test brief supplied by the supervisors. The docking station with camera was positioned at the waypoint nearest to the SDU Hanger *Waypoint A*, whilst the farthest waypoint was purposefully left empty *Waypoint B*. This was done as the Docking station requires an connection to the main GCS for the precision landing. When the UAV initiates a landing at Waypoint A, the landing handler will attempt a precision landing when it detects the drone within the camera FOV. On the other hand, the landing handler will perform a simple GPS landing at waypoint B, as there is no docking station present to perform the precision landing.

Before the UAV could take off, a pre-flight check was performed to ensure both the GCS and UAV were flight-ready. The preflight procedures is as follows:

- Turn on transmitter
- Start the ROS nodes. All but UI and pathplanner.
- Attach propellers.
- Check propellers are correctly attached.
- Check SOC.
- Turn on the drone.
- Test Holding mode on the drone.
- Test manual arm/disarm.
- Confirm messages from drone on GCS.
- Confirm initial starting position.
- Confirm target destination.

The GCS laptop was setup on a table with an extension monitor attached to provide a larger screen for system observations and node monitoring. The docking station was setup a distance of 10m from the GCS and connected with an 80m ethernet cable. The Docking station also contained its on own battery supply. Due to the windy conditions, The antenna extender was positioned next to a support vehicle to prevent it from tipping over.

During the test periods, each team member was assigned a role and responsibility. This is to increase efficiency and organisation in case of unforeseen scenarios. This is the responsibilities assigned as:

Per	Pilot
Thor	GCS/Spotter
Oscar	GCS/Communication
Blaise	GCS
Mathias	GCS/Spotter
Mads	Communication/Video recording

The weather conditions throughout the test day was cloudy weather with wind speeds of 8-11 m/s and gusts of 12-15 m/s[27]. The max horizontal speed of the drone was set to 10 m/s, as faster speeds are suspected of causing the internal battery calibration to give errors. The 10 m/s was also the specified speed in the project specification.

Before any flight test could be performed, it was found that the drone needed to be re-calibrated. The recalibration was attempted 4 times without success. It was suspected that the recent addition of the payload magnets had caused interference with the sensors, and thus were removed. Following this, a recalibration was attempted without success. After a brief discussion, it was decided to try recalibrating in a different location - the UAV was moved to the middle of the test field which resulted in a successful calibration. The reason for the recalibration failure is still unknown, but resulted in a 45 minutes loss of test time.

As mentioned above, the wind conditions were very strong, which caused the antenna extender to tip over before the first tests were performed. A inspection of the telemetry casing showed no signs of damage and the antenna was raised, ensuring it was secured to the car. The first attempts fly indicated that there was a problem with the telemetry connection, with a low number of packages successfully being sent. To resolve this, the antenna kit was replaced and later inspection showed that the antenna soldering was broken away from the board.

3.8.3.1 The tests Because of the initial testing did not go as planned, because of the telemetry issues, the takeoff position was moved into the middle of the field. This means the setup was changed and the drone would now fly back to the docking station for precision landing attempt.

A total of 15 flight tests were performed (omitting all non-takeoff tests caused by the telemetry issues).

The purpose of tests 1 and 2 involved flying to the docking station to perform a precision landing, but during both flights, a critical battery warning was raised, causing the drone to immediately land. The battery was charged to a SOC level of 90 % before the flights. These problems are a result of a poor battery calibration and an attempt to avoid this, a new battery was replaced after every flight test.

During test 3, offboard mode was lost multiple times. The drone only travelled halfway towards the intended landing spot before the pilot took over and landed the drone due of the unstable flight. At this point, the crew switched positions with Mathias as the new pilot and Per as GCS Monitoring and Operations Control. The switch was done as Per needed to observe the errors with the Offboard, as the high number of lost-Offboard connections had not been experienced before.

In the 4th test, the UAV performed takeoff and a few Offboard problems were experienced before the drone went into critical battery failsafe and landed. Initial the battery SOC was more than 90%. The battery was swapped for a fully charged.

The results of 5th test fared better than the previous tests, but the mission was was never completed. The path is shown in figure 74.

[27] `www.dmi.dk`

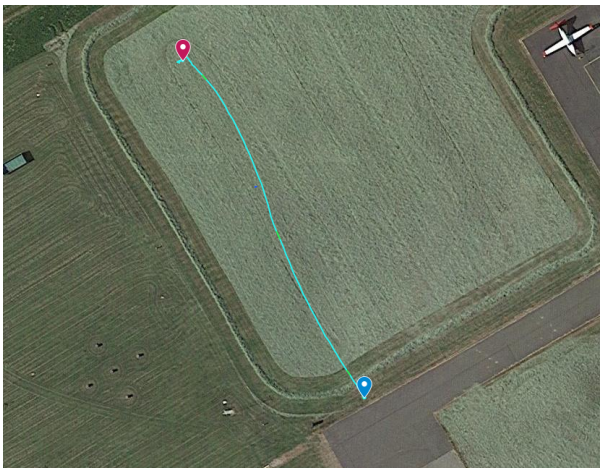


Figure 74: *Takeoff at blue marker and intended landing spot at red marker.*

During this test, the drone performed a good takeoff, with minimal losses in Offboard. It flew towards the landing spot and reached the final waypoint successfully. However it was observed, that the position updates did not transmit successfully to the drone, resulting in the mission planner not being able to send the drone to the landing waypoint, resulting in offboard connection loss. The drone was landed manually by the drone pilot.

The purpose of test six was to check the operation of the drone within close range of the telemetry antenna, in an attempt to reduce the problems in offboard mode. The plan was for the drone to takeoff and landing close to the antenna.

The drone took off directly towards the position of the docking station (not intended). The pilot was under the assumption that he had taken over manual control at this point - but the log showed that this was not the case. The drone should have taken off into the air from the current spot, but it did not. Then it flew above the test crew from our own group and at this point the pilot was able to put it into position hold and performed a controlled manual landing.

An observation in this test; since the drone took off directly to a wrong coordinate, this must be because some mixup of coordinates in the memory. It is believed that this was caused by some of the nodes not being restarted in the correct order before the test was carried out. Thus, the internal origin was shifted after the placement of the drone. The order of the launching the ROS nodes was discussed and agreed upon for future tests.

The 7th test is shown in figure 75.

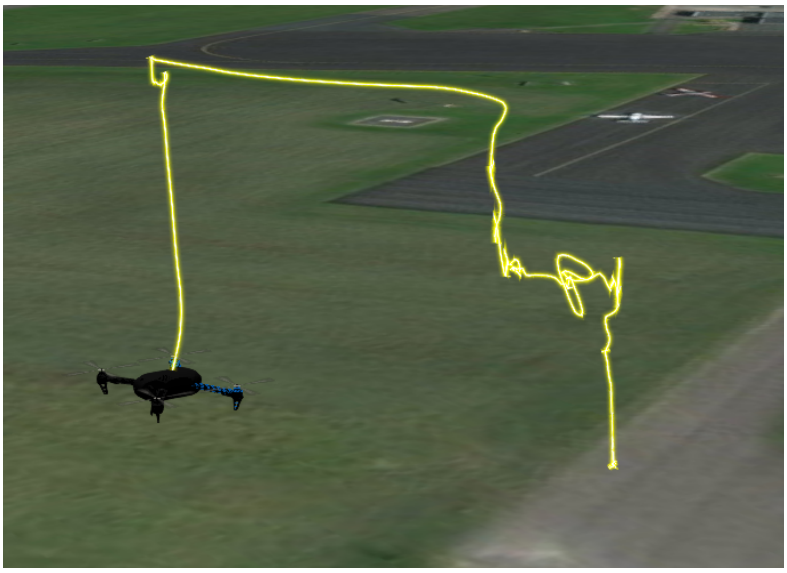


Figure 75: *Takeoff in the left side and a attempted precision landing in the right side.*

This test was an attempt to use regulation for the precision landing. During operation, the landing handler started regulating on final approach. The drone performed takeoff as expected, but the flight was very unstable because of the Offboard was lost multiple times during path following. This Offboard problem was also heavily observed during precision landing attempt.

When the landing regulator was enabled to land the drone, the regulator was became very unstable. As a result, Manual control was taken when it was deemed the controller was too unstable.

During the landing attempt, it was observed that the local position received from the drone was at a rate of only 0.25 Hz. This is very critical fault, because this rate is used to directly control the drone. The regulator only sends new setpoints, when this local position is received.

Test 8 had the same intent as the previous test, but the flight looked similar and the publish rate for the local position was still very low during the flight. The drone was landed by the pilot.

This low throughput was considered mission critical and it was suspected, that this was caused by poor antenna or interference from other groups. A better antenna for the ground station was obtained from another group and mounted on the antenna extender.

Another precision landing attempt was done in test 9, but with new ground antenna. There was a great improvement in offboard mode, with almost no loss in communication during the majority of the flight. During the landing phase, the Offboard kept disconnecting whilst the drone was positioned at an altitude of approx. 10 meters. The regulator was found to be slow in responding to movements. The drone was taken down manually, with modifications the landing handler to increase descend speed.

In test 10, the above test was repeated with faster descend speed. Offboard mode was mostly stable, without significant connection loss during the flight.

The landing handler started regulating the drone whilst descending, but the regulator did not do as expected due to the descent speed still being too low. The can be explained by the landing handler only descended the drone when it gets received a position update. This meant drone was landed by the pilot. The 3D path is shown in figure 76.

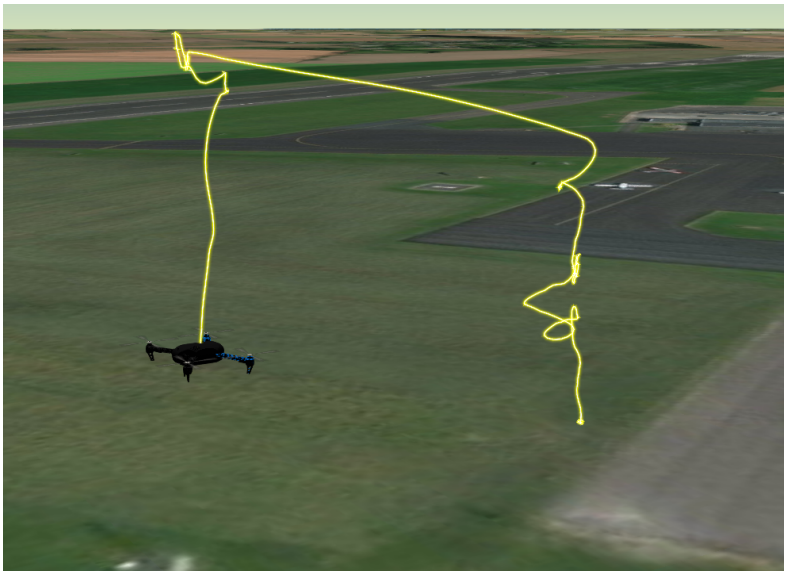


Figure 76: *During landing in the right side of the image, it is visible, that some level of correction is coming from the regulator.*

In the 11th test, the landing descend rate was adjusted again and the test above was repeated. A problem occurred where the landing regulator did not detect the pose during descent. The drone was manually landed and inspected where it was discovered, that two of the IR LED's used for landing had become damaged. The damage was caused be high pressure and was likely caused by accident when repositioning the drone between tests. By design, the LED's are by covered by a lid and shroud, but these lids were removed, as it was found they were occluding the LEDs when the drone was tilted at a steep attitude (such as position-holding in the strong wind). As a result, no more precision landing tests could be carried out.

