University of Southern Denmark

Master project - Autumn 2020 to Spring 2021

Vision based navigation and precision landing of a drone

Kenni Nilsson
kenil16@student.sdu.dk

Supervisor: Henrik Skov Midtiby and Jes Grydholdt Jepsen
Institute name: The Faculty of Engineering (TEK)
Project type: Master Project
Number of ETCS-points: 40
Project period: 2020-09-01 - 2021-06-01

Abstract

This paper proposes methods for navigation of an unmanned aerial vehicle (UAV) utilizing computer vision. This is achieved using ArUco marker boards which are used for pose estimation of the UAV. The goal is to have a UAV to fly autonomously using GPS in outdoor environments from where missions can be executed. When the UAV needs to recharge, a reliable GPS to vision transition is performed where the UAV uses its front camera to detect an ArUco marker board located on a wall to the entrance of a building. When the UAV is of a certain distance in front of this board located on the wall, it will start using its bottom camera for navigation in indoor GPS denied environments. In the indoor environment, a precise vision based landing can then be performed when it needs to recharge, by using its front camera for pose estimation of ArUco marker boards located in front of the landing stations. For vision based navigation, sensor fusion will be used for pose optimizations.

To achieve autonomous flight for offboard control, the robot operating system (ROS) is used along with the PX4 autopilot. Using ROS, a number of nodes for drone control, autonomous flight, ArUco marker detection and sensor fusion can be run concurrently. The ArUco marker detection and pose estimation are accomplished by using OpenCV and an unscented Kalman filter for sensor fusion. The implementation has been tested in simulation using Gazebo and real flight using a Holybro Pixhawk 4 mini QAV250 Quadcopter with a Raspberry Pi 4B which was used as companion computer where an OptiTrack system has been used for ground truth tracking of the UAV.

Results show that the implementation works well, where the UAV was able to make GPS to vision transitions even in strong wind as well as its ability to use a minimum amount of ArUco markers located on the ground for vision based navigation. The latter was accomplished by successfully implementation of sensor fusion where IMU, barometer and vision data were fused together to achieve reliable pose estimates. The mean error of the precision landings were found to be in the order of centimeters which concludes a reliable implementation of the vision based landings.

Reading guide

The reader is highly recommended to have a PDF version of this master thesis available even though if the reader prefers a paper edition for reading the thesis. This is because links to web pages are used in the report. The most important link is to the Github repository where the code and videos of simulations and real flight of the UAV can be found. These links are labeled light blue in the text. Moreover, most of the illustrations, diagrams, tables and images used are implemented with very high resolution. This is done so that the reader can zoom in on important aspects of the used methods and results if wanted. Moreover, labels to all sections, figures, equations and tables will be labeled dark blue for easy access. Lastly, references will be labeled with a green color as [24] where this is a reference to the mentioned GitHub repository.

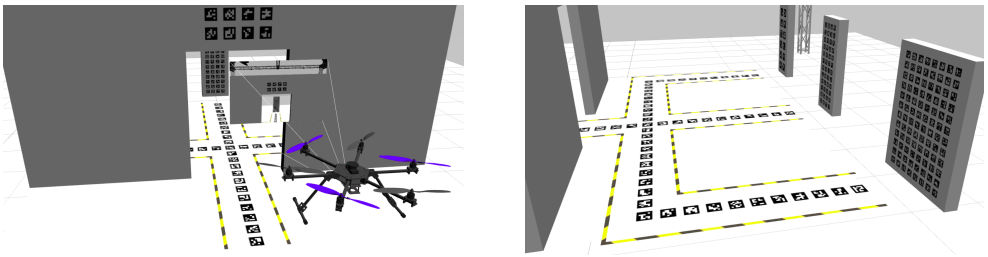
Contents

1 Introduction

This thesis is about vision based navigation of a UAV where it is to perform a precise landing according to ArUco markers. The reason for choosing this area of subject is because of the ongoing HealthDrone project which is being developed in Odense. This project is about transportation of blood samples between Odense University Hospital (OUH) and Svendborg Hospital using drones. This is a three-year innovation project funded by Innovation Fund Denmark and is to be completed in 2021 [14]. Another inspiration has been the Experts in teams (EiT) project 2020 [25] where an autonomous solution for fence inspection were to be made. In this case a drone were used which had to follow the fence and analyze the possibility for breaches using a deep neural network. Here the UAV had to be able to return to an indoor landing station for recharging after the inspection task was completed.

For these projects to be completely autonomous, an efficient and robust solution for indoor navigation and landing must be considered from where battery recharging or replacement is to be performed. This also calls for a robust solution for the transition of flying outdoor to indoor even in windy conditions. Furthermore, because the UAV is to land on a recharging or battery replacement station, the landing must be performed with high accuracy and precision.

Using the Global Positioning System (GPS), the UAV can be set to fly between destinations. However, most of these systems comes with an error in the range of meters. To achieve better, real time kinematics (RTK) can be used, which reduces the GPS error to centimeters. However, RTK is pretty expensive and hence not an optimal solution for low cost applications. Moreover, for indoor navigation, the use of GPS would not be possible because of the reduction in signal strength.



(a) *(b)*

Figure 1: Illustration of the procedures in the GPS to vision based navigation. Initially the UAV uses GPS as navigation to a predefined location. Then it finds and navigates to the ArUco marker board located on the wall as seen in Figure 1a using its front camera. When close enough to the board located on the wall, the UAV will start using the bottom camera for vision based navigation according to the markers located on the ground. Now the UAV can move to the indoor environment for recharging and land in front of one of the three landing ArUco marker boards as seen in Figure 1b where the UAV again will use the front camera for detecting the ArUco marker landing boards

A solution to this problem could be to attach cameras on the UAV and analyze the incoming data using computer vision. By placing markers on the ground, the position of these markers could be found with high accuracy and precision. This would enable autonomous flight tasks where high accuracy and precision of the position is needed.

This project proposes methods for navigation in environments using computer vision. The basic idea of this can be seen in Figure 1. Here the UAV will fly autonomously using GPS coordinates until the UAV has to move inside a vision navigation area for recharging. The UAV will use the ArUco marker board located on the wall for the actual GPS to vision transition and afterwards using the markers located on the ground for the navigation using vision in the indoor environment. Then an acurate and precise landing can be executed using the ArUco marker landing boards from where the recharging is to take place. This is a cheap and effective solution when lack of GPS is present e.g using low cost GPS systems or inside buildings [3].

ROS will be used along the PX4 autopilot in order to achieve autonomous flight of the UAV with Gazebo as the simulation environment so that the UAV can be sufficiently tested before flying.

1.1 Related work

Vision based navigation is a vast research area for utilizing computer vision for navigation and precision landings. According to [28], landing platforms can be categorized into the three groups for different landing scenarios. Category one defines platforms which are fixed, category two for moving platforms of two degrees and category three which considers landings on ships where the platform of the ship deck can be modeled as a rigid body with six additional degrees of freedom based on the psychical movements of the waves.

Fixed based platforms using computer vision have been used for some time to enable landings for better accuracy compared to that of using GPS. In [22], an implementation of using ArUco markers located on a landing platform is proposed. Here the UAV is able to land and recognize the ArUco marker located on the ground from an altitude of twenty meters using Pixhawk 4 as flight controller with a Raspberry Pi 3 for onboard computing.

In [8] [16] autonomous landings are performed where the UAV is to land on a wireless charging station. However, though these systems perform quite well, both use a single ArUco marker for pose estimation with a cost in reduced accuracy which may affect the overall performance of the landing system. The advantage of using more markers for pose estimation was investigated by Tiago Gomes Carreira [6]. He performed an evaluation in using different sets of ArUco markers from where the error in the estimation is based on one, four and eight ArUco markers. Results show performance improvements when more markers are present in the image.

Shifeng Zhang et al. [21] proposes a different method where more than one ArUco marker is used in the vision based landing. In this setup a landing platform with ArUco markers with different sizes is used. This was found to make the system more reliable when the pose estimation of the markers is made from different distances.

Instead of only using a camera attached to the UAV which points downwards, Ghader Karimian et al. [18] presented a method for using both a bottom-facing and front-facing camera to utilize automatic navigation and landing of a UAV in indoor environments. This solution uses a single ArUco marker and IMU measurements to be fused in a Kalman filter. In this setup, the UAV will use the front camera to navigate towards a marker located on the ground. When the UAV loses sight of the marker using the front camera, it will continue in the same direction using IMU data until the marker is visible from the perspective of the bottom camera where it is to land.

For vision based navigation in indoor inviroments Chouaib Harik et al. [26] proposed a solution by having an Unmanned Ground Vehicle (UGV) to be used as platform for the UAV. Here the platform contains a marker from which the UAV can takeoff and land. This UGV can navigate among rows of racks where the UAV takes off and scans the inventories where it keeps its pose using sensor fusion of IMU and marker data. However, this solution requires an UGV equipped with lidars for navigation which is expensive.

Another method was proposed by Michael Maurer et al. [4]. This method uses model-based visual localization to that of a precomputed map of the environment which is used to estimate the drift of a odometry sensor to achieve centimeter precision of the pose of the UAV. Both these methods yields solutions to vision based navigation for indoor environments, but does not take into account ways of making transitions between using GPS and vision as navigation.

The way this thesis differs from previous work is the implementation of a GPS to vision based transition using the front-facing camera of the UAV to navigate to an ArUco marker board located on the wall. Moreover, this implementation will be build on a completely autonomous system, where the UAV can handle the transition of using GPS, then vision, land and then move out of the vision navigation area for then again to use GPS for mission executions.

1.2 Problem Statement

The UAV must be able to make a reliable GPS to vision transition. This leads to the challenge of finding a robust solution for the transition of using GPS coordinates to indoor vision based navigation even in windy conditions. This transition must be incorporated using markers located on the wall to the entrance of the vision navigation area. The UAV then needs to be capable of navigating in indoor environments using markers located on the ground.

A solution for onboard computing for pose estimation of the markers as well as autonomous flight must be found. This says that the computer must be able to perform real-time calculations of the pose of the markers to ensure a reliable system.

Furthermore, the precise landing should be implemented in such a way that the time, from which the UAV is hovering above the marker to the landing is performed, is minimized. The same goes for the error associated with the precision landing.

This leads to the following problems:

P.1 How can computer vision be used to detect markers?

P.2 How can a smooth transition between using GPS coordinates to vision based navigation be found?

P.3 How can navigation between markers be performed?

P.4 How can sensor fusion be implemented for pose optimization?

P.5 How can a precise landing be executed?

P.6 How can the landing time be reduced without causing instabilities to the UAV?

1.3 Specification of requirements

From the outline of the project as well as the problem statement, the following requirements for the project have been formulated:

R.1 The implemented system must be able to function completely autonomous. This is wanted because an autonomous solution would yield a reduction in operational cost for the target user e.g HealthDrone project.

R.2 The mean error of the landing must not exceed 10 cm. This is based on results from related work in Section 1.1, where this implementation as a minimum must yield similar landing precisions

R.3 The pilot must be able to send missions on a wireless connection to the UAV for execution. This is critical in order to control and test the system when each element of the implementation is executed

R.4 The UAV should be able to make the transition of using GPS coordinates to indoor vision based navigation even in windy conditions i.e up to 8 m/s. This is based on the average wind speed in Denmark from three coastal stations (DMI) which is found to be between 7-8 $\frac{m}{s}$ as a sort of worst case scenario for normal operation of the UAV

R.5 The landing must be performed within 5 seconds from where the UAV is hovering 1.5 meters above the landing sight to a landing is performed. From the perspective of a vertical velocity of 0.3 $\frac{m}{s}$ of the UAV for this demand to be met, this is set as an upper limit for the landing time which is found appropriate.

1.4 Tentative time schedule

At the start of the project a tentative time schedule was formulated. This was done to keep an overview of the most essential elements which had to be implemented. Time was put aside to complete each block of work so that things could be done in an order which was found appropriate. This time schedule can be seen in Figure 2 and 3. The main focus has been to get things to work properly in simulations before conducting any tests on the UAV to reduce the risks of failures.

In the initial phase of the project CAD models were created for buildings and markers. Then the ROS implementation along the PX4 software was installed which would be used in the simulation. Hence, the main focus of the project for 2020 was the computer vision part where marker detection, pose estimation, navigation between markers and landing was in focus which was programmed in python with OpenCV as the computer vision software. These used methods will be discussed in Section 3.

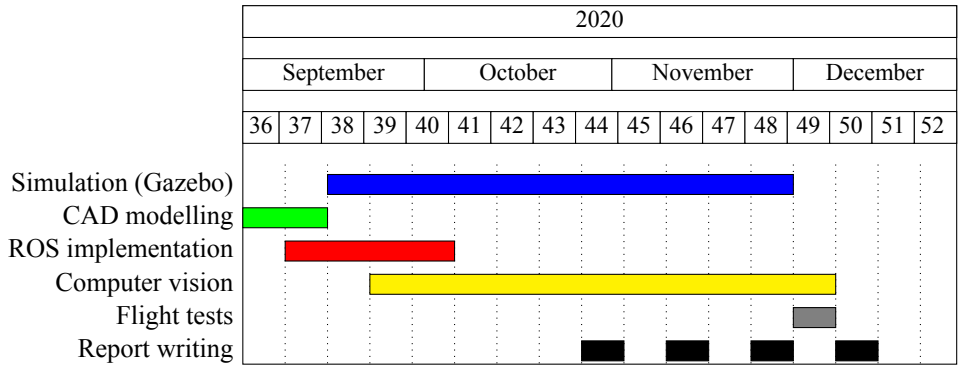


Figure 2: Tentative time schedule for 2020

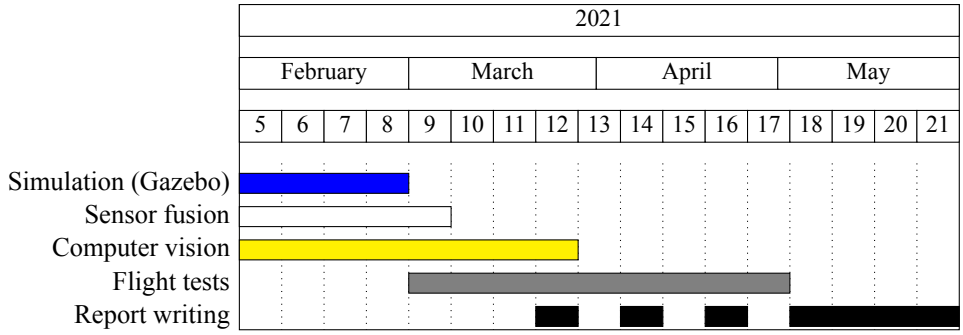


Figure 3: Tentative time schedule for 2021

In the second part of the project, the initial idea was to test the implementation on the real hardware. This was scheduled to begin in the start of March. However, due to the current situation with Covid-19, this was not possible because all access to Hans Christian Andersen airport in Odense was rejected. Due to these limitations, more work has been put into the simulations. As seen in Figure 3, approximately two months were put aside for flight tests. Because of the limitations just discussed, this period of time had now been used for simulations and report writing instead. The result of the implementation of the autonomous system for offboard control was completed at the beginning of April where videos of all the tests in simulation were recorded for easy visualization of the procedures for testing the system.

The possibility of presence at the airport was made possible at the end of April. Hence, two weeks of work at the airport was put aside in May for execution of some of the planned flight tests before all time was put aside for report writing at the end of May. Report writing has been done in small steps throughout the period of the project, but the main contribution to this part was done in May where the complete implementation of the system could be documented.

2 Materials

This section deals with the materials used to build the UAV, for communication and onboard computing. A description of the different products and a wiring diagram for the UAV platform will be analyzed in Section 2.1. In Section 2.2, the bill of materials is discussed with the overall cost of the implementation in regard to hardware.

2.1 Hardware

The UAV used is a Holybro Pixhawk 4 mini QAV250 Quadcopter. The reason for choosing this kit is because of the Pixhawk 4 mini that is part of the product which is used as autopilot for the UAV. The software part of PX4 will be discussed in Section 3.2.3. The UAV can be seen in Figure 4a.

For communication between the pilot and UAV, a FlySky FS i6 transmitter is used. Though this is a basic transmitter it offers great features together with a receiver for a low price. The transmitter and receiver can be seen in Figure 4b.

(a) *(b)*

Figure 4: Illustration of the Holybro Pixhawk 4 Mini QAV250 UAV in Figure 4a and flysky transmitter in Figure 4b

Because the UAV and onboard computer has to be powered separately, two gens ace LiPo batteries with 11.1V and 2200 mAh have been considered. These batteries matches the XT60 connection which is used by the power board of the kit. The reason for choosing these as 3-cell batteries compared to 4-cell, is to reduce the height of center of mass of the UAV which could make the system unstable because the batteries are placed above the propellers as seen in Figure 4a. The UAV and Raspberry Pi has to be powered separately because the Raspberry Pi must be supplied with constant 5 volts and 3 amps for best performance. If the Raspberry Pi gets either under- or overvoltage, the Pi could potentially shut down which could be fatal if the UAV were relying on pose estimates from the onboard computer. The used LiPo battery can be seen in Figure 5a.

(a) *(b)* *(c)* *(d)*

Figure 5: The used materials for battery and power configurations. The used Gens ace LiPo battery in Figure 5a, Ubec 3A power converter in Figure 5b, SKYRC balance charger in Figure 5c and LiPo safety-bag in Figure 5d

To convert the input voltage from the LiPo battery to the Raspberry Pi, a KINGKONG 5V 3A Switching Power UBEC converter for switching between 7.4-22.2V (2-6S LiPoly) to constant 5V/3A is used. The setup of this requires a little soldering where the ends of the wires have to be soldered to a XT60 connector to be used to connect to the LiPo battery, where the output of this is connected to the 5V and ground pins of the Raspberry Pi. This switch can be seen in Figure 5b.

To charge the LiPo batteries, a SKYRC balance charger is used. This unit can charge 2-4 cell batteries with constant 1-3 amps of current which fits the 2200 mAh batteries. Moreover, it includes a XT60 charging cable which matches the already mentioned connectors. This charger can be seen in Figure 5c.

Because LiPo batteries can be dangerous and may cause fire if damaged or overcharged [20], a LiPo safety bag is used as seen in Figure 5d. This bag includes fireballs which encapsulates the possible fire of the batteries to be only in the safety bag.

In order to achieve serial communication between the PX4 and the Raspberry Pi, the universal asynchronous receiver-transmitter (UART) port of the PX4 is used via a 6 pin connection cable seen in Figure 6b. Either two ways could be used to achieve serial communication from the point of view of the Raspberry Pi. One method is to use the general purpose input output (GPIO) pins of the Pi which is Tx (transmit) and Rx (receive). If choosing this setup, the data will be transmitted through the */dev/ttyS0* port of the Raspberry Pi where you got to have read/write privileges or being part of the group *tty* to do so. These settings can be changed accordingly from the terminal in Ubuntu as seen in Listing 1.

```
$ cd chmod a+rw /dev/ttyS0
$ usermod -a -G tty ubuntu
```

Listing 1: How to change read/write permissions along with adding a user (ubuntu) to the tty group

The other method is to connect the 6 pin connection cable to a TTL 232R cable seen in Figure 6a to the universal serial bus (USB) port of the Raspberry Pi. This can easily be done by solder the Tx, Rx and ground parts of the two cables together. Some benefits comes by using the later solution e.g no problems with u-boot of the Raspberry Pi when the PX4 transmit data on the GPIO pins during boot up of the system [**uBoot**]. Therefore, the latter option is used.

(a) *(b)*

Figure 6: Cables for serial communication between the PX4 flight controller and Raspberry Pi. To enable the use of serial communication from the USB of the Raspberry Pi, a USB to TTL-RS232 cable is used which can be seen in Figure 6a and a six pin serial connection cable in Figure 6b

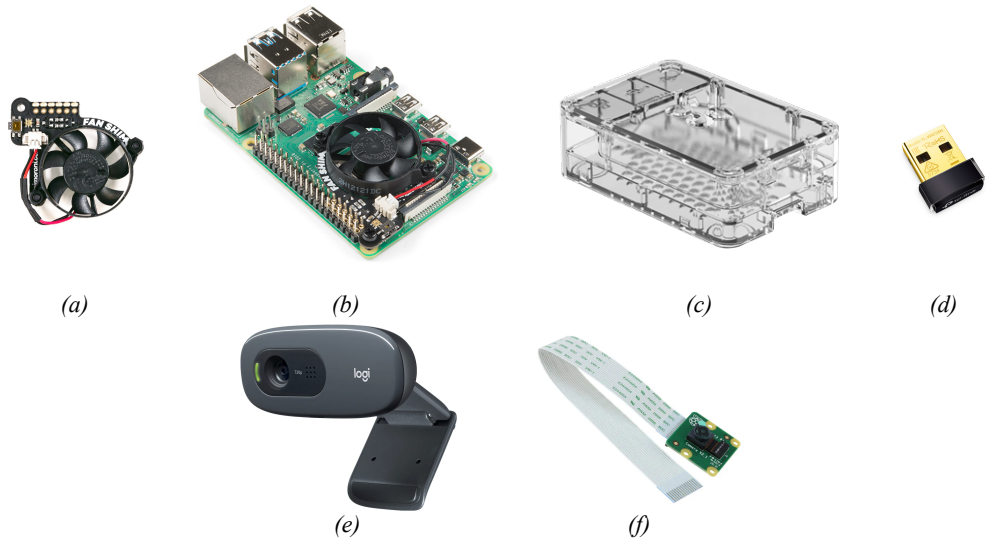
In regard to the onboard computer, a Raspberry Pi 4B with 8 GB of random access memory (RAM) as seen in Figure 7b will be used. Here the operating system Ubuntu 18.04.5 LTS (Bionic Beaver) 64-bit server edition will be implemented. This is further discussed in Section 3.5 and why this was chosen compared to other companion computers.

Because the central processing unit (CPU) of the Raspberry Pi can overheat during constant computational load, a fan is used to cool down the Raspberry Pi during operation. The Shim fan along with the fan attached to the Raspberry Pi can be seen in Figures 7a and 7b respectively. Moreover, by using active cooling instead of passive cooling e.g an aluminum heatsink case, the weight of the Pi is reduced. This also enables the use of overclocking of the Raspberry Pi for better performance which is discussed in Section 3.5.

To safely attache the Raspberry Pi on to the UAV, a transparent case is used as seen in Figure 7c. This case is appropriate because it can store both the Shim fan and Raspberry Pi camera without any chance of damage to the hardware on the board. In order to control the UAV using WiFi, a WiFi adapter seen in Figure 7d is used in order to get ground truth data

from the Optitrack system. This is discussed in Section 3.5 and 3.6.