The 12th test was the first attempt to testing the path planner with dynamic NFZ. However, on planning a route through a dynamic NFZ - it was found to have expired.

The 13th test did manage to plan a route around a NFZ, but also required the uav to fly across the flight-line of other teams within the test areas. The planned flight is shown in figure 77 and the actual path is shown in figure 78.

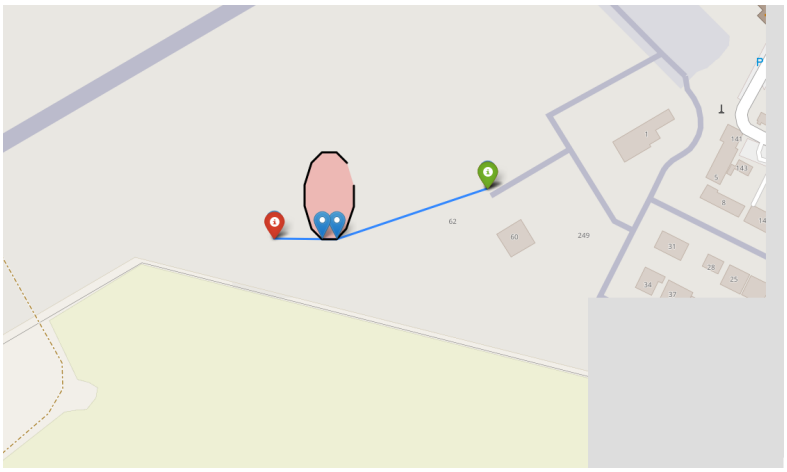


Figure 77: *The planned path, with 2 addtional waypoint to go around the no-fly zone.*

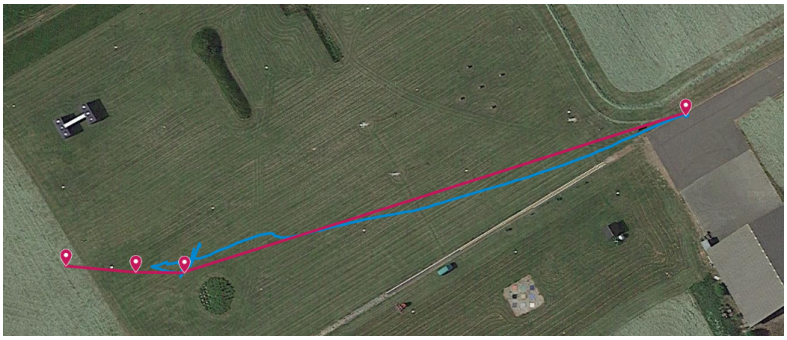


Figure 78: *The red line show the expected route and the blue show the actual route. The planned path was approx. 200 meters.*

During this test, a no-fly zone was found, and a path was planned around this zone with additional waypoints instead of a straight line. In comparison, the route the UAV actually took did not follow these waypoints and resulted in flying the NFZ. On entering the NFZ, the UAV experienced many Offboard problems and went into failsafe.

In Test 14, The UAV did not takeoff, however this test could be considered a success because the drone was situated inside a NFZ during the desired takeoff time.

The last test, number 15, was another attempt to plan around a NFZ. It was found that the previously-used NFZs had expired, with another one appearing. The problem with this new NFZ is that is was not situated within the goal waypoint used for the previous tests, resulting in the drone never being able to plan around the new zone. During this flight, numerous Offboard problems occurred and the drone fell back into failsafe - it was manually piloted back to the landing site.

4 Discussion

Pilot control In the current implementation, each pilot will run if they are allowed to do so. If there is an error, they could possibly publish setpoints when they were not allowed to do so. A fix for this could be that all pilots publishes setpoints at all times, and these messages are relayed through a central node to control which message is being transmitted to the drone. This will however, introduce some overhead in the internal ROS system in terms of bandwidth. Another approach could be to combine the current implementation with the same central node that republishes messages. In this way, the pilots would only publish if they were "allowed" to do so - they can never accidentally publish to the drone. This it be caught in the node which relays the messages, it will know which pilot is currently being active.

Deconflict According to the documentation provided by the supervisors, all navigation should follow the WGS84 standard. The native GPS from MAVLink uses MSL as a height reference. To convert it the WGS84 height, the zone needs to be subtracted from the MSL and then used to correct the altitude.

The deconfliction used in this project was only conducted to a bare minimum, and was only usable between the groups. If more time was available, the node would have been able to handle more situations and could also take the priority of the UAVs into account. A priority shift could happen when carrying critical cargo, or if a UAV have a higher risk class.

Path planner The path planner was created as a graph structure, with the idea to reduce the calculation time by having as few nodes as possible. Another solution discussed was to create a grid with a fixed resolution, using the gridlines to navigate from start to end. However, this would create a lot of nodes to take into account - as a result, this approach was abandoned in exchange for the current solution.

As is, the path planner calculates a new path each time it has finished the previous path. This could be considered an inefficient approach if the environment conditions have not changed. The path planner would not utilize the same route plan, and instead use system resources to plan the identical path. Additionally, changes in environment conditions during the planning process would not be taken into account until the next planning phase. To solve this, the path planner could be improved to be more reactive or reconstructed as a callable service.

The path is calculated using Dijsktra's algorithm, but other algorithms could also be considered, like the A*. The major difference between the two algorithms is the heuristic distance between destination and start, used by A*. One could argue that with the current system, A* would fit better because the drone only flies to one destination. Thus, the system can ignore other target nodes and just calculate the quickest path to the destination. However, one could argue that Dijkstra's algorithm would serve as the better algorithm for future purposes, where there could be multiple target locations at which A* would have to be calculated from each start and end node.

The buffering size of the NFZs could be reduced since the GPS error is not as high as initially thought, but it was decided that it was more important to

stay clear and safe of the NFZs rather than taking a path on the boundary.

When creating the bounding box, only the starting position and the end position are considered which could lead to an undesired outcome if a polygon reached outside the box. The bounding box should then be expanded to so it pertains the polygon. Otherwise the path planner does not take into account the airspace from the polygon and back into the bounding box. This could lead the path to go through NFZs without the it knowing it because it does not look at NFZ outside the box.

The path is always planned according to the information that the path planner has when it starts to plan. This means that even if it gets information about a dynamic NFZ popping up in a few seconds, the path planner does not take any action because it does no preemptive planning.

If the dynamic zones had an epoch start time, some time before it appeared, then the path planner could estimate from a drones current location, if the drone would have enough time to cross/exit the dynamic NFZ. The path planner could also try and reroute the drone, before the drone ever entered the "to-be" dynamic NFZ.

The pathplanner does not take the SOC into account and only assumes that is enough charge left to complete the mission. These assumptions should not be pertained in further development, as the path planner should be adapted to check whether the UAV has enough SOC to perform the mission.

Drone Command Handler Why use the MAVLinkLora library when other implementations to use Mavlink already exist? MAVROS is one example, but this implementation introduces a large overhead of redundant data, which would flood the telemetry channel.

Using Offboard mode required a great amount of developing our own subsystems, which would already available in Pixhawk - if Auto mode was used instead of Offboard. A common rule of thumb is that you should not develop anything that already exists. A reason for using Offboard mode is that it gives the developers fast and responsive control over the drone through minimal messages in the telemetry. However, it has become very clear that Offboard mode is not ideal for use in this way due to the problems in telemetry communication. A companion computer could be used instead, with the communication happening through the use of GSM modules or similar hardware. The Offboard issues could possibly be mitigated be using a hardware repeater, for instance a RPi, or writing a custom PX4 module to repeat the received setpoints. It is clear at this point, that the RPi repeater did not work as intended, but time constraints meant that debugging this before the final tests was not possible.

The tests regarding Offboard mode were performed at a late stage in real life. If the problems had been discovered earlier, it might have provided the chance to change the implementation and continue with Auto mode. Notwithstanding, changing to Auto mode could have solved some problems but also introduce others. Additional components, such as precision landing, would not have worked using the Auto mode and require the use of offboard mode to function. Unfortunately, this is also where we experienced the majority of the offboard communication losses when the drone came close to the ground. Using built-in modules within the Pixhawk firmware to land without Offboard required modifications to the firmware, or further development of the MavlinkLora library. Modifying

the PX4 firmware was (as already mentioned) not allowed and developing the MavlinkLora library to support newer Mavlink version was not considered.

The initial tests using Offboard mode proved to be significantly stable than what was experienced during the final acceptance day. These tests were performed using only the actual subsystem for the Drone Command Handler. Further tests would be necessary in order to determine whether the laptop or complexity of the GCS subsystems were cause of Offboard problems.

Precision landing During the real-life tests, the PID landing regulator was found to be unstable during operation. Two possible explanations have are listed below:

- The gains for real-life were not able to be tuned correctly during the timeframe.
- The latency between receiving messages and updating the target position.

One reason for sub-optimal results in the PID landing regulator could be down to incorrectly tuned gains. During simulation, the gains could be adjusted to improve the operation, as the virtual conditions would be the same every time. However, the environmental conditions in the simulation do not reflect their real world counterparts. A solution to this would simply be to spend more time repetitively testing the regulator in the real world in a variety of different conditions.

One big downside to the precision landing system is that it requires a stable telemetry link to operate. At the final acceptance test, the MAVLink communication was found to have an update rate of **0.25 Hz**, which made the landing pilot incapable of handling the procedure. Both the descending handler and the pose estimations rely on a constant flow of the local position updates from the drone to adjust the drone accurately. Extra tests would be required to determine the minimum rate of communication, and then implementing failsafes in the case where the communication cannot be consistent. Alternatively, a good telemetry link could employed by adding a local telemetry link to each docking station, which would be utilized during the landing phase and takeoff phases.

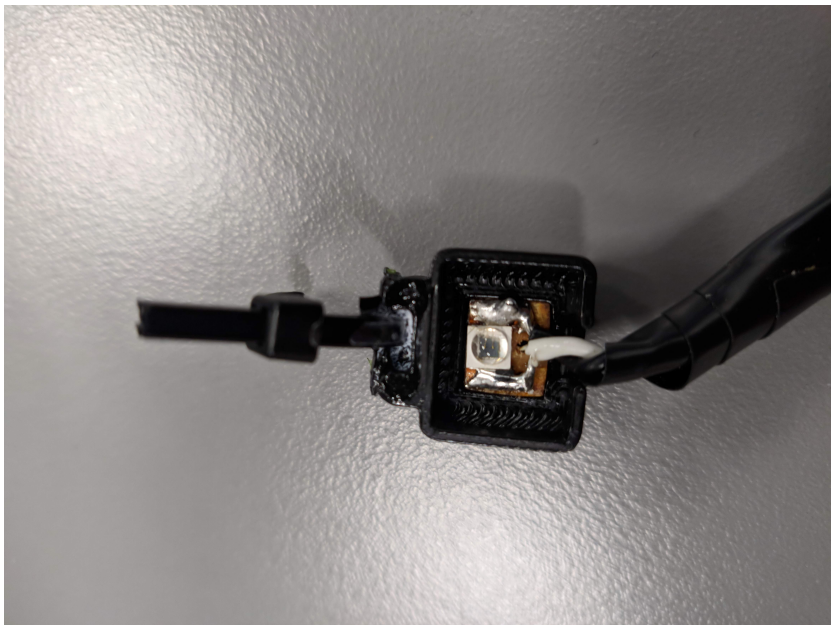


Figure 79: *An example of a single LED casing with the lid removed.*

During the acceptance test, a problem with the LED mounts was found that the LEDs were obscured by the case lids, causing the pose estimator to lose tracking. These were subsequently removed (figure 79) which improved the results. However, removal of the lids resulted in the LEDs not being fully protected and resulted in two of the four LEDS being broken through contact - presumably with the terrain. A quick way to fix this is through better positioning of the LEDS, or choosing components that were more rugged in nature. A more complex solution could be to mount the camera onto the UAV, whilst attaching the lights to the docking station (similar to the the IRLock system). A reason this method was not pursued in the beginning is primarily down to control - the team wanted to implement a method where it was possible to experiment and develop their skills, rather than purchasing a "plug and play" solution with limited modification or expandability.

ROS During the acceptance test, the full system was tested and due to not having full knowledge of ROS a mistake was made: setting the queue size to 0, which in reality means that it is infinite. This mistake has been made in the full system and every publisher has an infinite queue.

This error made it possible to quickly fill the system with data and you would have memory problems, this could be why the system was transmitting data at the slow rate that it did. This together with the telemetry problems made it so the acceptance test did not go smoothly.

5 Conclusion

Due to the complexity of the system, each part was assessed and evaluated depending on the initial expectations and then compared to the output perfor-

mance

For the UI, The design of the webpage emphasized the ease of use for user with a non-technical background. The GUI was kept simple with a few buttons sparse feedback from the system. Time was not spent making this website look aesthetically pleasing, as it should be stressed that it was a proof of concept.

The main state handler was able to administer control and provide it to any handler responsible for a specific task. This administration was implemented through a state machine to provide a reliable framework to which only one pilot was able to control the UAV, and not have deal with conflicting WPs. To keep the status updates as simple as possible, the state and pilot in charge, is also only published once when a state change.

The UTMS handler was designed to monitor and update the UTMS without any intervention from the user. It was able to send and receive tracking updates whilst providing information on static/Dynamic NFZ wherever possible. The coding framework was structured in such a way that it could handle unstable internet connections and notify the user accordingly - providing as much feedback without being too distracting. However, there was some functionality that was not implemented that could be considered for further work. For example, the "System critical" status monitoring was not implemented as it was assumed that a test pilot would take control in the scenario during the testing phases.

As the Deconfliction Pilot was sourced from another group, it can be assumed that it was thoroughly tested before distribution. Additionally, the Deconflict was tested in simulation, but during the acceptance test it was ensured that each group would work within separate areas as to cause problems. For future development, it would be ideal to test this system further in real-life scenarios

The path planner was able to plan around both static and dynamic NFZs, taking into account a number of few special criteria including landing or taking off within a NFZ - as well as entering a NFZ. The algorithm was able to solve the task of planning and presenting the route in an simple visual manner. In general, the Path planner worked as intended and was able to fulfill it designated task. However, the pathplanner suffered from being very inefficient and was shown to be very resource intensive during operation - an improvement for future implementations would be to optimise the code to run faster and more reactive.

The development of the mission handler turned out to be a vital significance for using Offboard mode with the drone. It was designed with the approach of using basic operations to reduce its complexity. The mission handler was tested thoroughly, and determined to work within an acceptable level for the task requirements.

The use of Offboard flight mode proved cause a lot of unforeseen challenges, which was a major factor in the limited successes during the final acceptance test. The main problems experienced were due to latency issues and an unstable telemetry link, which caused the UAV to utilise its internal failsafe measures on multiple occasions. At numerous times during the project, it was discussed whether the current solution should be changed to use existing functionality and avoid using Offboard flight mode. Offboard mode was considered a better solution for optimizing a fast reacting UAV and also the Offboard mode was needed to perform the final landing, since the landing solution required non-supported firmware functionality. As the Offboard mode was preferred after the several discussions, with multiple arguments both for and against this mode, the telemetry issues needed to be mitigated. A solution attempted was to remove

the negative effects of having a antenna close to the ground - here a longer antenna holder was constructed, which led to smaller improvements. Another solution to cope with the latency problems was to use a repeater situated on the drone, thus using a wired connection directly to the flight controller, using both available telemetry inputs on the flight controller. The repeater did not appear to have any effect and would require further investigation to locate the issues here.

Despite complications in acquiring parts, the precision landing system implemented showed signs of promise despite being developed in a short period of time. These complications in obtaining vital components for the vision system resulted in limited time for testing and tuning, or implementing alternative methods to complete the task. Should the project be pursued beyond the deadline, further testing and refinement would be needed to ensure the robustness of the landing algorithm and guarantee fully operational status.

The simulation of the UAV and docking station in this project was used to its full extent, with all the vital test being conducted before real life testing could occur. As is always the case with simulations, the full extent of the real life scenarios cannot be considered, but Gazebo-sim played a vital role in finding important bugs that would cause the drone to crash in real life.

The Docking station for this project was designed to accommodate a lot of features in a compact and portable solution, whilst fulfilling the criteria stated in the project brief. The loading and unloading mechanism was tested and shown to have worked, but problems with self-made wires occasionally caused the stepper motors to stop working. The embodied design of the funnel arm had shown to minorly succeed in aligning the UAV to the slider gangway, but the problems during the final acceptance test resulted in the docking station not being utilized to is intended purpose.

The project management method was heavily influenced by the SCRUM technique. The tasks were organized in backlogs, with sprints being conducted each week and making tasks more manageable to carry out whilst giving an overview of the progress. The estimated time and the actual time was tracked through the use of a burndown chart which helped to identify possible delays.

It was difficult to obtain an accurate estimate of the time used per team member as each worked at different rates to complete their assigned task. The team met two times a week to discuss progress and identify problems, however due to differing schedules of each team member it was not able to meet more than this during a week.

The current solution of transporting blood samples, is expensive and resource intensive. An accurate estimate of the costs to bring this product to market, is uncertain, as the project requires technologies that are still in the process of initial development and thus have no price on system costs or maintaining. In addition there is a high uncertainty about future requirements for a product of this type.

The budget for the current development of 500.000 DKK is only a fraction of the real budget for a complete and proven system. Without financial background or knowledge, it is impossible for student developers to state a more accurate estimate without it being a wild guess for the payback period or return on investment.

In closing, the group succeeded in creating an system which lets the user command a blood delivery drone through an easy-to-use web page. The GCS

is able of planning a safe path through the environment and demonstrates it can precision land the drone when required. The System can by no means be considered complete, however it does demonstrate the core functionality set out in the Project description. As a result, the system would need more work before it can by sold as a commercially-viable system.

References

[1] http://www.diva-portal.org/smash/get/diva2:783979/FULLTEXT01.pdf. (Date last accessed: 16/12/18).

[2] http://rpg.ifi.uzh.ch/docs/ICRA14_Faessler.pdf. (Date last accessed: 16/12/18).

[3] https://github.com/uzh-rpg/rpg_monocular_pose_estimator. (Date last accessed: 18/12/18).

[4] http://afar.net/tutorials/how-far/. (Date last accessed: 16/12/18).

[5] https://hobbyking.com/en_us/hkpilot-transceiver-telemetry-radio-set-v2-433mhz.html?___store=en_us. (Date last accessed: 16/12/18).

[6] https://www.amazon.com/433mhz-Antenna-Connetcor-Female-Cable/dp/B07DDF5YCJ?ref_=fsclp_pl_dp_1. (Date last accessed: 16/12/18).

[7] https://docs-emea.rs-online.com/webdocs/1443/0900766b81443a1b.pdf. (Date last accessed: 19/12/18).

Appendices

A Flight Log

Total multirotor	**2.50**	hours

Model	**F550 quadcopter**
UAS category	Multirotor
UAS type	Quadcopter
Transmitter	Spektrum DX6/DX9 w. satellite receiver
Frame	F550
Frame diameter wo. rotors [m]	0.55
Weight (approx) [g]	1200
Control board	PixHawk 2 (standard firmware)
Motor	AIR2213 KV920
ESC's	AIR 20A ESC 3-4S LiPo
Battery	~5.8Ah, 4s LiPo, 35C
Props	T9545 self-tightening props
Battery duration	30 min
Standard flight time (DOD 20%)	27 min
Total flight time	**2.50** hours