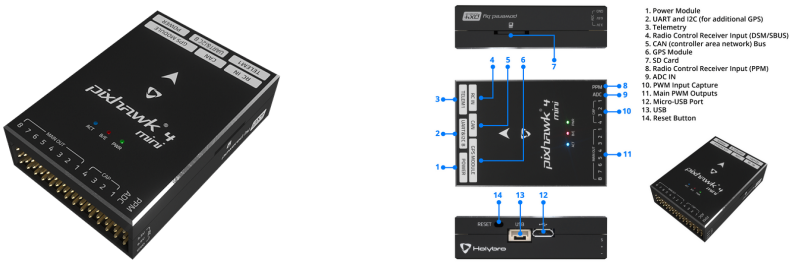
A Logitech C270 HD camera is used as the front camera and a Raspberry Pi V2 as bottom camera as seen in Figures 7e and 7f respectively. The Logitech camera will be attached to the USB port of the Raspberry Pi while the Raspberry Pi camera is connected through the camera serial interface (CSI).



(a) *(b)* *(c)* *(d)*

(e) *(f)*

Figure 7: The Raspberry Pi 4 can be seen in Figure 7b with the SHIM fan connected. The case for the Raspberry Pi can be seen in Figure 7c. The used USB WiFi adapter from TP-Link can be seen in Figure 7d. Lastly, the Logitech C270 HD and Raspberry Pi V2 cameras can be seen in Figures 7e and 7f respectively



(a) *(b)*

Figure 8: Illustration of the Pixhawk 4 mini in Figure 8a and interface connections in Figure 8b

The Pixhawk 4 mini can be seen in Figure 8a with all interfaces in Figure 8b. The PX4 features accelerometers, gyros, a magnetometer and barometer along with a 32-bit arm cortex-m7 processer for computing. For communication, a number of serials ports (UART) are included in the PX4. For more information, a link to a detailed description of all features of the board can be seen in Figure 8.

All connections between the user, PX4 and Raspberry Pi can be seen in Figure 9. The green boxes indicate hardware which the user can interact with the UAV. The ground control station (GCS) is software installed on the computer from where communication between the GCS and PX4 can be performed. The commands can be passed directly to the PX4 system from the GCS e.g takeoff, landing, UAV whereabouts etc. This is done from a set of transmit/receiver pairs attached to the computer by a USB connection and the other to the PX4 using the Telem interface (UART). In order to take command of the UAV from manual flying, the pilot can do so by using the transmitter which is connected to the PX4 using a radio receiver module connected through RCIN. The telemetry links along with the GPS can be seen in the yellow boxes, where

the GPS is attached to the GPS input of the PX4.

To communicate directly with the Raspberry Pi, an access point can be set up from a hot-spot or wireless connection to a WiFi network. The configuration of this will be explained in Section 3.5. This enables commands directly passed to the Raspberry Pi for controlling the UAV or video streaming of the cameras to the laptop.

The shim fan is connected to the Raspberry Pi from the GPIOs. This is because the fan can be set with a certain speed of rotation to take into account the temperature of the CPU to act accordingly. All of the hardware directly associated with the Raspberry Pi are labeled red.

Through the output pins of the PX4, power is regulated using electronic speed controllers (ESC) before past to the four motors connected to the propellers of the UAV. As already mentioned, the batteries are connected separately to the PX4 and Raspberry Pi through power converters from where constant and stable voltage and current are supplied to the two units.

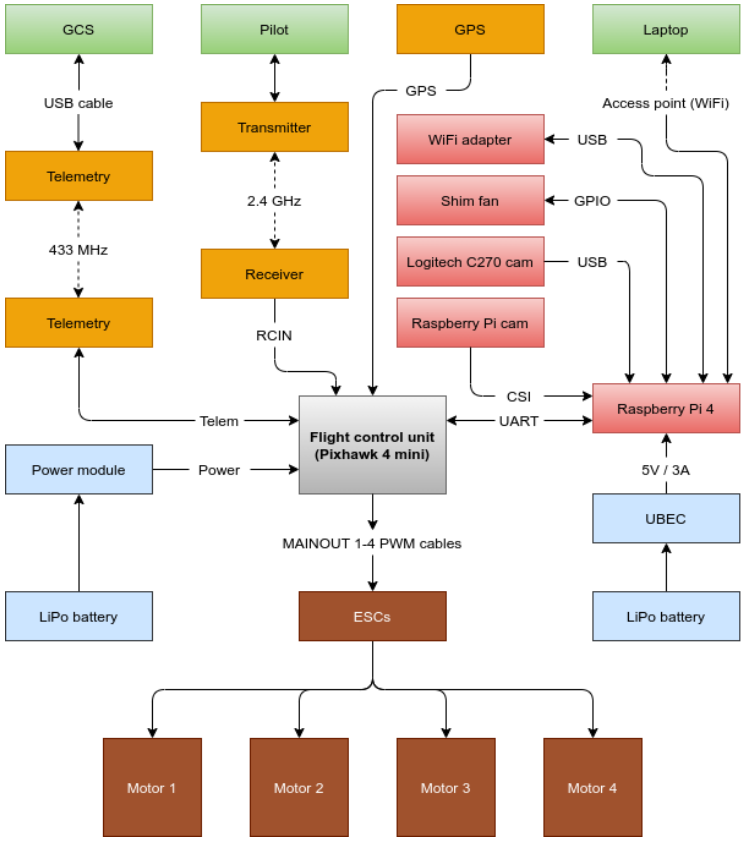
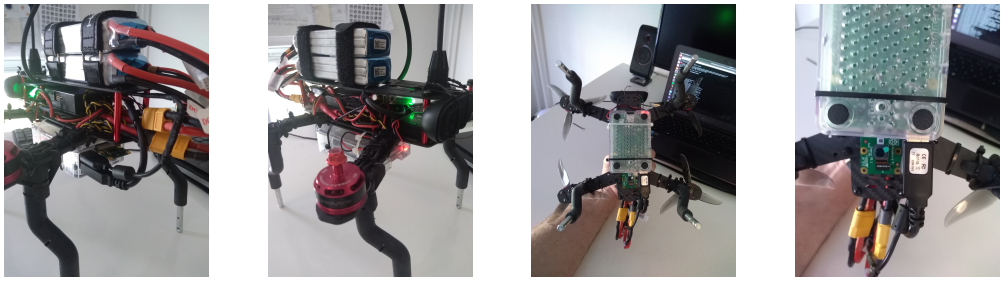


Figure 9: The wiring diagram which illustrates the setup of all connections between hardware. Green boxes defines user interactions with the PX4 and Raspberry Pi, yellow boxes communication modules and GPS, blue boxes the batteries and converters, brown boxes motors and regulators, red boxes Raspberry Pi and associated hardware and lastly the PX4 flight controller labeled gray. Solid and dotted lines defines wire and wireless connection respectively

The final configuration of the QAV250 Quadcopter can be seen in Figure 10 and 11. Here a landing gear is attached to the UAV to make room for the Raspberry Pi, which is attached at the bottom of the UAV. The GPS module has been placed at a higher position compared to the original design in order for better data transmission. Because the propellers of the UAV are near the flight controller, cables have been set close to the center of the UAV in order to avoid collisions. The web camera case of the Logitech C270 HD camera has been removed to fit the UAV which is placed at the front of the UAV which can be seen in Figure 10a. Moreover, the USB cable of the Logitech C270 HD has been shortened and re-soldered to fit the configuration of the UAV which may be noticed by the tight ordering of the cables in Figure 11a and 11b.

(a) (b)

Figure 10: Illustration of the final updated design of the QAV250 Quadcopter. As it can be seen in Figure 10a, a landing gear has been attached to the UAV to make room for the Raspberry Pi



(a) (b) (c) (d)

Figure 11: Illustration of the final updated design of the QAV250 Quadcopter. In Figure 11d, the Raspberry Pi camera is seen to pointing straight down which is accomplished by using the Raspberry Pi case

2.2 Bill of materials

The materials used in this configuration can be seen in Table 1 where products, price and supplier is mentioned. It may be noticed that the overall price of the kit is approximately 4.502,14 DKK which is mostly due to expensive items like the drone kit and Raspberry Pi 4.

Table 1: List of products with price for the materials used

Products	Supplier	Amount	Price
Drone (*Drone and communication*)			
HolyBro QAV250 + Pixhawk4-Mini	banggood.com	1	2.037,73 DKK
FlySky-FS-i6-2.4G-6CH	banggood.com	1	352,76 DKK
Batteries (*Batteries and related materials*)			
Gens ace LiPo 11.1 V 2200 mAh Cell: 3 45 C XT60	conradelektronik.dk	2	249,00 DKK
KINGKONG 5V 3A Switching Power UBEC	hobbyking.com	1	23,65 DKK
SKYRC e430 charger 3 A LiPo, LiFePO	banggood.com	1	209,00 DKK
EXTRON Modellbau LiPo safety-bag 1 Set	cdon.dk	1	259,00 DKK
Raspberry pi (*Raspberry Pi and related materials*)			
Raspberry Pi 4 model B - 8GB	raspberrypi.dk	1	669,00 DKK
Fan SHIM for Raspberry Pi	raspberrypi.dk	1	99,00 DKK
Okdo Raspberry Pi 4 Standard Case Clear	dustinhome.dk	1	55,00 DKK
Logitech C270 HD camera	elgiganten.dk	1	249,00 DKK
Raspberry Pi Camera V2	proshop.dk	1	299,00 DKK
USB WiFi adapter	avxperten.dk	1	149,00 DKK
Overall product price			**4.651,14 DKK**

3 Methods

This section is about the methods used to utilize offboard autonomous flight of the UAV with the aim of making a reliable GPS to vision transition for vision based navigation and autonomous landing using ArUco marker boards.

In Section 3.1 the used 3D models will be discussed and why they were considered to be important in the simulations along the simulation environment Gazebo. Section 3.2 deals with the implementation of autonomous flight of the UAV and offboard control. This implementation will be used in simulation and in the real UAV hardware using a Raspberry Pi. Pose estimation using ArUco marker boards is explained in Section 3.3. Here methods for detecting the ArUco marker boards along with pose estimation will be considered. Moreover, camera calibration is explained, which will be used in the actual hardware on the UAV. For pose optimization, sensor fusion is used which will be discussed in Section 3.4 along with the implementation. In Section 3.5 the onboard computer is explained along with the configuration of the system for the operating system and network configuration. The OptiTrack system will be explained in Section 3.6.

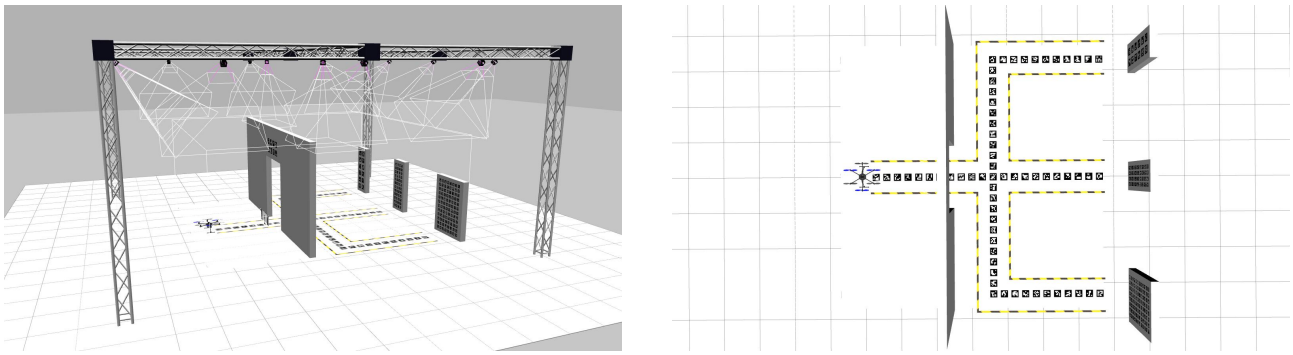
3.1 3D modeling for Gazebo simulations

This section will go through the 3D-models created to be used in Gazebo and why they were considered to be important for properly testing the system before real life implementations on the UAV. The created 3D-models will be discussed in Section 3.1.1 with a short introduction to Gazebo in Section 3.1.2.