Date	Time	Air time [min]	Location name	Lat, DD	Lon, Lat, DD	Opperator	Flight notes + flight log
13-09-18	16.19	1	Model airfiled Odense	55.47204849	10.41470375	Mathias Madsen	Test for MavlinkLora and QGround control https://review.px4.io/plot_app?log=90267d10-2c16-4741-b4e1-cf42d80da277
13-09-18	16.22	1	Model airfiled Odense	55.47204849	10.41470375	Mathias Madsen	Test for MavlinkLora and QGround control https://review.px4.io/plot_app?log=3123af36-59f3-4db1-b02e-ae84ce4d0b7b
01-09-18	14.17	1.5	Model airfiled Odense	55.47204849	10.41470375	Mathias Madsen	Test of battery lifetime at 5 m/s https://logs.px4.io/plot_app?log=1aad1f9e-3513-4855-88ff-1fdd019790f4
01-09-18	14.33	1	Model airfiled Odense	55.47204849	10.41470375	Mathias Madsen	Test of battery lifetime at 10 m/s https://logs.px4.io/plot_app?log=9e9fa486-85b7-4008-977d-91a8217e44ae
01-09-18	14.40	1	Model airfiled Odense	55.47204849	10.41470375	Mathias Madsen	Test of battery lifetime at 15 m/s, crashed because of battery settings was wrong https://logs.px4.io/plot_app?log=5882317f-daab-41cd-bb4e-1e3c67cae560
30-10-18	14.31	2	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://review.px4.io/plot_app?log=ee5b086a-59de-46f0-bcf7-6af0ce091a5b Flight test description: https://docs.google.com/document/d/1uEEwlkjVM9oQvFMZAgXO4_Ecuy3Eajp3Tqscz1KTFEk/edit?usp=sharing (Flight #1)
30-10-18	14.35	2	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://review.px4.io/plot_app?log=62730126-b1fb-4597-967c-24af1925c85b Flight test description: https://docs.google.com/document/d/1uEEwlkjVM9oQvFMZAgXO4_Ecuy3Eajp3Tqscz1KTFEk/edit?usp=sharing(Flight #2)
30-10-18	14.42	2	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://review.px4.io/plot_app?log=62730126-b1fb-4597-967c-24af1925c85b Flight test description: https://docs.google.com/document/d/1uEEwlkjVM9oQvFMZAgXO4_Ecuy3Eajp3Tqscz1KTFEk/edit?usp=sharing (Flight #3)
1-11-2018	11.19	1	Model airfield, Årslev	55.309922	10.463653	Per Breum	The following flight over several days is documented in: https://docs.google.com/document/d/1imb2Bu7J0aXgOX0oL5hGH4nuAICBEoDsQ1F2_yQzWE4/edit?usp=sharing
1-11-2018	11.27	1	Model airfield, Årslev	55.309922	10.463653	Per Breum	-||-
1-11-2018	11.33	2	Model airfield, Årslev	55.309922	10.463653	Per Breum	-||-
2-11-18	16.20	3	Svendborg Model airclub	55.057801	10.569069	Per Breum	-||-
3-11-18	13.34	1	Model airfield, Årslev	55.309922	10.463653	Per Breum	-||-
5-11-18	14.21	1	Model airfiled Odense	55.47204849	10.41470375	Per Breum	-||-
5-11-18	14.38	1	Model airfiled Odense	55.47204849	10.41470375	Per Breum	-||-
5-11-18	14.58	1	Model airfiled Odense	55.47204849	10.41470375	Per Breum	-||-
7-11-18	16.13	5	Model airfield, Årslev	55.309922	10.463653	Per Breum	Test description: https://docs.google.com/document/d/1BjVSPbpKzhScymIwZ22qHvmxsEh0CZnistG4L5lqzDI/edit?usp=sharing
7-11-18	16.24	4	Model airfield, Årslev	55.309922	10.463653	Per Breum	-||-
11-11-18	14.23	8	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://docs.google.com/document/d/1ZgpME-GWFuWCG9hcZDZGh9mrgHerfE2Cy-rvb_oYl4c/edit?usp=sharing
12-11-18	13.23	2	Model airfiled Odense	55.47204849	10.41470375	Per Breum	https://docs.google.com/document/d/1a2vNDtC_O0WNj2zXKEDEVPlnVsjctleZfCVGuyZlBX4/edit?usp=sharing
12-11-18	13.34	2	Model airfiled Odense	55.47204849	10.41470375	Per Breum	https://docs.google.com/document/d/1a2vNDtC_O0WNj2zXKEDEVPlnVsjctleZfCVGuyZlBX4/edit?usp=sharing
28-11-18	12.34	2	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://drive.google.com/open?id=1OdiaY9TT7EfEZBQbrdAoXlfSbkUexVcliJVEiiBAdXU
28-11-18	12.55	2	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://drive.google.com/open?id=1OdiaY9TT7EfEZBQbrdAoXlfSbkUexVcliJVEiiBAdXU
28-11-18	12.00	8	Model airfield, Årslev	55.309922	10.463653	Per Breum	Hover test to obtain video with rosbag
3-12-18	12.30	7	Model airfiled Odense	55.47204849	10.41470375	Per Breum	Hover test to test precision landing controller - no results, need to rearrange LEDs
5-12-18	11.30	13	HCA airport, SDU hangar	55.47037	10.329436	Per Breum	Precision controller test attempt. No results, GPS is drifting to much to trust the drone to stay away from the walls
5-12-18	13.30	12	Model airfield, Årslev	55.309922	10.463653	Per Breum	Precision controller test. Rosbag recorded. The drone controlled the wrong way - need to correct this in the controller
6-12-18	14.00	15	Model airfield, Årslev	55.309922	10.463653	Per Breum	Testing precision landing controller. Drone is being controlled, but the system seems to be unstable with increasing oscilliations

7-12-18	08.30	5	Model airfield, Årslev	55.309922	10.463653	Per Breum	Full SIMPLE system test without precision landing https://docs.google.com/document/d/1QVhILVrSn1oLI_FOdEAQp0RPdJQ-xWpyHqhBJLVfDLo/edit?usp=sharing
9-12-18	12.00	24	Model airfield, Årslev	55.309922	10.463653	Per Breum	https://docs.google.com/document/d/1sz0amW6yK1PYKAuyUpaqRKQ26xVQXa3iwVlIFPKa64s/edit?usp=sharing
10-12-18	10.00	20	UAS Denmark testfield, HCA Airport	55.472288	10.325293	Mathias/Per	https://docs.google.com/document/d/1akdzUaQ_Ik1DeZF91caPIf2wP7xWGi81RFPg_P4whIk/edit?usp=sharing

B Group contract

- What are your expectations for this project?
  - Get an autonomously working drone that can fly BVLOS
  - 2-3 meetings pr week
  - Small groups doing individual tasks independent of each other.
  - Decent work hours
  - Create a complete system.
  - Develop good professional and personal relations to groups members you have not worked with before.
  - Good teamwork
  - Evenly distributed workload.
  - Develop project management skillZ
  - Grow in collaboration skills with "random groups"
  - Become an Engineer
  - Dividing the work into small tasks.
  - Discussions about implementation before starting working on it.
  - Working on something new, instead of something you are good at
- What is your level of ambition?
  - We want SORA and BVLOS
  - Have a cool roll up
  - Prioritizing a good report
- How much time are you willing to invest in the project?
  - 16 hours a week, consistently
- How much time do you expect the other team members to invest in the project?
  - 16
- How should the work be shared between the team members?
  - Personal choice. Work in groups with mixed strengths and weaknesses.
- Should the team select a team leader or not?
  - We don't have a team leader but we choose a project manager who will be in charge of the meetings.
- How should the team make decisions?
  - Democratically
- What should be the normal working hours for the team?

  - Monday 8-16
  - Thursday 12-16
  - Small groups decide when to meet.

- What should a team member do if he does not show up one day (legal absence)?
  - Contact the group as fast as possible. If you are late or break something bring beer/soda. If you break anything vital you should bring breakfast.

- What are the consequences of a team member not performing his tasks?
  - Will be brought up if it's gonna be a problem

- What are your concerns regarding the project?
  - Time limit
  - concerns
  - A lot of technical concerns

- What are your expectations regarding communication within the team (email, sms etc.)?
  - Messenger is fine for small stuff such as being late.
  - Slack for project management
  - Git for the project
  - Latex for the report.
  - Are there any other topics which you think should be included in the contract
  - We will clean the lab after ourselves.
  - Soft deadline for when the drone should be finished December the 3.88

C Battery Test

When flying is it important to know that you have enough battery to complete your mission. As this depends on many factors such as wind, payload, air pressure etc. an estimate should suffice. This estimate comes from a test at the model airfield in Odense, in this test a mission plan is created that is 350 meters long, and has 3 turns as to make almost a complete square. The battery voltage is measured each time a new plan is initiated and the voltage is recorded when it lands again. This recording is redundant as the log from the pixhawk will also have the information but if anything should go wrong it is nice to have, and it does not require much time to do. The test is also carried out at three different velocities set through QGroundControl at 5, 10 and 15 m/s.

Time	Start voltage	Final voltage	Distance on full (meters) full[mAh] / used[mAh] * dist[m] = expected distance
14.18	16,7	16.2[1]	4250/910*383 = 1789m
14.33	16,3	15.7[2]	4250/763*390 = 2180m
14,41	15,7	15,7[3]	4250/610*265.1 = 1846m

[1] https://logs.px4.io/plot_app?log=1aad1f9e-3513-4855-88ff-1fdd019790f4
[2] https://logs.px4.io/plot_app?log=9e9fa486-85b7-4008-977d-91a8217e44ae
[3] https://logs.px4.io/plot_app?log=5882317f-daab-41cd-bb4e-1e3c67cae560

D Docking Station Landing and Design

RMUASD Vision based Landing methods

Requirements

- Operational at both day and night
- Needs to be robust enough to be detected from a distance.
- As decided, must use an off board camera/processing to land
    - Simple, lightweight must be used on the drone.

Hardware and filters:
40x IR 940nm diodes (40 dkk)
https://dk.rs-online.com/web/p/ir-lysdioder/8187663/
1x 740nm + filter (11.50 dollars, 74 dkk)
https://www.amazon.com/gp/product/B003TY10AS/ref=as_li_ss_tl?ie=UTF8&psc=1&linkCode=sl1&tag=brandonlosa0c-20&linkId=3ddc0437465b86d188496e821bbf6bd8

In total: 114 dkk

"Ideal" filter - 883nm bandpass (43 euros, 321 dkk)
https://www.edmundoptics.com/p/rg-9-ir-254mm-dia-colored-glass-bandpass-filter-/27898/
"Ideal" IR LED 870nm 50mW (7.3dkk pr/pcs.)
https://dk.rs-online.com/web/p/ir-lysdioder/6997644/
IR LED 870nm 780mW (12 dkk per piece, 56 dkk per package [5 in])
https://dk.rs-online.com/web/p/ir-lysdioder/8947679/

BEC IN:7V-23V, OUT: 3.3V/3.5A (26 dkk)
https://hobbyking.com/en_us/x3-pro-3-3v-3-5a-ubec-2-5s-lipoly-7-2-21v.html

Method one: Selective light detection
This method will use light outside the visible spectrum (either IR/UV) to act as a beacon on the drone. Ideally, the camera will use a bandpass-style filter to only show light of the specific wavelength, such as a diode mounted onto the drone.
Things to consider:

- Will solve night time detection, as an active light source can be used.
- Daytime however, is a different factor due to sun illumination.
- Cost: filters can be very expensive (€100+) for good quality
    - https://www.edmundoptics.com

https://photography.tutsplus.com/tutorials/an-in-depth-guide-to-infrared-photography-setup-and-capture--photo-9533
Source:
https://publik.tuwien.ac.at/files/PubDat_210294.pdf

From talking to Lecturer (HSM), it seems that the best thing to do is to use a band pass filter that corresponds to the LED.