3.1.1 3D modeling

The models have been created using the 3D modeling software Blender. This is an open source software used on Ubuntu which is the operating system used. In the modeling software, the ArUco marker boards can be imported as plane images and the final model saved as a *.dae* file which is easily imported in a simulator software like Gazebo.

A number of different 3D models have been created to be used in the simulation environment where two of these can be seen in Figure 12 and 13. Since the actual evaluation of the performance of the UAV is to be done in the OptiTrack system in the airport in Odense, a model of this system has been used with the actual dimensions of the real system. The is done so the pose estimation of the ArUco marker boards can be compared to that of the OptiTrack system which got a millimeter precision in regard to tracking the UAV. Hence, the model has been created with these considerations in mind which had put a threshold to the size of the model which has to fit inside the OptiTrack system as seen in Figure 12a. The OptiTrack system will be discussed in Section 3.6.



(a) *(b)*

Figure 12: Visualization of the OptiTrack model in Figure 12a and top view of the one pattern ArUco marker board in Figure 12b

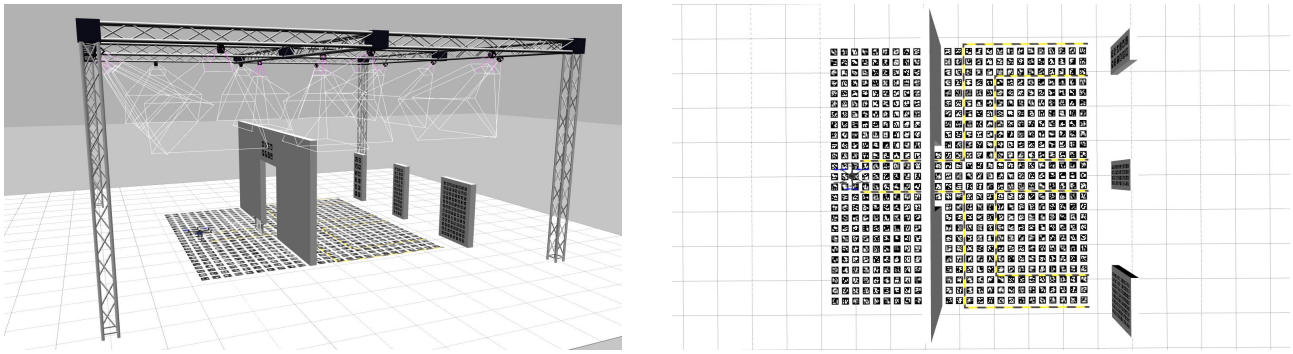
The general idea behind this design is that the UAV can fly towards the wall and locate the ArUco marker board placed at the top of the wall. This board can be seen in Figure 14a which is called GPS2Vision board because the UAV will use this board when switching from using GPS to vision coordinates. This will enable GPS to vision based navigation where the

UAV shifts between using the GPS as the global coordinate system to that of the ArUco marker board. When the UAV has positioned itself with a certain distance to this marker board, it will switch to using the bottom camera where the ArUco board located on the ground will function as a world coordinate system. In this scenario, the UAV is to land on one of three different locations in front of the ArUco board located at the end of each track. Here transformations of the pose of these three ArUco boards have been performed to align these with that of the ArUco board located on the ground to avoid switching between different coordinate systems. This will be explained further in Section 3.3.

The ArUco markers used in these models will be (5×5) bit markers. The GPS to vision marker board located on the top of the wall will be (2×4), the ground marker (25×25) and the landing marker board one (6×2). All these will have a marker length of 0.2 m and marker separation of 0.1 m. The landing board two and three will be (12) and (12×8) respectively both with marker length of 0.1 m and marker separation of 0.05 m. These sizes have been chosen to be appropriate taking into account the size of the OptiTrack system where a flying height of approximately 1.5 meters will be used.

If this setup were to be implemented in real life, an implementation with fewer ArUco markers would be beneficial if this would not decrease the precision of the pose estimation too much. In this small (25×25) ArUco marker board scenario it would not be a general problem, but if this has to be scalable e.g. big industrial companies, great performance using a small amount of ArUco markers would be preferred.

Because the precision of the pose estimation from ArUco marker boards will be based on the number of ArUco markers located in the image [11], the estimation of the pose will be based on a full (25×25) ArUco marker board and a one pattern setup as seen in Figure 13b and 12b respectively. The performance of these different setups will be evaluated in Section 4.1.5.



(a) *(b)*

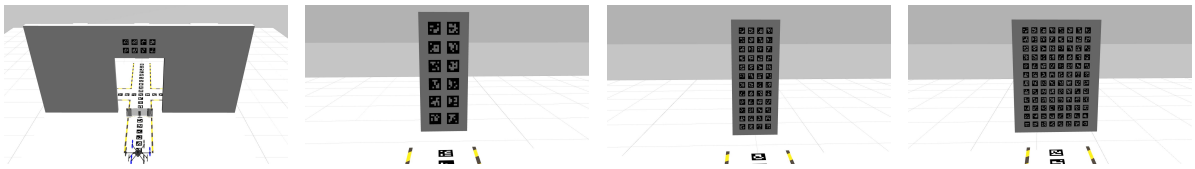
Figure 13: Visualisation of the OptiTrack model in Figure 13a and top view of the full pattern ArUco marker board in Figure 13b

Because relying completely on vision based navigation will put the system in a vulnerable position e.g. no markers are found in the image, optimizations have to be performed. This will be utilized through sensor fusion where the pose estimation from the ArUco marker board and sensor data from the inertial measurement unit (IMU) and barometer will be fused together to give a more reliable estimation of the current position of the UAV. The implementation of this will be discussed in Section 3.4.

To evaluate the performance of sensor fusion, two models with missing ArUco markers have been created as seen in Figure 15c and 15d. The goal of this is to have the UAV to fly a couple of meters without any vision position update for the pose estimation and still keep its track until the pose will be updated when markers again are visible for the UAV.

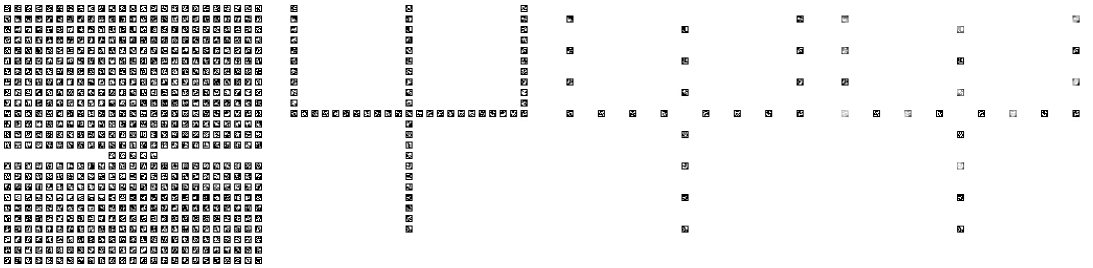
For the vision based landing, three different ArUco marker boards have been created. These can be seen in Figure 14b, 14c and 14d which are called landing board one, landing board two and landing board three respectively. Because these boards have been created is to analyze the performance of pose estimation when a different number of ArUco markers are visible in the image. More ArUco markers in the image should give better pose estimates because more points from each

marker are fused together to give a better pose estimation. Hence, these three landing boards will be used to analyze if a very high number of ArUco markers in the image are required for a precise landing. Results of this can be seen in Section 4.1.3 and 4.1.6.



(a) *(b)* *(c)* *(d)*

Figure 14: Illustration of the created ArUco marker boards used in simulation. The board in 14a is used for the GPS to vision transition for navigation and the boards in Figure 14b, 14c and 14d are used for vision based landings



(a) *(b)* *(c)* *(d)*

Figure 15: Illustration of the ArUco marker boards used which are located on the ground. Four .world files will be created where each include one of the following boards namely the full board in Figure 15a, the one pattern board in Figure 15b and the one pattern board with missing markers and custom made wear in Figure 15c and 15d respectively

3.1.2 Gazebo simulator

Gazebo is an open source 3D robotic simulator which integrates the open dynamics engine (ODE) physics engine, openGL rendering and supports code for both actuator control and sensor simulation. Using this software, the UAV can be tested and simulated in environments and conditions close to real life. Hence, all tests will be performed in simulation before further action will be executed with the UAV. Furthermore, wind can be applied to the simulation which will also be a critical aspect to consider before deployment of the UAV.

To use the newly created models, a world file must be defined. The world file includes all elements which are to be included in a given simulation such as robots, lights, sensors, objects etc. The file must end with a *.world* extension to be recognized as a world file. The GZServer will read this file which builds the world as a virtual environment for which the robot can operate. This means that the operation can be visualized using GZClient or be run in the background only using GZServer to save systems resources. The latter corresponds to running Gazebo in headless mode, which is a great feature to use if a number of simulations have to be performed in an automatic fashion.

The world file uses the SDF format which contains things like *worlds*, *models* etc. Joints can be set to be either revolute or prismatic according to the wanted configuration. To each joint a link can be attached. The collision, visualization and inertia tags specifies the visualization for the model, collision detection and physics in Gazebo respectively. The static tags are used for models which have to be static throughout a simulation like ground, tress etc. The same SDF format is used when creating a model, where the *.dae* file is included to use the 3D models created in Blender.

The plugins in gazebo are code which is compiled as a shared library and inserted into the simulation. These plugins includes world, model, sensors, system, visual and GUI. For instance, if one wants to include a feature to an existing model, like a wind plugin, these can be included into the world file to be used in a simulation. This wind plugin will be used in some of the simulations for stress testing of the ArUco pose estimation and the flight control in general.

3.2 Autonomous flight

This section deals with the implementation of autonomous flight of the UAV. The PX4 autopilot will be used as flight controller which offers many features to enable autonomous flight. This is discussed in Section 3.2.3. Along the PX4 software, ROS will be used which enables onboard computing on the Raspberry Pi and transmission of messages to the PX4 using the MAVROS package. ROS is discussed in Section 3.2.2. For easy setup and access to the UAV QGroundControl is used which is discussed in Section 3.2.1.

The main reason for chosen PX4, ROS and QGroundControl is because the software have been widely used in MSc in Engineering in robot systems with specialization in unmanned aerial systems technology at the University of Southern Denmark. A lot of experience have been gained which will be utilized in the implementation of autonomous flight.

3.2.1 QGroundControl

QGroundControl offers full flight control and setup of the vehicles for a number of flight controllers including PX4. The advantage of using this as the GCS is that one gets immediate updates of the state of the UAV e.g pose, power of battery, errors etc. to a laptop using a wireless connection. Moreover, the latest stable PX4 Firmware can be fetched and flashed directly to the SD card of the PX4. A visualization of the QGroundcontrol GUI can be seen in Figure 16.

The primary usage of this software will be to change parameters on the UAV in flight e.g vertical and horizontal velocities, changes of parameters of PIDs for pose control etc. Furthermore, sensor calibration of the compass, gyroscope and accelerometer can easily be done before every flight to achieve the best performance of the UAV. Besides of this, transmitter and telemetry setup can be performed where different flight modes can be encoded to belong to a channel on the transmitter for manual control of the UAV. Especially two flight modes will be encoded in the system to be triggered when the transmitter is used namely altitude and manual. The altitude mode offers altitude control where the altitude is kept constant using the barometer in the PX4 sensors. In manual mode, the pilot has full control of the UAV where he has to control both altitude and position of the UAV. These modes can then be used to take manual control of the UAV if something unexpected should happen when the UAV is in autonomous offboard mode. A more thorough description of the different flight modes offered in PX4 can be seen in flight modes.