E UTMS Test Planning

Test Plan for 08/12/18

Test #	Subsystem Testing	Test intent	Method of test	Expected result	Actual result
1	UTM system	Check for traffic being published to proximity node	Use simTraffic script to position a drone within 100m of the UAS	Sim drone ID 911 to be published to **“/utmNode/proximityTraffic”**	Sim droneID: 911 was displayed
2	UTM system	Check critical alarm is published	Use simTraffic script to position a drone within 15m (horizontal) and 5 Metres Vertical	“data:true” published to node **“/state/close_encounter”**	A true signal was sent
3	UTM system	Check Dynamic No-Fly-Zones (NFZ) are published when requested	Manual injection of a “data:1” to the /UTMrequest topic	List of dynamic NFZ posted to **“/utmNode/NoFlyZoneList"**	The list was only submitted when requried.
4	UTM system	Utm service - deconflict integration	Check that the UTM server is posting current positional data and next waypoint information	Current and next waypoint posted to **“utmNode/uasPos”**	The data is shown when the UAV is airbourne, next waypoint set to 0,0,0 when landed.

F Product Backlog

Color ID (assigned to most similar)	ID	Color ID	Task Description	Priority (1-5)	Completed	Ready for sprint	Add. Descr.	Start date	End Date	Date added
1 = Drone	1		Finish framework for product backlog on Trello	2	☑	☑	☐	06/09/2018	09/09/2018	03/09/2018
2 = GCS	2		What is the maximum distance the telemetry reach	3	☑	☑	☐	10/09/2018	10/09/2018	03/09/2018
3 = Docking St	3		Setup GCS OS	1	☑	☑	☐	07/09/2018	08/09/2018	05/09/2018
4 = UI	4		Acquire a laptop	1	☑	☑	☐	06/09/2018	08/09/2018	03/09/2018
5 = Report	5		Solutions for guided landing (basic)	1	☑	☑	☐	06/09/2018	10/09/2018	03/09/2018
6 = Other	6		Create Block diagram of the system	1	☑	☑	☐	06/09/2018	10/09/2018	03/09/2018
	7		Explore Pros/Cons of Companion computer	1	☑	☑	☐	06/09/2018	06/09/2018	03/09/2018
	8		Aquire two raspberry pi's	1	☑	☑	☐	12/09/2018	12/09/2018	06/09/2018
	9		put Ubuntu mate on two Pi's	1	☑	☑	☐	12/09/2018	13/09/2018	06/09/2018
	10		Can link between drone and gcs be switched during the flight? (radio -> gsm)	2	☑	☑	☐	10/09/2018	10/09/2018	10/09/2018
	11		Is it possible to get a 3/4G module for the drone?	3	☑	☑	☐	06/09/2018	10/09/2018	06/09/2018
	12		Tuning Pixhawk	2	☑	☑	☐	14/09/2018	15/09/2018	04/09/2018
	13		Research cameras for docking station	1	☑	☑	☐	16/09/2018	17/09/2018	06/09/2018
	14		Research on selective light detection for landing	1	☑	☑	☐	10/09/2018	17/09/2018	06/09/2018
	15		Prepare for autonomous/manual flight monday 17th	2	☑	☑	☐	17/09/2018	17/09/2018	06/09/2018
	16		How to make an IR filter	2	☑	☑	☐	13/09/2018	17/09/2018	06/09/2018
	17		Setup ROS on GCS	2	☑	☑	☐	13/09/2018	13/09/2018	06/09/2018
	18		Setup apache server	2	☑	☑	☐	13/09/2018	20/09/2018	06/09/2018
	19		Setup MAVLINK on PC	2	☑	☑	☐	18/09/2018	20/09/2018	06/09/2018
	20		Necessary mavlink protocols (research)	2	☑	☑	☐	18/09/2018	20/09/2018	05/09/2018
	21		Define tasks (initial)	1	☑	☑	☐	06/09/2018	04/10/2018	03/09/2018
	22		aquire camera's	1	☑	☑	☐	27/09/2018	27/09/2018	06/09/2018
	23		Code UI on apache server	3	☑	☑	☐	13/09/2018	20/09/2018	06/09/2018
	24		Create overview of states for the drone	2	☑	☑	☐	23/09/2018	27/09/2018	20/09/2018
	25		create script with basic drone operation	1	☑	☑	☐	17/09/2018	26/09/2018	16/09/2018
	26		COM to docking	2	☑	☑	☐	20/09/2018	20/09/2018	05/09/2018
	27		COM to drone	2	☑	☑	☐	17/09/2018	27/09/2018	05/09/2018
	28		create service to handle test script to manually set local se	1	☑	☑	☐	09/10/2018	10/10/2018	09/10/2018
	29		Create keypress options in terminal GCS.	3	☑	☑	☐	09/10/2018	11/10/2018	09/10/2018
	30		create activate offboard script	1	☑	☑	☐	07/10/2018	09/10/2018	07/10/2018
	31		create terminal node to display gcs information	3	☑	☑	☐	08/10/2018	11/10/2018	08/10/2018
	32		UI Server design	3	☑	☑	☐	27/09/2018	27/09/2018	05/09/2018
	33		UI server setup	3	☑	☑	☐	27/09/2018	27/09/2018	05/09/2018
	34		UI ability/attributes	3	☑	☑	☐	27/09/2018	27/09/2018	05/09/2018
	35		Create script to convert between local and global coordina	2	☑	☑	☐	17/10/2018	17/10/2018	11/10/2018
	36		Create Roll up design	2	☑	☑	☐	13/09/2018	04/10/2018	04/09/2018
	37		Setup ROS on Pi's	2	☑	☑	☐	13/09/2018	27/09/2018	06/09/2018
	38		Order Hardware for landing system	1	☑	☑	☐	13/09/2018	20/09/2018	06/09/2018
	39		System diagram for guided landing (LED and camera)	2	☑	☑	☐	13/09/2018	20/10/2018	10/09/2018
	40		Setup Pi with Pi cam	3	☑	☑	☐	27/09/2018	20/10/2018	10/09/2018
	41		Brainstorm (initial for whole project)	1	☑	☑	☐	03/09/2018	04/09/2018	03/09/2018
	42		Create mission handler	1	☑	☑	☐	17/10/2018	01/11/2018	09/10/2018
	43		Guidelines for Node structure	1	☑	☑	☐	24/10/2018	25/10/2018	27/09/2018
	44		Implementing vision based guided landing system	1	☑	☑	☐	20/09/2018	04/11/2018	20/09/2018
	45		Setup Gazebo for simulation	2	☑	☑	☐	17/09/2018	24/10/2018	10/09/2018
	46		UTM - first setup with basic get/set request	2	☑	☑	☐	26/09/2018	25/10/2018	26/09/2018
	47		Define 'cutting surface' for GCS	1	☑	☑	☐	16/09/2018	20/09/2018	05/09/2018
	48		COM to UTM	2	☑	☑	☐	27/09/2018	25/10/2018	05/09/2018
	49		Measure current draw from drone to estimate flight time (e.g. use alternative)	2	☐	☑	☐	13/09/2018		04/09/2018
	50		UTM - filter list of drones based on X radius input	1	☑	☑	☐	4/10/2018	31/10/2018	26/09/2018
	51		UTM - create script to send our drone position + other data	1	☑	☑	☐	4/10/2018	25/10/2018	26/09/2018
	52		Monitor node (health status)	2	☑	☑	☐	11/10/2018	1/11/2018	11/10/2018
	53		Construct draft for report structure	3	☐	☑	☐			04/09/2018
	54		Create/start SORA application	1	☑	☑	☐	25/10/2018	12/11/2018	04/09/2018
	55		Upload/set parameters on drone via mavlink	5	☐	☑	☐			30/09/2018
	56		LED control circuit	3	☑	☑	☐	19/11/2018	26/11/2018	06/09/2018
	57		Preliminary test with IR and OpenCV	3	☑	☑	☐	19/11/2018	20/11/2018	10/09/2018
	~~58~~		~~Test landing through mavlink over long distance~~		☐	☐	☐			06/09/2018
	59		Testing the influence of the magnetic field from the box mount on the pixhawk (error of orientation)	3	☐	☑	☐			13/09/2018
	60		Upload whole mission through mavlink	5	☐	☑	☐			26/09/2018
	61		Generate Template for flight log	1	☑	☑	☐	23/10/2018	24/10/2018	08/10/2018
	62		Mount parachute module	1	☐	☐	☐			05/09/2018
	63		COM to user	2	☐	☑	☐			05/09/2018
	64		aquire two 3G modules	5	☐	☑	☐			06/09/2018
	65		Identify all needed failsafe measures needed for this appli	3	☐	☑	☐			05/09/2018
	66		Design docking stations (idea generation and mechanical	2	☑	☑	☐	21/9/2018	30/11/2018	06/09/2018

G Contribution matrix

	Per	Mads	Oscar	Blaze	Thor	Mathias
User interface		60			33	7
Main state handler	10	35				55
UTM			100			
Deconflict		50				50
Path planner		35			65	
Mission handler	100					
Drone command handler	90					10
Presicion landing			25	75		
Simulation	50		50			
Docking station	7	7	65	7	7	7
Project management	70	5	5	5	10	5
SORA	5	20	7			68
Rollup	5	5	15	65	5	5
Report	17	17	17	15	17	17
Field test	50	9	8	11	12	10

H SORA

Mathias Witten Madsen | Per Breum | Oscar Schofield
Mads Tilgaard Jensen | Błażej Banaszewski | Thor G. Jensen

Specific Operations Risk Assessment

Application for BVLOS flight by SDU UAS Center

SDU UAS Center 2018

1. Introduction

This document contains a Specific Operations Risk Assessment (SORA) and associated documentation to be presented to the Danish Civil Aviation Authority (CAA) Trafikstyrelsen (TBST).

The content of this document is based on the draft JARUS guidelines on Specific Operations Risk Assessment (SORA) for external consultation, Edition 1.2 dtd 31.05.2018.

2. Concept of Operation

This Concept of Operation (ConOps) describes operation of an fixed-pitched multirotor aircraft performing the task of transporting simulated blood samples between two docking stations.

The UAS will be operated and piloted by the SDU UAS Center, HCA Airport (EKOD) in Odense, Denmark. Specifically, the operation area is within the restricted air spaces EK R OD1 respecting a 2.5 NM horizontal separation as defined in Appendix B (and map figure B.1).

3. Technical information

4. The SORA process

4.1 Pre-application evaluation

4.2 Step #1 – ConOps description

Operation relevant information

A.1.2.1 Safety

The testing is conducted at HCA Airport which means that the airspace can be closed and thus taking care of the safety in the air

A.1.2.3 Training of staff involved in operations

The proposed flight will be autonomous. A pilot without a drone licence is holding the transmitter, having a constant RC signal available, with which the UAV operations can be terminated at all times using the this.
suggested rephrasing
The proposed flights will focus on testing the autonomous operations of the UAV, and as a result a pilot will be required to monitor the craft at all times. The pilot will have the correct drone license and will be in possession of a Remote Control (RC) transmitter which will allow for termination of the autonomous flight should a critical event occur.

A.1.2.4 Maintenance

(a) The UAS is inspected and assessed for any faults before and after every flight.
(b) If an inspected component shows signs damage or failure, the part is replaced.
(c) The UAV is powered off and the damaged part is replaced

A.1.2.5 Crew

The crew for the proposed operations consists of 3 roles:

1) External Pilot - the crew member launches the UAV remotely through the Ground Control Station (GCS) and monitors the telemetry during operation. The role also requires continuous contact with the remote observers whilst providing feedback to the safety pilot.
2) Safety Pilot - this role involves being in constant possession of the RC transmitter and being prepared to disable autonomous operations should problems arise.
3) Remote Observers - these crew members are responsible for providing visual feedback of the UAV status once it exceeds the safety pilots line of sight (LOS). The Observers are positioned in proximity to the expected flight plan and are required to be in constant communication with the external pilot.

A.1.2.6 UAS Configuration Management

If any changes to the design takes place all the necessary changes is applied to the software of the FC.

A.1.2.7 Other position(s) and other information

All members of the crew are students connected with the University of Southern Denmarks' TEK Faculty.

A.1.3.1 Type of operations

The operation will perform a beyond visual line of sight (BVLOS) flight of a unmanned aerial vehicle (UAV) provided by SDU. The UAV has been modified by the team described in [A.1.2.5] to carry blood samples between two docking stations. The UAS is intended to fly at HCA Airfield within which the airspace can be closed, and no people present besides the crew. The test would be conducted on a day with little to medium wind. The UAS is expected to fly 2 kilometres during the test, with a C2-telemetry link established at all times to the GCS. The GCS is connected to a UTM server, which provides information of fictional traffic (provided for testing purposes) within the operational area, in addition to static and dynamic “no-fly zones”.
The crew will place the UAS on a docking station, by pressing a button on a phone, it will pick up a parcel with blood samples and head to the waypoints provided by the GCS which the crew monitors at all times.

A.1.3.2 Standard Operating Procedures

A reference to the applicable operations manual (OM) is acceptable. **(Where to find this?)**

A.1.3.3 Normal Operation Strategy

The operation is to be conducted at HCA Airport within a closed airspace, which results in no external air traffic being allowed to fly within the vicinity of the UAV. The perimeter of the Airport is also surrounded by a fence which requires authorisation for access. The combination of both closed airspace and restricted ground access can be assumed that the operational area is controlled and safe. The UAS will be controlled in “Offboard node” which means that the operator has control at all times through the GCS. When travelling through the airspace, the UAS continually receives new coordinates on where to move, which means it can be dynamic and the drone has a fast response time.

A.1.3.4 Abnormal operation and emergency operation

During operation, the UAS system monitors the communication for issues and can utilise three possible failsafes should a problem arise. Firstly, in the case of temporary loss of offboard mode, the UAV will hold its current position and will assume a timeout phase for communication to be restored.
If this timeout phase is exceeded, the UAV has two options: if an RC connection is still available, the drone will fly to the last transmitted rally point and perform a landing - also known as “return to home”.
Alternatively if no RC connection is available, the drone will simply perform a landing at it current location. In the event that of a critical failure, where uav is unable to maintain its attitude and position - a parachute is deployed and the motors are disarmed, reducing the descent speed and minimising damage.

A.1.4 Training

A.1.4.1 General information

All crew is from SDU and the RC operator have more than 2 hours flight time with similar UAV. The drone has been tested using Software-in-the-loop (SITL) and a simulator to check the software works as expected, also providing the crew some information about how the drone should operate.

A.1.4.2 Initial training and qualification

No official training has been undertaken by the crew; all proficiency and experience has been self-taught.

A.1.4.3 Procedures for maintenance of currency

No procedures are in place to ensure that each member of the crew has been trained to do one specific task, instead the whole team is involved in the development phase of the UAS and thus have an insight as to the UAV operation.

A.1.4.4 Flight Simulation Training Devices (FSTD)

The operation is conducted automatically and no manual control is to be done, thus there is no need for any FSTD, as the safety systems will take control if errors occur.

A.2 Technical relevan

t information

A.2.2 UAS descriptionA.2.2.1 Unmanned Aircraft (UA) segment

A.2.2.1.1 Airframe

1. Physical characteristics
    a. Dimensions
        Body: 40x40x15 cm
        Propeller: 20 cm
    b. Mass
        Mass: 2 kg
        Maximum Takeoff Mass: 6 kg
    c. Center of mass
        In the center of the FC
2. Materials
    a. UAV
        i. Main body: Fiberglass Printed Circuit Board (PCB)
        ii. Arms: Fibreglass composite
        iii. Propellers: Plastic

3. Loads
    a. No real test has been carried out with substantial loads and thus not enough data is available to show a flutter-free flight under high stress
    b. N/A
4. Sub-systems
    a. Parachute module that has been tested and will trigger if the in-built Inertial Measurement Unit (IMU) detects the UAV is in freefall.
    b. A camera module is incorporated into the docking station. The module pushes images to the GCS, and is used for precision landing.
    c. A analog representation for the blood samples container is used, consisting of a 3D printed square box.

A.2.2.1.2 Aircraft Performance Characteristics

1. Performance of the aircraft
    a. Performance:
        i. Maximum altitude: 100 m
        ii. Maximum endurance: 30 min
        iii. Maximum range: 2 km
        iv. Maximum rate of climb: 5 m/s
        v. Maximum rate of descent: 10
        vi. Maximum bank angle: 40 degree
        vii. Turn rate limits: 10 cm
    b. Airspeeds:
        i. Slowest speed attainable: 0 m/s
        ii. Stall speed (if applicable): N/A
        iii. Nominal cruise speed: 10 m/s
        iv. Max cruise speed: 13 m/s
        v. Never exceed airspeed 30 m/s
2. Performance limitations
    a. Wind speed limitations
        i. Headwind: 10 m/s
        ii. Crosswind: 10 m/s
        iii. Gusts: 17 m/s
    b. Turbulence restriction: N/a
    c. Rain, hail, snow, ashes resistance or sensitivity: 179 days of rain average 765 mm rain[1][2]. Snow and hail is generally not a problem.
    d. Minimum visibility conditions, if applicable: N/A
    e. Outside Air temperature (OAT) limits: Max weather ever recorded in Denmark is 36.4[3] degrees. The average in the summer is 20 degrees[4]
    f. In-flight icing:
        i. No icing is presumed to be formed on the UAV
        ii. No detector is mounted

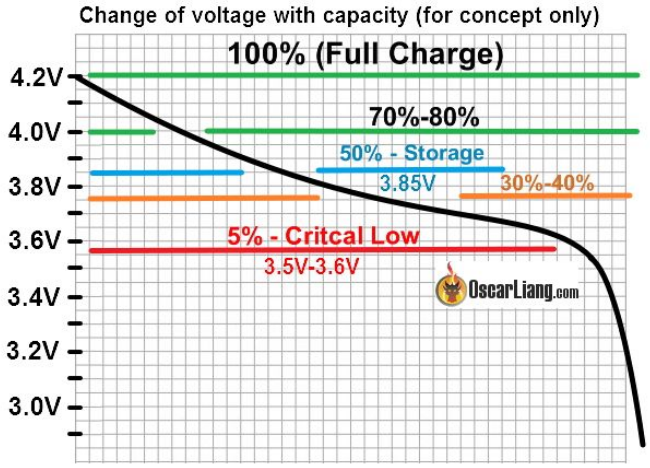
[1] https://www.dr.dk/nyheder/vejret/grafik-saa-meget-regner-det-i-danmark-paa-et-aar
[2] https://www.dmi.dk/klima/klimaet-frem-til-i-dag/danmark/nedboer-og-sol/
[3] https://samvirke.dk/artikler/se-de-danske-vejrrekorder
[4] https://www.dmi.dk/vejr/arkiver/normaler-og-ekstremer/klimanormaler-dk/

iii. No protection is provided

A.2.2.1.3 Propulsion System

1. Principle
    a. The system consists of 6 Brushless DC (BLDC) motors symmetrically placed on the UAV. These are powered by a 4 cell LiPo battery. The Flight Controller FC, together with a GCS controls the attitude of the multirotor.
2. Fuel-powered propulsion systems
    a. N/A
3. Electric-powered propulsion systems
    a. The BLDC motors are regulated by Electronic Speed Controller (ESC), which controlled by the FC through the use of Pulse-Width Modulated (PWM) signals. The power to the ESC and motors is provided through a Power Distribution Board (PDB), splitting the power from the 4S battery into 6 parallel outputs. This PDB also has an integrated BEC which provides 5 Volts to the FC.
    b. It is a BLDC motor [KV] controlled by an ESC
    c. 6 motors are installed
    d. 75 continuous motor output [Watt]
    e. 288 peak motor output [Watt]
    f. 0-24 Current range of motors [Amps][5]
    g. Explained in (a.)
    h. The battery is [mAh] and has a C rating og [C-rating] meaning that it is possible to provide [MAX Amps] to the system [system level diagram here]
    i. The system draws all its power from a 4S battery and no new power is generated
    j. The battery is charged up to 16.8V and can be discharged all the way down to 14.8V. The performance of the battery can be seen on the figure.[6] According to a test while to done is flying at 10 m/s[7] the battery lifetime is approximately 7 min. [but this seems very small]

[5] c - f is from this flighlog https://review.px4.io/plot_app?log=49f1270a-e334-4da8-b228-c9609610f550
[6] https://oscarliang.com/lipo-battery-guide/
[7] https://docs.google.com/document/d/1NaMf755UNPBCMEbVKZ0ez5Grdh6Xehr4dfRUjBfi7Fo/edit?usp=sharing



k. To monitor the power system, a current monitor is placed in series with the battery and the PDB. The power readings are sent through the C2 link.
l. No backup power is provided
m. No direct motor monitoring is provided at this time. But the IMU is monitored for abnormal activity and acted upon.
n. [not sure about this one]
o. It would be catastrophic if the battery dropped below critical as it would quickly deplete from this moment and the drone could experience freefall. Also if a wire were cut mid flight for whatever reason the UAV would also become uncontrollable.
p. what safeguards are in place for propulsion system loss for the following:
    i. **Low battery:** The UAV “returns to home” which is described earlier
    ii. **Failed signal input from the control station:** [No safeguards?] (The UAV would loiter)
    iii. **Motor controller failure:** Either “return to home” or if uncontrollable a parachute would deploy.
q. There is no in-flight reset capabilities for the motors.