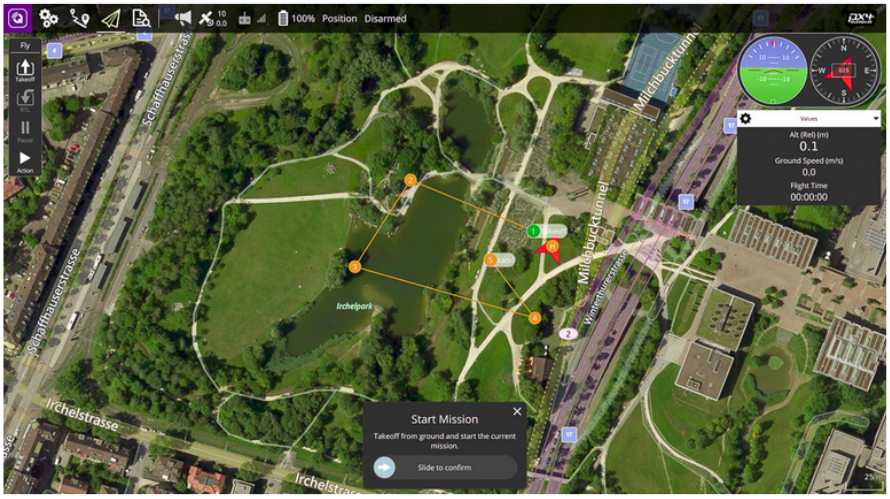
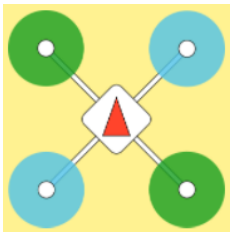
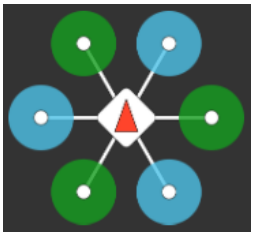


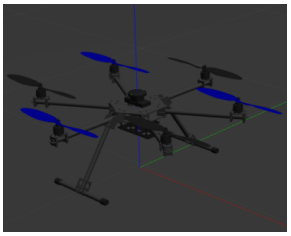
Figure 16: Illustrations of the GUI in QGroundControl. Waypoints can be placed that the UAV can be set to follow from the yellow circles. Here the orientation of the UAV can be seen illustrated by the red arrow. The UAV may also be controlled from commands e.g takeoff



(a) Quadrotor airframe



(b) Hexarotor airframe



(c) SDU drone model

Figure 17: Illustration of different air frame configurations which can be set in QGroundControl. The Holybro QAV250 Quadcopter uses the configuration in Figure 17a and the SDU drone model seen in Figure 17c created by Jes Grydholdt Jepsen uses the configuration seen in Figure 17b

Two different air frames were considered to be used. These can be seen in Figure 17a and 17b. The reason for having two different setups is because of the current Covid-19 situation where access to Hans Christian Andersen airport in Odense is limited. If possible, the SDU drone model seen in Figure 17c, will be used in real life for testing at the airport. Otherwise, the Holybro QAV250 Quadcopter will be used. The reason for wanting to fly with the SDU drone is because of the gained experience in the EiT 2020 at SDU, where this drone was used throughout the project. That is also why the SDU drone model will be used in the simulations.

When configuring air frames in QGroundControl, a number of predefined setups can be used. This goes for the Holybro QAV250 Quadcopter and SDU drone where all parameters are predefined like PID parameters for pose and attitude control.

As it may be noticed in Figure 17a and 17b, the green and blue colors illustrate the spin direction of the propellers. Blue is counterclockwise and green clockwise. The use of QGroundControl offers the possibility to check the spin direction of each propeller separately to insure a proper setup. This feature will also be used in this case for safety reasons.

3.2.2 Offboard control

For the implementation of offboard control as well as communication between units, ROS will be used. ROS is an open source software framework to be used in robot applications. This includes a number of libraries and tools with the aim of simplifying the creation of new robotic solutions. Another important feature is the message handling between nodes in the ROS architecture which is visualized in Figure 18.

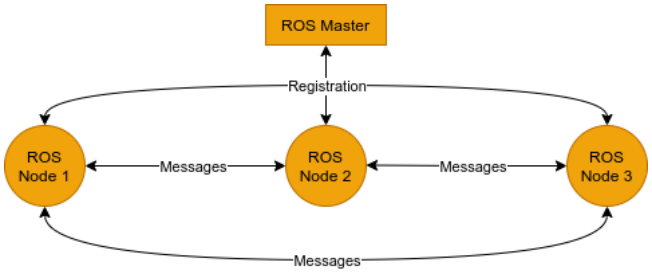


Figure 18: Illustration of a high level view of the architecture in the communication between nodes in ROS. Here three ROS nodes have been created, but the number of nodes can be much bigger according to the specific application

Here the ROS master keeps track on all the nodes in the system where each node goes through registration via the master. Then each node can publish or subscribe to messages which is initiated through the master. This is called topics under the ROS framework and will be used in offboard control where a number of nodes have to communicate to each other.

A number of ROS nodes have been created in the implementation of autonomous flight, which will be discussed in Section 3.2.2.2.

In order to achieve communication between the Raspberry Pi and PX4, MAVROS is used. MAVROS is a packages that provides communication drivers for various autopilots with the MAVLink communication protocol. MAVLink is categorized as a light weight messaging protocol for communication with drones, other unmanned vehicles and there onboard components. This follows a hybrid publish-subscribe and point-to-point design pattern. A visualization of how this works can be seen in Figure 22.

3.2.2.1 ROS Packages

To make the code accessible from within the ROS workspace, a number of ROS packages have been created which will be used in the ROS nodes discussed in Section 3.2.2.2. In order to publish and hence subscribe to barometer and IMU data, two additional plugins are needed which will be used in sensor fusion in Gazebo simulations. These plugins have been downloaded and used as packages in the workspace. These can be seen in Table 2.

Table 2: Plugins downloaded to utilize the IMU and barometer data within the simulation environment Gazebo

ROS package: Hector localization-catkin	
ROS package: Hector quadrotor-kinetic-devel	
Downloaded software [9] [10]	Plugins for publishing IMU and barometer data in Gazebo

To give an overview of the created ROS packages and the used python scripts within, three tables have been formed with a small description of the purpose with each python script. This can be seen in Table 3, 4 and 5. A further discussion of the theory involved in marker detection and sensor fusion will be discussed in Section 3.3 and 3.4 respectively.

Table 3: List of python scripts used in the ROS catkin package marker detection from the created ROS workspace

ROS package: Marker detection	
marker_detection.py	Responsible for detection and pose estimation of ArUco marker boards
marker_detection_ros_interface.py	Interface between the marker detection script and ROS
log_data.py	Responsible for logging of estimated ArUco marker board pose data

Table 4: List of python scripts used in the ROS catkin package sensor fusion created in the ROS workspace

ROS package: Sensor fusion	
sensor_fusion.py	Responsible for sensor fusion of ArUco pose, IMU and barometer data
sensor_fusion_ros_interface.py	Interface between the sensor fusion script and ROS
ukf.py [1]	Unscented Kalman filter (UKF) implementation
log_data.py	Responsible for logging sensor fusion data

Table 5: List of python scripts used in the ROS catkin package offboard control from the created ROS workspace

ROS package: Offboard control	
autonomous_flight.py	Responsible of execution of missions and autonomous flight
drone_control.py	Responsible for onboard state change from keyboard commands and the passing of setpoints and local pose to PX4
ground_control.py	Responsible for listening to keyboard commands which is passed to the program from the laptop for UAV control
loiter_pilot.py	Responsible for UAV control given setpoints from keyboard commands
waypoint_checking.py	Responsible for checking if the UAV has reached its destination
waypoint_generator.py	Responsible for creating waypoints to a given route
transformations_calculations.py	Responsible for calculating the orientation of the UAV in regard to the estimated pose of the ArUco marker board
uav_flight_modes.py	Responsible for arming, mode change, service calls and parameter changes of the UAV to the PX4
log_data.py	Responsible for logging of UAV data in tests

3.2.2.2 Structure of ROS nodes

A flowchart of the system used in offboard control can be seen in Figure 19. This gives a simplified overview of the main messages distributed between the nodes. In total of six nodes are used in this design. The core node is called ***drone_control*** which is set with 30 updates each second. This node is responsible for publishing set points to the PX4 as well as the state on the onboard system. The reason for having only one node publishing set points is for safety reasons so that no more than a single node can publish new set points to the UAV. The onboard states includes takeoff, loiter, return to home or some of the missions which have been defined to be executed during flight. Hence, through keyboard commands, the onboard state can be changed.

This is done using the node ***ground_control*** which is set with 10 updates each second. This node listings to keyboard commands and publishes these if an update occurs e.g a keystroke. The reason for having this as a separate node is that keystrokes do show up in a the main terminal which makes it hard to analyze important updates from PX4 and other nodes that publishes updates to the user from the terminal. Hence, a separate terminal will open for keyboard inputs and a main terminal with updates on the current state of the system will be used.

Another node which subscribes to updates from keystrokes is called ***loiter_pilot*** which is set with 20 updates each second. This node enables the user to take full control of the UAV using keystrokes which makes it much easier to debug the system. The possible commands from the user can be seen in Table 6. This table is split up into commands passed to the drone control and loiter pilot node. The latter will increment/decrement the current position of the UAV with 0.5 meters and increment/decrement the yaw angle by 5 degrees for each keystroke. The loiter pilot node also subscribes to the local position topic, but is not visualized in Figure 19.

All missions passed to the UAV will be controlled in the ***autonomous_flight*** node which is set with 10 updates each second. In this node different missions can be executed which is discussed in Section 3.2.2.4. These missions are started by the control node after passing numerical values to the terminal by keystrokes as seen in Table 6. This is done so that a number of missions can be executed for testing the performance of the UAV in the simulation as well as the implementation on

the Raspberry Pi.

Figure 19: Overview of the offboard control system. Nodes are labeled orange, onboard topics green, MAVROS topics blue and hardware gray. Arrows towards a node are subscriptions and arrows away from a node are publications

Moreover, this node listings to the current position of the UAV which will be used to determine when a set point has been reached from a predefined flight plan.

The use of ROS gives the opportunity to subscribe to services from the PX4. These services includes arming the UAV, change parameters like vertical or horizontal velocity and much more. This is beneficial because the parameters can be changed during flight in offboard control. This is done using a script called uav_flight_modes.py.

Furthermore, this node publishes the wanted ArUco marker board from which a pose estimation is wanted to be estimated. This is done to distinguish between the GPS to vision marker board, world markerb board on the ground and landing marker boards. Also the detection of the specified ArUco marker board will be returned as a Boolean to the autonomous flight node by a subscription to the ArUco board found topic. This is done so that the mission will not start until the wanted marker board has been located. Furthermore, the distance to the GPS2Vision marker board, its center and a boolean of whether the estimated pose is considered stable is passed to this node.

The ***marker_detection*** node will analyze the incoming images from the front and bottom camera and estimate the ArUco pose if any markers is located in the image. This node is set with 5 updates each second. The theory behind the pose estimation of the ArUco markers will be explained in Section 3.3. When the pose has been estimated, the result is published to other nodes in the system.

The node which uses the ArUco marker pose is called ***sensor_fusion*** which is set with 20 updates each second. This node takes the pose and optimizes it based on fusion with IMU and altimeter (barometer) data. This is done to ensure that the UAV still gets an updated pose estimation even though no markers being detected in the image. This is very important because the UAV uses the ArUco markers as its coordinate system. The implementation of sensor fusion will be discussed in Section 3.4. Moreover, this node also gets a boolean of whether the ArUco marker board has been detected, because it first initiates when reliable pose estimates from ArUco markers are being published.

As seen in Figure 19, the drone control node publishes set points to either global position (geographic coordinates), local position (Cartesian coordinates) or vision pose (Cartesian coordinates). The later is a very important aspect of this implementation which must be mentioned. When the UAV gets the pose from a GPS as either geographic or Cartesian transformed (UTM) coordinates as global or local respectively, the UAV uses this coordinate system for navigation. The PX4 auto pilot is relying on an extended Kalman filter for sensor fusion which can be altered from the parameter *EKF2_AID_MASK* from the list of services. This uses GPS positions as default, but can be changed to fuse the estimated vision pose instead. By using this configuration, the estimation of the local coordinate system can be changed during flight when the UAV is to navigate according to ArUco markers. This way, the use of the build in control schemes for pose control can still be used, but now the UAV navigates according to vision instead of GPS. Hence, a smooth transition from using GPS to vision and vision to GPS can be applied whenever needed. This is also one of the reasons why the pose estimation from the ArUco markers has to be very precise without much variance in its estimates. Otherwise, this could lead to an unstable system where the UAV could have trouble positioning itself according to the predefined pose.

3.2.2.3 GPS to vision transition

Because going from one coordinate system to another comes with a cost of an unstable system for a short amount of time, the parameters *MPC_XY_VEL_MAX*, *MPC_Z_VEL_MAX_DN* and *MPC_Z_VEL_MAX_UP* for horizontal and vertical velocities and *MC_ROLLRATE_MAX*, *MC_PITCHRATE_MAX* and *MC_YAWRATE_MAX* for angular velocities can be set to a very low value before changing between coordinate systems e.g GPS to vision. This ensures the UAV responds very slowly in the phase of instability caused by the local position and set point position are not aligned i.e. either one is publishing values from the old configuration due to a time delay. Then when the system has settled, the speeds are set back to its initial configuration yielding a smooth transition. This is simply done by inserting a time delay of a few hundreds of milliseconds after a change in the coordinate system to avoid this instability causing problems to the performance of the UAV.

However, this solution does no take into account the contribution of wind, which could make the UAV fly away because of decreased rates in angular as well as positional velocities. Moreover, because the ArUco marker board has its own coordinate system, the local coordinate system of the UAV could potentially be very different in regard to position which would yield errors in the build in Extended Kalman filter of the PX4 if big changes were made in position from one time instant to other.

A mitigation to this problem is to map the local pose of the UAV when it is about to change from using GPS to vision based navigation to that of the vision pose from the ArUco pose estimation. This is done by finding the rotation between the current pose of the UAV and that of the ArUco marker board. This is done in Equation 3.1.

$$r_{aruco2uav} = r_{aruco}^{T} \cdot r_{uav} \tag{3.1}$$

$$offset = \begin{bmatrix} x_{uav} & x_{aruco} \\ y_{uav} & y_{aruco} \\ z_{uav} & z_{aruco} \\ yaw_{uav} & yaw_{aruco} \end{bmatrix} \tag{3.2}$$

This rotation defines the difference in the orientation between the ArUco board located on the ground and the UAV before

initializing the GPS to vision transition. Furthermore, the position and yaw angle of the UAV is mapped to that of the ArUco marker board which can be seen in Equation 3.2. Because the roll and pitch angles are found negligible in size they are not used in the mapping process.

Now new updates from the pose can be used to transform the estimation from the ArUco marker board to that of the UAV offset which is seen in Equation 3.3.

$$t_{aruco} = \begin{bmatrix} -(offset_{x_{aruco}} - x_{aruco}) \\ -(offset_{y_{aruco}} - y_{aruco}) \\ -(offset_{z_{aruco}} - z_{aruco}) \end{bmatrix} \tag{3.3}$$

$$t_{uav} = r_{aruco2uav} \cdot t_{aruco} \tag{3.4}$$

This vector of the difference between the estimated ArUco position and the offset can be transformed to align to the initial coordinate system of the UAV before the GPS to vision transition. This is done in Equation 3.4.

$$x = offset_{x_{uav}} + t_{uav_x} \tag{3.5}$$

$$y = offset_{y_{uav}} + t_{uav_y} \tag{3.6}$$

$$z = offset_{z_{uav}} + t_{uav_z} \tag{3.7}$$

$$yaw = offset_{yaw_{uav}} + t_{uav_{yaw}} \tag{3.8}$$

When the position has been transformed to align to the original UAV coordinate system they are simply added to the offset of the UAV using Equation 3.5, 3.6 and 3.7. Because both the UAV and ArUco marker board uses coordinate systems with the z-axis pointing up, the yaw angle is just added to the offset as seen in Equation 3.8 without any transformation needed. These steps are performed in a method called transformations calculations which can be seen in GitHub. Hence, the latter solution will be used in the transition from using GPS to vision based navigation to ensure a stable and reliable transition.

This transition of using GPS to vision involves a number of steps. After reaching the target GPS location, the ArUco marker board must be localized, then the UAV has to navigate to the board still using GPS coordinates and finally the GPS to vision transition can be initialized if the UAV is close enough to the board and estimated poses are stable. A flowchart of this can be seen in Figure 20.

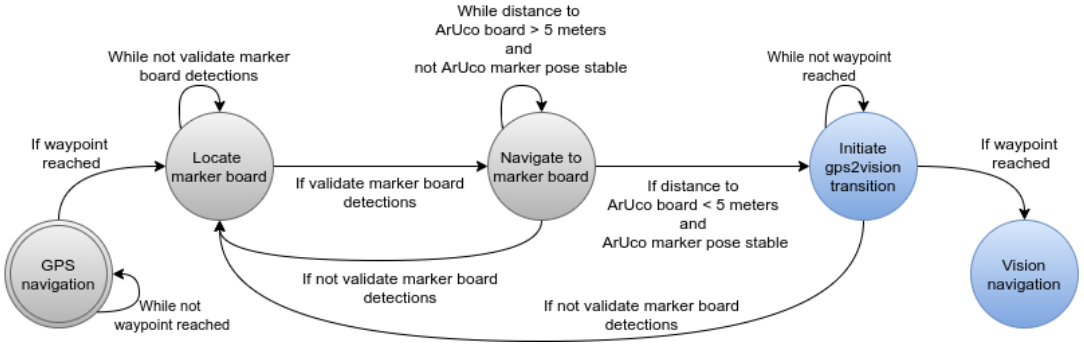


Figure 20: Illustration of the substates when initializing GPS to vision based navigation. Gray boxes defines the use of GPS for pose estimation and blue boxes vision based estimation

As it may be seen, the UAV starts by flying towards a predefined waypoint. The approach for checking if the waypoint has been reached, is to use the current pose of the UAV and subtracting the target setpoint. Here only the x, y, z and yaw values are used. Then the euclidean distance of the four values are computed and if this value is below a threshold (default value is 0.25), the waypoint is reached and the UAV begins to locate the marker board.

Because there is a risk that the UAV will not see the ArUco marker board located on the wall at its final position, the UAV is set to locate the board by moving in a grid structure of predefined waypoints until the board is found. These waypoints are set from the offset of the final position where the UAV will move in steps of one meter and rotate left to right in steps of 45°. The UAV will complete this list of waypoints and repeat the process if the board has not been found. If the board is still not found after a predefined number of repetitions, the UAV will land and go into idle state and wait for further commands. If the board is found, the UAV will begin to navigate to the ArUco marker board. Because only a single detection of the board could be too unreliable, a sequence of positive detections of the board must be found. This is accomplished by only checking that the board is visible once a second and pushing the result as a boolean on a list with a size of five, where the validate marker board will be true if 80% of the values on the list have positive observations of the board. This variable is called validate marker board detections in Figure 20.

Now the UAV will navigate towards the board while keeping its orientation to the board by using Equation 3.9.

$$error = \frac{\frac{imageWidth}{2} - boardCenter_x}{\frac{imageWidth}{2}} \tag{3.9}$$

$$yaw = error \cdot 30 \tag{3.10}$$

Here the goal is to have the board be centered in the middle of the image so that the UAV keeps its orientation towards the marker board. This results in a normalized error which will be used in a simple P-controller seen in Equation 3.10. Here the maximum change in yaw angle is set to 30° which will be based on the normalized error going from zero to one. This limit is chosen to mitigate the possibility of oscillations in the orientation. Now this yaw angle is added to the setpoint yaw, which will keep the UAV orientated towards the board. From results in Section 4.1.1 and 4.1.2, the maximum allowed distance away from the board when initiating GPS to vision transition is found to be approximately five meters with the selected camera resolution and size of the board discussed in Section 3.3. This is done to make sure that pose estimates from ArUco markers are reliable. Hence, both the distance to the board and rolling average of latest estimates are used as validation of the estimated pose. The latter is defined by the variable called ArUco marker pose stable. If these conditions are satisfied, the UAV will initiate GPS to vision transition as seen in Figure 20. What goes for both these two substates is that they will fall back to locate the marker board if they loose sight of the board.

In the initiate GPS to vision transition, the UAV will use vision pose estimations of the board for navigation using the front camera towards the GPS2Vision marker board located on the wall until it is at a predefined location in front of it and then switching to using the bottom camera for the actual vision based navigation which is the last step in the GPS to vision procedure. In the initiate GPS to vision step, the UAV has also been set to keep its orientation towards the marker board, but now in a different way. Because the pose estimation is reliable at this point, knowledge of the placement of the GPS2Vision board on the wall can be used along the current pose of the UAV to calculate the new yaw angle of the UAV to keep its orientation towards the board. This is done in Equation 3.11, 3.12 and 3.13 where the $aruco_x$ and $aruco_y$ are the current estimate of the position of the board in the x and y direction.

$$\Delta_x = aruco_x - 3.2 - 0.5 \tag{3.11}$$

$$\Delta_y = aruco_y - 3.0 \tag{3.12}$$

$$yaw = atan2(\Delta_y, \Delta_x) \tag{3.13}$$

Here 3.2 meters in Equation 3.11 is the distance of the GPS2Vision marker board away from the origin of the board located on the ground and 0.5 meters is to add it to the center of the board. The same goes for Equation 3.12. The atan2 function is used to calculate this angle in Equation 3.13, where 180° will be added to this value if the found yaw is below zero.

3.2.2.4 Offboard control system

In order for easy testing of the autonomous system, a set of predefined missions have been set up, which can be executed during a simulation as well as real life testing. For real life execution of these commands, the UAV must be within range of the wireless network or hot-spot created for communication between laptop and the Raspberry Pi on the UAV. However, these commands could also be triggered by using the transmitter for long range communication, where the Raspberry Pi, using ROS, was set to listening for activations of certain switches (high/low) to start the execution of missions from within the script. However, because the UAV will be close to the pilot in this case, wireless network communication is sufficient for activating missions and control the UAV in offboard mode. These keyboard commands can be seen in Table 6.