A.2.2.1.5 Sensors

The sensors within the UAS are the used for controlling the stability of the multirotor. They consist of an IMU, compass, altimeter and GPS. All sensors, with exception to the GPS, are placed internally within PX4 FC and tested by the community. The GPS is standard pixhawk equipment and also tested by the community.

A.2.2.1.6 Payloads

A square box is attached to the underside of the UAV, this box weighs [400 gram] and is dimensioned [15x15x15] cm.

A.2.3 UAS Control segment

A.2.3.1 General

A Ground Control Station (GCS) have been implemented that handles all the control of the drone. This station gets information from the telemetry link regarding the UAS status,

including all the sensor data, battery status and position. From this it is possible to know where the drone is at all times. [describe the location of the sensors compared to the FC].

A.2.3.2 Navigation

1. The UAS has a standard GPS and uses this to determine its location.
2. the UAS uses waypoints provided by the GCS in order to fly between the docking stations.
3. On arrival at a docking station, the UAS has a rough idea of where it needs to land. Control is then taken over by the docking station, using computer vision to provide positional feedback and accordingly move the UAV.
4. How does the operator or UAS respond to instructions from:
    a. A UTM server is available. Information from this server can be pulled and contains data from the airspace and other traffic in the area. Information about the UAV is published to the UTM server for others to use in the same manner.
    b. No visual observer is present as it is a BVLOS flight.
    c. Crew is present at both docking stations to report successful land and takeoff. The crew are also for changing the battery as that feature is not implemented in the docking stations.
5. The altimeter is set according to a GPS and an altimeter. No other test is present.
6. A node monitoring the signals from the drone is in place, if anything non-regular is sensed the system goes into failsafe and lands at a designated location. If this is not possible it slowly descends where it is.
7. No backup for the navigation is provided other than a compass that can orientate the UAV.
8. The parachute will deploy as the UAV has encountered a critical state.

A.2.3.3 Autopilot

1. The autopilot is developed using MAVLink Lora, a branch of MAVLink, that has less overhead as it only uses the packages needed to do a specific task. The path planner is developed using a visibility graph from the no-fly zones and establishes a path around these.
2. The autopilot is developed in-house
3. The autopilot is tested in a simulator called gazebo using Software In the Loop (SITL) and later tested in the field on hardware.
4. No limit observation is implemented, but as the mission is fully controllable from the GCS, it is not necessary.
5. Both Software and Hardware (SITL/HITL) is used when testing the drone. First intense testing in SITL then taken to hardware later.

A.2.3.4 Flight Control System

1. Multirotor, no control surface is available N/A

2. The UAV goes through multiple stages as seen figure 1

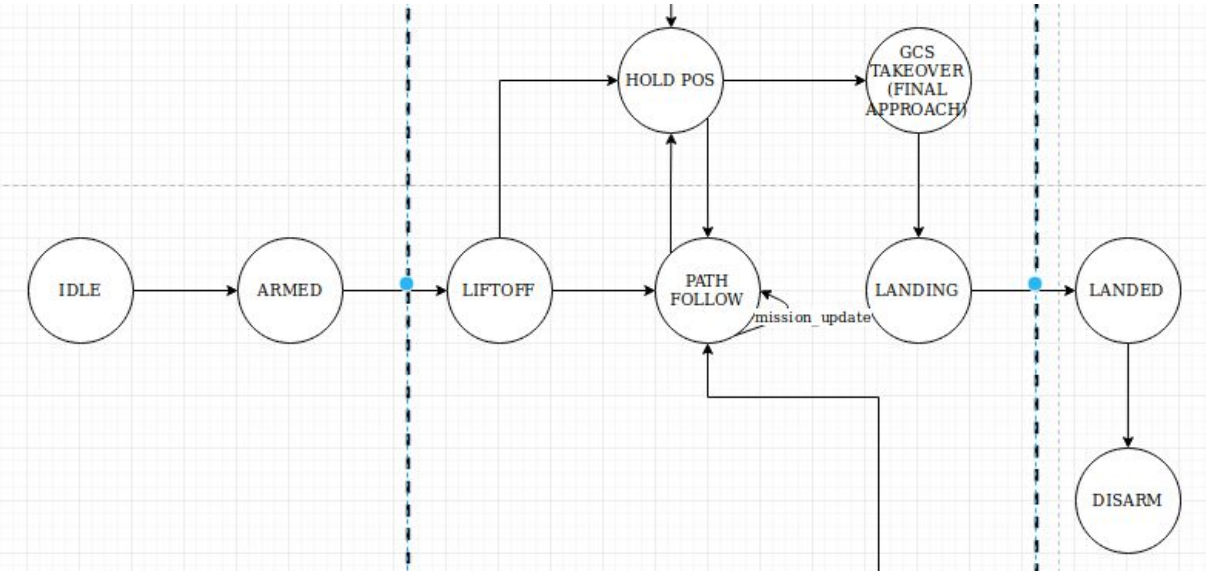


Figure 1: UAV states from idle to flight to landing

3. Flight Control Computer/autopilot:
    a. Only the kill switch is present as an auxiliary input. All other control is happening through software. No extra protection is around the kill-switch as this will have to be removed before it could be toggled.
    b. All documentation is present at https://mavlink.io/en/.
    c. The flightstack is a PX4 based on a pixhawk2.1 flight controller.

A.2.3.5 Control Station (CS)

1. N/A
2. N/A.
3. A UTM server is available through the GCS that has all the info about the surrounding airspace that is relevant for a BVLOS mission.
4. N/A
5. N/A
6. When a conflict appears to the UAV, the GCS will determine what action to take. This response is based on a priority list, which contains, but not limited to, what payload is carried, the speed or the purpose of the flight. The UAV will do conflict avoidance by either stopping or setting a different flight level.
7. 8. 9. N/A

A.2.4 Geo-fencing

a. The Geofencing follows automatic from the visibility graph of the pathplanner

A.2.5 Ground Support Equipment (GSE) segment

a. The most relevant GSE is the docking stations, for which there are two used within the system. These docking stations comprise of a level landing surface, an integrated Pi camera with an IR filter, making it possible to detect infrared light in a specific color spectrum, namely 860nm. The docking station also has an Raspberry Pi installed with Robot Operating System (ROS), that publishes to the central GCS. Image

processing is undertaken on the raspberry pi, with the position corrections published to the GCS for precision landing.

b. The UAV gets is power through a battery, this battery is switched and recharged manually according to its charge C-rate.
c. The standard equipment consists of: a docking station containing a Raspberry Pi (with ROS) and a PiCam.
d. The UAS is picked up after a mission and transported in the back of a van or carried by hand.

A.2.6 Command and Control Link (C2 link) segment

a. The C2 link complies with the ETSI EN300 220 7.2.3 standard and transmit with a maximum freq of 434790Hz, and a min of 433050Hz.
b. Two telemetry modules are used in the system. One connected to the GCS and one at the UAV. both modules operate as transceivers, with the drone-situated (C2D) is transmitting data from the FC (IMU, sensor, battery and GPS data). The GCS is then using this data to monitor the health of the UAV. The telemetry module at the GCS (C2G) is continuously transmitting waypoints and other message types to the UAV that the MAVLink can interpret and lease to the FC resulting in the UAV head to this waypoint. The C2 link has an air speed of 64kbps, and the latency between two messages is maximum 1 sec according to tests[8].
c. Describe the control link(s) connecting the UA the CS Specifically address the following items:
    i. The frequency spectrum is between 433050Hz - 434790Hz[9] with the telemetry link we are using. The max power is 100 mW[10]
    ii. The signal processing is gaussian frequency-shift keying. For security the base frequency is skewed up to one channel based on a random seed generated from the NETID, which should be the same on both devices.
    iii. The drone uses about 31% of the bandwidth at the maximum distance from the GCS. This was measured by monitoring the data rate to and from the drone.
    iv. It is possible to measure how often you receive data on the C2 link, we also know how much data is expected. From this it is only necessary to to set a threshold of a critical value, this threshold is obtained from tests that are yet to be done, but a starting point is 80% in a period of 2 seconds because of the possible latency. This data can then be displayed in a table with other useful information to the operator.
    v. No redundant C2 link is established.
    vi. N/A

8 https://docs.google.com/document/d/1ZgpME-GWFuWCG9hcZDZGh9mrgHerfE2Cy-rvb_oYI4c/edit?usp=sharing
9 https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/02.04.01_40/en_30022001v020401o.pdf
10 http://ardupilot.org/copter/docs/common-3dr-radio-advanced-configuration-and-technical-information.html

vii. What system is in place to prevent data-link loss?

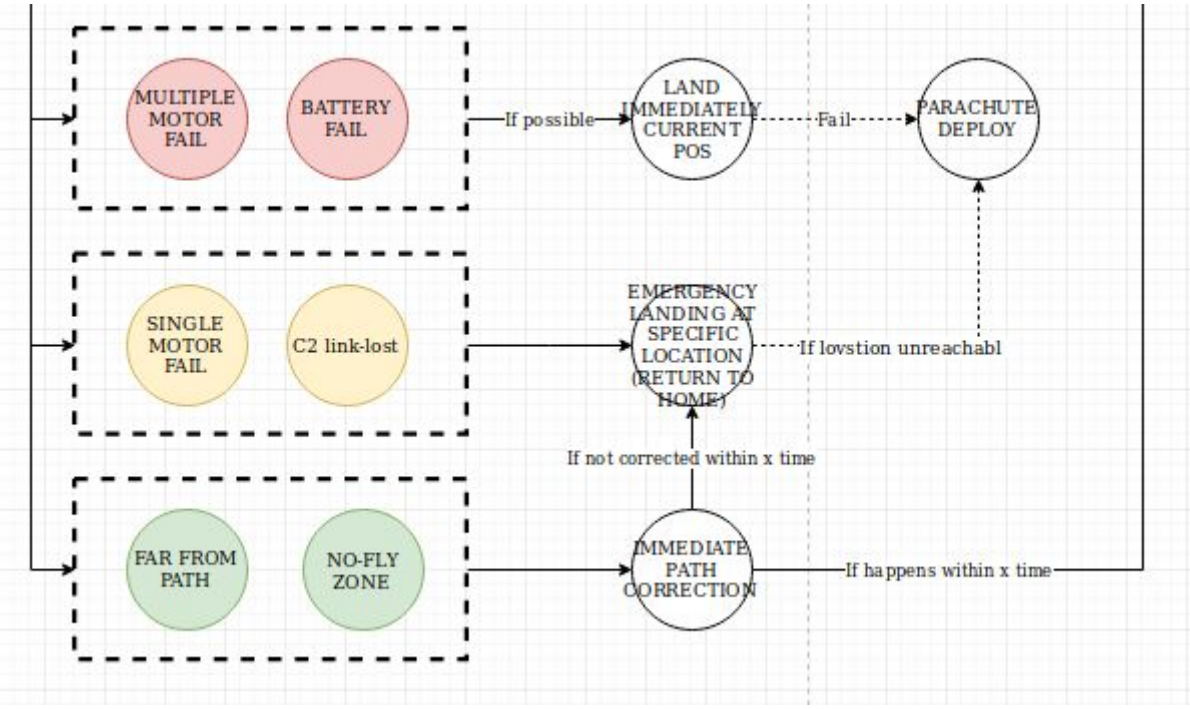
1. Three different checks are install according to the standard stated: the minimum number of channels, the maximum dwell time on a channel and the return time to a channel.
2. 3. 4. 5. 6. All er N/A. The way we handle loss of C2 link is to define the states as failsafe and the FC has ways to handle these, described earlier.

A.2.7 C2 Link degradation

a. What are the measures in case of a link degradation?
   i. The mission can still continue and it will do so.
   ii. A status telling the operator that the signal is weak is displayed in the terminal

A.2.8 C2 link lost

a. If there is no communication between the two modules for more than 3 sec the link is deemed lost.



b. What are the measures in case of loss of the C2 link?
   i. Multicrew communication is in place and the operator calls out to the team that the link is lost.
   ii. If the link is lost, it is seen as a yellow class emergency resulting in the middle failsafe. If the link is established within a certain time it will continue the mission, else it will try to land at a specific location.
   iii. The “return to home” location is always in a straight line from the drone, without any no-fly-zones, in between.

A.2.9 Safety

The system is designed to fly BVLOS, and for this reason the UAV is equipped with a parachute module that detects unexpected behaviour, such as a freefall. The safety system proceeds to kill the motors and deploy its parachute, reducing the descent speed and reducing the likelihood of damage. The drone is also fully under the control of a GCS via what is called “Offboard mode” this means that new waypoints or course corrections can be received and carried out in an instance.

4.3 Step #2 – Determination of the initial UAS Ground Risk Class

When the drones reaches maximum velocity toward the earth the drag force will be equal to the force from the gravity.

$$Drone\ mass * gravitational\ Force = 1/2 * Airdensity * Drone\ velocity^2 * DragCoefficient * Drone\ area$$

The drones mass is 2 kg, the air density is estimated to be 1.2kg/m^3 with a drag coefficient is estimated to be 0.5. A drag coefficient of 0.5 was chosen as the drone is estimated to be a very non streamlined body. The drone cross-sectional area is calculated as the diameter (55 cm) times the height(10 cm) which is the smallest possible area of the drone. These estimations gives the following:

$$2\ kg * 9.82 m/s^2 = \frac{1}{2} \bullet 1.2 kg/m^3 \bullet velocity^2 \bullet 0.5 \bullet 0.055 m^2$$

Solving this equation gives a terminal velocity of 34.5 m/s. With this velocity the kinetic energy is 1190 J. The drones characteristic dimension is 0.55 m. The drone will operate BVLOS over a sparsely populated area. Based on this and according to table 2 in [11] the risk is **3.** Even though some of the estimate for the kinetic energy might be a little off the drones kinetic energy is well below the the upper limit which is 34 kJ.

4.4 Step #3 – Final GRC determination

As a parachute is installed on the drone that is thoroughly tested, the effect of the ground impact is reduced thoroughly. No response plan is in place and no containment is in place, the results is as follows:

	GRC
Initial	3
An Emergency Response Plan (ERP) is in place, operator validated and effective	+1
Effects of ground impact are reduced (emergency Parachute)	-2
Containment in place and effective	0

[11] https://drive.google.com/file/d/13pWwiSsGTeaT5_b9ezcEUy2MH2K1U8gg/view?usp=sharing

Final	**2**

4.5 Step #4 – Determination of the Initial Air-Risk Class (ARC)

The initial ARC follows directly from the ARC using figure 4 in [12]. Since the airspace at HCA airport is closed/reserved during our test flight, the risk of collision with manned aircraft is low to non-existent. This is interpreted as an a-typical airspace, which gives the determines the Initial Air-Risk Class to be a ARC-a (AEC-12)

4.6 Step #5 – Application of Strategic Mitigations

A full battery will be able to fly the drone approx. 2.000 meters, thus operating at all times within the closed airspace (the closest edge of the airspace is located approx. 5.500 km from the center of the airport, which places the operation distance well below half of this). Based on this, no air collision is considered of any risk and the Initial ARC is the final ARC.

4.7 Step #6 – Adjacent Airspace Considerations

As before, the UAS will always be within the test area 1 (EK R OD1), and will not reach halfway to the edge of this flight zone. If the drone was to operate 2.000 meters and fly back after changing a battery, the drone would, in case of loss of control, not be able to reach the edge in this case either. The UAS will thus always be contained within a closed airspace and the adjacent airspace will be the contained area also - providing a high level of containment robustness.

4.8 Step #7 – Tactical Mitigation Performance Requirement (TMPR) and Robustness Levels

According to table 6 in [13] the robustness level of TMPR is shown the table below

Final ARC	**ARC-a**
Tactical Mitigation Performance Requirements (TMPR)	**N/A**
TMPR Level of Robustness	**N/A**

4.9 Step #8 – SAIL determination

According to the previous steps and table 7 from [14] the final SAIL score is **I**

[12] https://drive.google.com/file/d/13pWwiSsGTeaT5_b9ezcEUy2MH2K1U8gg/view?usp=sharing
[13] https://drive.google.com/file/d/13pWwiSsGTeaT5_b9ezcEUy2MH2K1U8gg/view?usp=sharing
[14] https://drive.google.com/file/d/13pWwiSsGTeaT5_b9ezcEUy2MH2K1U8gg/view?usp=sharing

4.10 Step #9 - Identification of Operational Safety Objectives (OSO)

OSO Number (in line with Annex E)		**SAIL**
		I
	Technical issue with the UAS	
OSO#01	Ensure the operator is competent and/or proven	The operators are students at SDU with a specialization in Drone Technology, and thus is considered to have the right level of competenze.
OSO#02	UAS manufactured by competent and/or proven entity	DJI frame and PX4 stack on a pixhawk 2.1
OSO#03	UAS maintained by competent and/or proven entity	The UAV is maintained by students at SDU with a specialization in Drone Technology. Not only this but before each flight the drone is examined for damages and a general checkup. After each flight the UAV is also examined and repaired.
OSO#04	UAS developed to authority recognized design standards[1]	O
OSO#06	C3 link performance is appropriate for the operation	O
OSO#05	UAS is designed considering system safety and reliability	O
OSO#07	Inspection of the UAS (product inspection) to ensure consistency to the ConOps	All systems described in #STEP1 in the SORA, namely the ConOps, has to be operational during a flight. Connection to all these systems are tested before each flight to ensure no unexpected mishaps happens during a mission.

OSO#08	Operational procedures are defined, validated and adhered to	Pre-flight day list ● Weather ● Battery charge and undamaged ● Ground station charge ● Transmitter charge ● All parts for mission packed ○ Ground equipment ○ UAV ○ Spare parts ● Inspection of UAV ○ Motors ○ Propellers ○ All components securely in place ○ LEDs are working ● Permission to fly ● Enough people available to perform mission Pre-flight list ● Inspection of UAV ○ Motors ○ Propellers ○ All components securely in place ○ LEDs are working ● Both landing station setup ● Signal strength ● GPS connection Post-flight list ● Remove battery ○ Discharge to storage ● Inspection of UAV ○ Motors ○ Propellers ○ All components securely in place ● Download flight log
OSO#09	Remote crew trained and current and able to control the abnormal situation	A parachute module is integrated with the UAV and deploys if a critical state is entered. If this happens, the UAV will fall to the ground at a location close the the last transmitted GPS data and thus should be easily found. A vehicle is available should the landing site be situated far away from the GCS. Otherwise, the mission is contained to within 2 km and should easily be walkable.
OSO#10	Safe recovery from technical issue	Worst case is total failure of system, but here a parachute is available for safe landing. If a single motor fails the UAV should be able to transition into failure mode and proceed to a designated landing location based on GPS coordinates as described in the ConOps A.2.9.

	Deterioration of external systems supporting UAS operation	
OSO#11	Procedures are in-place to handle the deterioration of external systems supporting UAS operation	Pre-flight day list ● Weather ● Ground station charge ● Docking station charge ● All parts for mission packed ○ Ground equipment ○ GCS ○ Spare parts ● Inspection of docking station ○ Camera ○ Sliding system Pre-flight list ● Inspection of docking station ○ Camera ○ Sliding system ● Both landing station setup ● Signal strength ● All ROS nodes running ● Webserver running Post-flight list ● Remove battery ○ Discharge to storage ● Inspection of docking station ○ Camera ○ Sliding system
OSO#12	The UAS is designed to manage the deterioration of external systems supporting UAS operation	The UAV lands without precision if the docking station is not up to code, and releases the parachute if total failure erupts.
OSO#13	External services supporting UAS operations are adequate to the operation	UTM server is available with information of the surrounding airspace. The mission also takes place in an atypical airspace, namely an airspace that can be restricted to outside traffic. Constant communication to the airport traffic control [HCA] is in place in case of an emergency.
	Human Error	

OSO#14	Operational procedures are defined, validated and adhered to	All members of the crew knows the checklist in this OSO and follows them together to ensure no step is skipped, thus creating some redundancy.
OSO#15	Remote crew trained and current and able to control the abnormal situation	As each crew member is a student at SDU with specialization in drone technology, they can handle all abnormal situations.
OSO#16	Multi crew coordination	One person (main coordinator) monitors the GCS for all messages, this person is then the communication link between crewmembers. One person (drone operator) has continuous control of the RC transmitter in case of emergencies. Two persons are in charge of launch and recovery, they are in telecommunication with the main coordinator at all time, it is there responsibility to report if the takeoff/landing is failing.
OSO#17	Remote crew is fit to operate	Followed by OSO#15 they are able.
OSO#18	Automatic protection of the flight envelope from Human Error	O
OSO#19	Safe recovery from Human Error	O
OSO#20	A Human Factors evaluation has been performed and the HMI found appropriate for the mission	O
	Adverse operating conditions	
OSO#21	Operational procedures are defined, validated and adhered to	In case of adverse operating conditions the mission will be cancelled as no procedures are in place to handle these.

OSO#22	The remote crew is trained to identify critical environmental conditions and to avoid them	As each crew member is a student at SDU with specialization in drone technology, they can identify and handle all environmental challenges.
OSO#23	Environmental conditions for safe operations defined, measurable and adhered to	The temperature at mission altitude has to be positive as no icing can occur here. The wind should be below 10 m/s. The humidity should be low. Rain or snow is not expected during operation. The weather in Denmark is described in the ConOps A.2.2.1.2.
OSO#24	UAS designed and qualified for adverse environmental conditions	O

[1] The robustness level does not apply to mitigations for which credit has been taken to derive the risk classes. This is further detailed in para. 3.2.11(a).