Keypress	Node	Action
t	drone_control	Takeoff + offboard + loiter mode
h	drone_control	Move the drone to home and land
l	drone_control	Switch to loiter control
wasd	loiter_pilot	Forward, left, back and right respectively
qe	loiter_pilot	Rotate left or right (yaw) respectively
zx	loiter_pilot	Decrease/increase altitude respectively
0	drone_control	Move to GPS locations from vision (*mission*)
1	drone_control	Return to landing station one (*mission*)
2	drone_control	Return to landing station two (*mission*)
3	drone_control	Return to landing station three (*mission*)
4	drone_control	GPS2Vision ArUco pose estimation test (*mission*)
5	drone_control	Hold ArUco pose test (*mission*)
6	drone_control	GPS2Vision test (*mission*)
7	drone_control	Vision navigation test (*mission*)
8	drone_control	Landing test (*mission*)

Table 6: Table of all possible commands from the GCS

Here the UAV can be set to takeoff which will set it into offboard mode and then loiter mode. Because the UAV must get setpoints for the new pose before switching to offboard mode, these are past before this transition as well as afterwards. The UAV will be in the *takeoff* state until the waypoint is reached. Then it will switch to *loiter* state where the UAV can be controlled using keyboard commands as seen in Table 6 from the node ***loiter_pilot***. A simplified state machine of these transitions between onboard states of the UAV can be seen in Figure 21. The UAV can be set to return to its home position when it is in *loiter* state. The home position is defined as the place from where the UAV was initiated. This is done in the *Home* state which will result in the UAV switches to *Idle* state when the final home position waypoint has been reached. From *Loiter* state, a number of missions can be executed.

A number of tests will be performed to evaluate the autonomous system in the simulations. Here the pose estimation of the GPS2Vision board located on the wall must be evaluated to see how reliable the pose estimation is when moving further away from the board from the *GPS2Vision ArUco pose estimation test*. This is a critical test because this gives an indication of the maximum allowed distance away from this board, a safe GPS to vision transition can be performed. This is performed in Section 4.1.1 from where results of this have been used in Section 3.2.2.3 as already discussed.

The *Hold ArUco pose estimation test* is performed to see how well the UAV is capable of keeping its pose in front of the ArUco marker boards. The results can be seen in Section 4.1.3. This gives an indication of the performance of the control schemes for pose control used within the PX4 autopilot. Moreover, this pose will also be compared to that of the ground

truth for performance evaluation. Results from this test will be used in the creation of sensor fusion discussed in Section 3.4.

The *GPS to vision test* was performed to evaluate the performance of the GPS to vision transition discussed in Section 3.2.2.3. This test includes all steps from using GPS, to locate the marker board, navigate to the marker board and initialize the GPS to vision transition. Results of this can be seen in Section 4.1.4.

In the *Vision based navigation test*, the performance of using the bottom camera to navigate the UAV is evaluated. In this test, sensor fusion is used in the indoor environment with a number of different configuration of markers located on the ground. Results can be seen in Section 4.1.5.

Finally, the vision based landing procedure is tested in *Landing test*. Here the UAV will move between landing stations and land in front of the landing boards where the final landing position will be compared to that of the ground truth to evaluate the performance from the wanted setpoint location. Results can be seen in Section 4.1.6

From being in the *Loiter* state, the UAV can be set to return to one of the landing stations. That way the UAV can be set to execute a route and then afterwards return to the landing stations specified by one, two and three. When the UAV has landed, it will be in *Vision landed* state, where the pose is still calculated using vision from the ArUco marker landing boards. When wanted, the UAV can again execute a route by making it return to using GPS coordinates after moving outside the vision navigation area. This can be seen in Figure 21.

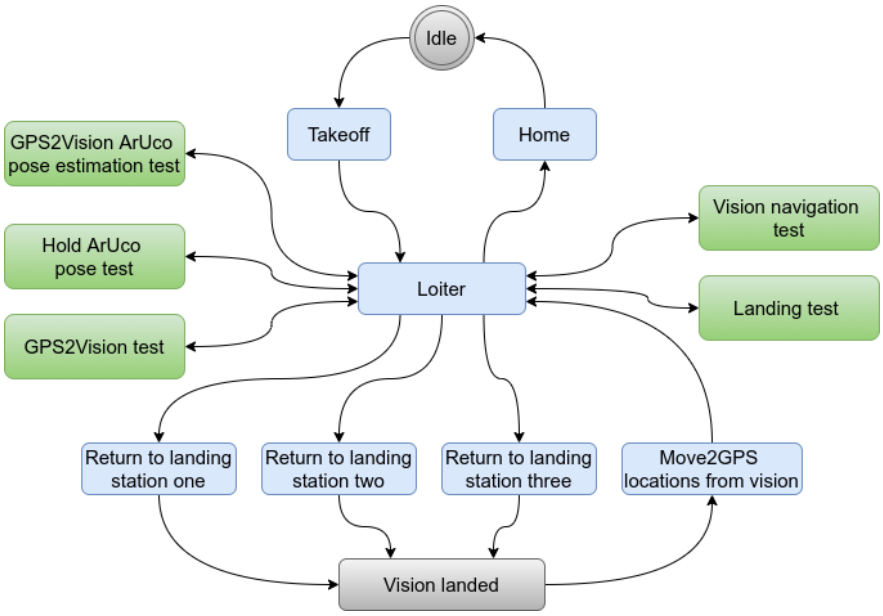


Figure 21: A simplified state machine of the onboard states of the UAV. Green boxes defines the states with the intention of testing the autonomous flight system. Blue boxes defines the actual autonomous system from where the UAV can be controlled from either keyboard commands in Loiter or return to a landing station from using GPS or being at a landing station and then move to a GPS location from using vision to GPS coordinates

A bash script has been created for automatic execution of tests which can be seen in Listing 2. This is done to make the execution of tests scalable and easy to perform. As seen in Listing 2, a directory is created if not already defined. Data from each test is saved in the data directory which is copied to the test directory for each execution. This is done so that all data from all simulations can be analyzed afterwards. A predefined number of tests can be set which will be executed in each run defined by iterations. The way the program is set to terminate in each run, is to call *rospy.signal_shutdown("Test completed!")* from within the end of the code e.g GPS2Vision_test from the autonomous_flight.py script in offboard control. However, to be able so make a single node terminate the entire program, *required="true"* must be added in the initialization of the node from where the shutdown takes places. This can be seen in the created launch file gazebo_sim_v1.0.launch.

This roslaunch file is used throughout the simulations which can be executed directly from the terminal. The way to start a simulation can be seen in line 12 in Listing 2. Here the wanted world can be set as input to the program, along with the wanted onboard state of the UAV. By giving this state as input to the program, tests can be started directly without having to initiate them by using keyboard commands. This gives the possibility of starting roslaunch in headless mode, which means that no GUI is used. The program will simply be executed silently as a background process which also requires less resources to run. Furthermore, the starting pose of the UAV is giving as input as well. This pose will have to be changed accordingly to the appropriate test from where the test is to be executed.

```
1  #!/bin/bash
2  #Specify data analysis directory, name of data file, name for the test and number of runs
3  DIRECTORY="../../data/analyse_vision_navigation_pose_estimation/test5_one_pattern_missing_mark
   ↪  ers_wear_board/"
4  data_file="../../data/sensor_fusion_data.txt"
5  test_name="test"
6  iterations=20
7  #Run for nunmber of test times
8  if [ ! -d "$DIRECTORY" ]; then echo "Dir does not exist"
9  mkdir -p $DIRECTORY else echo "Dir does exist"
10 fi
11 for (( c=1; c<=$iterations; c++ ))
12 do roslaunch px4 gazebo_sim_v1.0.launch
   ↪  worlds:=optitrack_big_board_onepattern_missing_markers_wear.world
   ↪  drone_control_args:="vision_navigation_test" x:=-3.0 y:=0.0 headless:=true gui:=false
13 cat $data_file > "$DIRECTORY$test_name$c.txt"
14 echo "Test $c completed!"
15 done
```

Listing 2: Bash script for automatic execution of tests

3.2.2.5 *Software setup*

Additional software will have to be installed and configured in order to use ROS. For performing conversions between UTM, geographic, local Cartesian coordinates etc. GeographicLib is used. This is mandatory in order to use MAVROS along with PX4. This can be installed using Listing 3.

```
1 #Update system before installation
2 $ Sudo update
3 #Fetch the sofware, change permissions and execute script from terminal
4 $ wget https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scripts/install_geograph
  ↪  iclib_datasets.sh
5 $ sudo chmod 755 install_geographiclib_datasets.sh
6 $ sudo ./install_geographiclib_datasets.sh
```

Listing 3: Installation of the Geographiclib datasets as dependencies for MAVROS

Because the simulations will be running on an Ubuntu 18.04.5 LTS (Bionic Beaver) 64-bit desktop operating system and server edition for the Raspberry Pi, ROS melodic will be used. Furthermore, the Raspberry Pi does not need the full installation of ROS due to the fact that no GUI is used as the communication between the Raspberry Pi and Laptop will be through SSH from the terminal. Therefore, ROS base will be installed on the Raspberry Pi and the full desktop version installed on the laptop.

The system must be configured to accept packages from ROS along with setting up the cryptographic keys for package authentication. When this is done, ROS can be installed which is seen in Listing 4. Here the full edition of MAVROS will be installed for communication as already discussed.

```
#Setup your sources.list
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
#Set up your keys
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
#Install ROS dependencies
$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential python-rosdep
#Installation
$ sudo apt install ros-melodic-desktop-full
#Install mavros along with extra packages
$ sudo apt install ros-melodic-mavros ros-melodic-mavros-extras -y python-catkin-tools -y ros-melodic-hector-gazebo-plugins dvipng
```

Listing 4: Installation of ROS with required dependencies

For easy debugging of the system, rqt is installed. This is a software framework which includes plugins for varies GUIs e.g plotting, graphs and image viewing. This will utilize the possibility of visualizing the images from the front and bottom cameras of the UAV, to see the detected ArUco boards. Moreover, to enable ROS to access images from attached cameras from the USB, the USB camera plugin will be installed on the Raspberry Pi using line 4 in Listing 5.

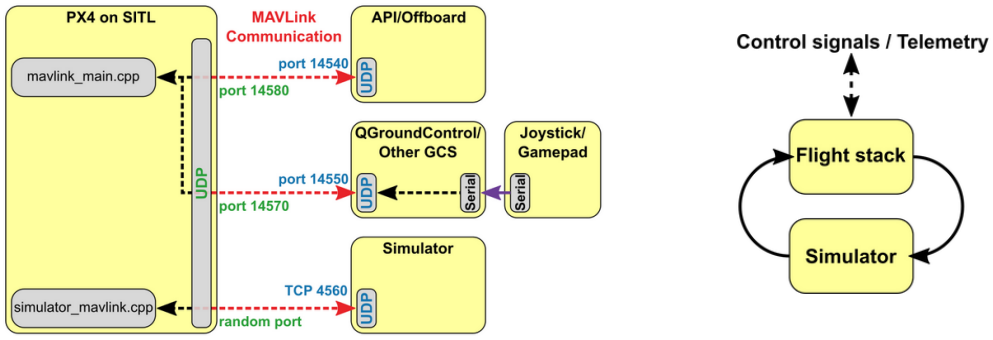
```
#Install packages for visualization of video stream (ros rqt)
$ sudo apt-get install ros-melodic-rqt ros-melodic-rqt-common-plugins
#Install packages for USB readings using ROS
$ sudo apt-get install ros-melodic-usb-cam ros-melodic-image-transport ros-melodic-image-transport-plugins ros-melodic-image-transport-plugins ros-melodic-camera-info-manager ros-melodic-vrpn-client-ros
#Install packages fto be able to execute scrips
$ sudo apt-get install liblapack-dev gfortran libblas-dev libxml2-dev libxslt-dev python-dev python-pip
$ pip install pathlib pymavlink pandas scipy opencv-contrib-python matplotlib numpy
```

Listing 5: Dependencies for created ROS nodes and visualization

For easy creation of paths for files *pathlib* is used. For communication with MAVROS a python version of *pymavlink* is installed. *Pandas* and *matplotlib* will be used for data analysis for plotting, *scipy* in the creation of the Unscented Kalman filter and *opencv-contrib-python* for enabling the use of OpenCV for computer vision algorithms e.g detecting ArUco marker boards where the *contrib* edition of OpenCV includes the mandatory ArUco libraries. Lastly *numpy* is used for data structuring.

3.2.3 PX4 autopilot

PX4 is an open source control software for unmanned vehicles. PX4 uses software in the loop (SITL) for simulations of a given unit before real world testing. The flight stack runs on a computer which could be any unit on the same network. In the simulations, PX4 will get UAV data from the simulation environment in Gazebo. A high level view of this can be seen in Figure 22.



(a) *(b)*

Figure 22: Illustrations of the message handling in the SITL simulation environment [27]

In Figure 22b, a high level view of the communication between the PX4 flight stack and simulation environment can be seen. Sensor readings like GPS, IMU data etc. are passed to the PX4 flight controller which returns control outputs (motor actuator commands) to the simulation according to sensor readings.

The transmission of messages between PX4 and external programs happens using the user datagram protocol (UDP) ports for MAVLink communication. This is illustrated in Figure 22a. Here a number of ports are associated to different applications. The UDP Port 14540 is used for offboard control which will be python scripts running in the simulation or on the Raspberry Pi for autonomous flight using information from sensors e.g. vision feedback. The UDP port 14550 is used for communicating with GCS. QGroundControl will be used which offers many features like sensor calibration, radio setup and calibration of transmitters, predefined air frame selection for different drones etc. Moreover, parameters for position and attitude control, velocity limits and the like can be changed during flights. However, these parameters can also be changed using ROS in the offboard control script. The last port, UDP 4560, will be used for transmission of messages between PX4 and the simulation environment.

3.2.3.1 Software setup

Whenever a new terminal is opened, the variables of the software have to be exported with the ROS workspace and firmware sourced so that the PX4 simulation can be started. In order to do this in one step, a script has been created as seen in Listing 6.

```
#!/bin/bash
$ source devel/setup.bash
$ source ~/PX4_SITL/Firmware/Tools/setup_gazebo.bash ~/PX4_SITL/Firmware
  ↪  ~/PX4_SITL/Firmware/build/px4_sitl_default
$ export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/PX4_SITL/Firmware
$ export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/PX4_SITL/Firmware/Tools/sitl_gazebo
$ export GAZEBO_PLUGIN_PATH=${GAZEBO_PLUGIN_PATH}:~/catkin_ws/devel/lib/
```

Listing 6: Setup gazebo and PX4 from ROS workspace

This script can either be run from the ROS workspace or past to the *.bashrc* file to be executed every time a new terminal is opened for easy initialization.

This implementation of PX4 SITL requires firmware vision 1.11.0 to work correctly. This setup relies on the PX4 firmware being installed in the home directory of the user, but can be modified to another installation directory with minor configurations. A walkthrough of these initial steps for installing and setting up of the firmware can be seen in Listing 7.

```
#Update system
$ sudo apt update
#Initialize PX4 SITL environment from your home folder
$ cd ~
$ mkdir PX4_SITL
$ cd PX4_SITL
$ git clone https://github.com/PX4/Firmware.git --branch v1.11.0
$ cd Firmware
$ git submodule update --init --recursive
#Make symlink to user software models to PX4 SITL
$ cd software/ros_workspace/
$ ln -s $PWD/PX4-software/init.d-posix/*
  ↪ /home/$USER/PX4_SITL/Firmware/ROMFS/px4fmu_common/init.d-posix/
$ ln -s $PWD/PX4-software/mixers/* /home/$USER/PX4_SITL/Firmware/ROMFS/px4fmu_common/mixers/
$ ln -s $PWD/PX4-software/models/* /home/$USER/PX4_SITL/Firmware/Tools/sitl_gazebo/models/
$ ln -s $PWD/PX4-software/worlds/* /home/$USER/PX4_SITL/Firmware/Tools/sitl_gazebo/worlds/
$ ln -s $PWD/PX4-software/launch/* /home/$USER/PX4_SITL/Firmware/launch/
#Build the PX4 SITL firmware
$ cd /home/$USER/PX4_SITL/Firmware
$ DONT_RUN=1 make px4_sitl_default gazebo
```

Listing 7: Installation and setting up of the PX4 SITL firmware

Here the PX4 firmware is fetched and installed in a directory called *PX4_SITL* from where dynamic links have been created from a ROS workspace. This configuration was made to easily link updates made in files from *init.d-posix*, *mixers*, *models*, *worlds* and *launch* directories. The *init.d-posix* directory is used for parameter initialization for the vehicle at startup which contains files for the used *SDU drone* to be used in simulations. The *mixers* directory contains files for the motor configuration of the vehicle. In this case it would be configurations for an hexarotor drone. The directory for *models* contains files to be used in simulations like the *SDU drone* seen in Figure 17c and models created in Section 3.1. The *worlds* directory includes all world files to be used in the simulation which were defined in Section 3.1. Lastly, the *launch* directory stores the launch files to be used for running the PX4 SITL simulation wrapped in ROS.

3.3 Pose estimation using ArUco markers

This section deals with pose estimation of ArUco marker boards to be used as the coordinate system for the UAV for vision based navigation.

A number of different visual markers could be used for pose estimation e.g ArUco markers, AprilTag or QR-codes. However, to achieve real-time performance, the latter cannot be used due to large amount of data in the marker. This is not the case with ArUco markers which also yields good performance in regards to detection accuracy and a low error rate [19].

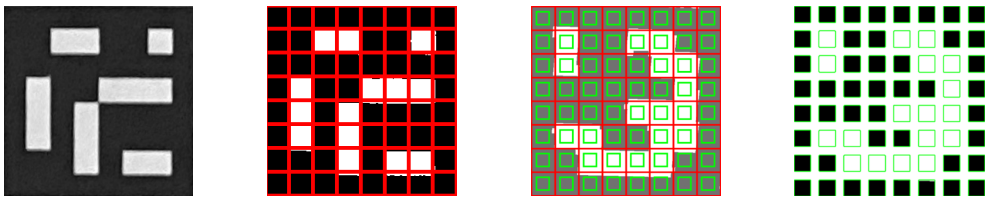
Another method using convolution with complex kernel for detecting n-fold edges of markers could be used [23]. This n-fold marker could be placed on the ground and used as a helipad for the UAV which could navigate to it from a very high altitude.

Because the ArUco markers have been shown to have high performance along with libraries for ArUco marker boards for better precision, ArUco markers will be used for pose estimation [19]. Furthermore, these estimations are performed using OpenCVs marker detection and pose estimation for ArUco markers.

3.3.1 ArUco marker detection

The ArUco marker consists of a black border with an inner binary matrix (bits) which defines its ID. This inner binary matrix is called the binary codification of the marker. To detect a marker, candidates in the image are first analyzed where adaptive thresholding is applied to the image and afterwards extracting contours which have to be square in order to be

categorized as a marker candidate. When the candidates have been found, a perspective transformation is applied to each of them to remove distortion in the images. This can be seen in Figure 23a where a marker candidate is given to the algorithm for further processing.



(a) *(b)* *(c)* *(d)*

Figure 23: Illustration of the procedures of ArUco marker detection [2]

In this step, Utsu thresholding has been used. Then the image can be divided into cells which corresponds to the predefined size of the marker (bits) and corresponding border size. In each cell, the number of black and white pixels are determined in order to categories the candidate marker to belong to a specific dictionary where it is finally evaluated to be an ArUco marker or not. These steps can be seen in Figure 23b, 23c and 23d.

3.3.2 ArUco pose estimation

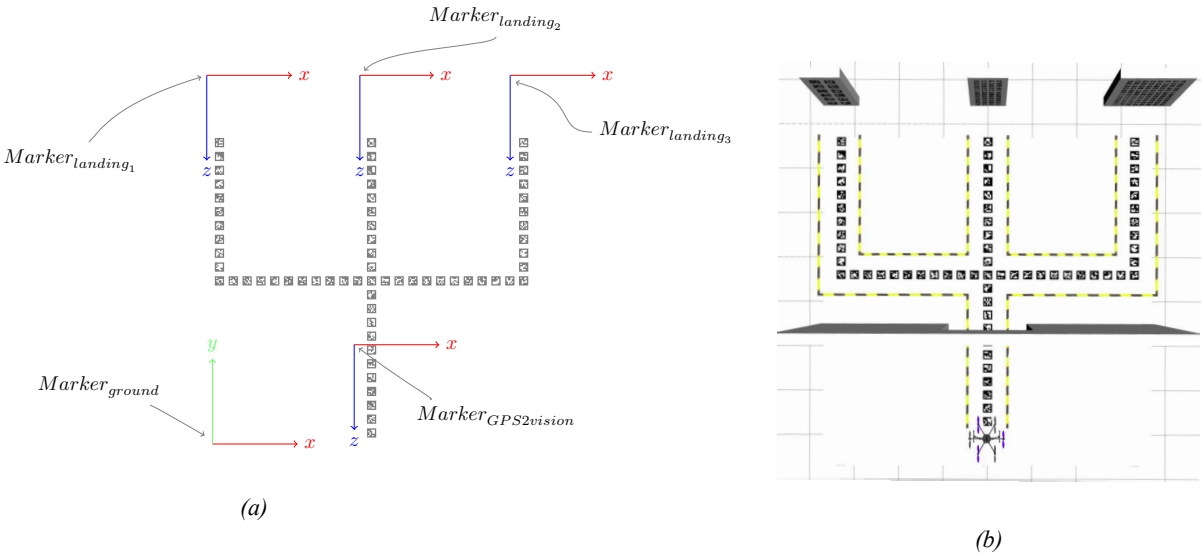
Knowing the size of the marker defined when creating the ArUco marker board, the intrinsic and corresponding distortion parameters in the camera and the found corners of each marker, a pose estimation can now be performed. For pose estimation, OpenCV uses an implementation of Levenberg-Marquardt optimization for solving the perspective-n-point (PnP) which estimates the pose of a camera in respect to the ArUco board giving a set of 3D points in the board reference system and there corresponding 2D projections in the image. The model for this can be seen in Equation 3.14 and simplified in Equation 3.15.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.14}$$

$$s \cdot p_c = K \cdot [R|T] \cdot p_w \tag{3.15}$$

Here f_x and f_y defines focal lengths, c_x and c_y are the principle points which are expected to be the center of the image, γ the skew parameter and s a scaling factor for the image. The two vectors p_c and p_w defines the 2D and 3D points for the projection in the image and known locations of the marker corners respectively. Hence, the rotation matrix R and translation vector T can be estimated, which yields a rotation and translation of the board in respect to the camera [13].

As defined in Section 3.1, the UAV will use the marker located on the ground as the coordinate system. Because the GPS2Vision marker board is located on the wall to the entrance of the building used for GPS to vision transition and three landing markers used when the UAV is to land, the $Marker_{GPS2vision}$, $Marker_{landing_1}$, $Marker_{landing_2}$ and $Marker_{landing_3}$ must be transformed to align with the coordinate system on the ground. An illustration of the placement of these marker boards visualized from above can be seen in Figure 24a.



(a) *(b)*

Figure 24: Visualization of the coordinate systems of the ArUco marker boards in the simulation seen from above. The actual coordinate systems can be seen in Figure 24a and an image of the simulation environment in Figure 24b for easy comparison

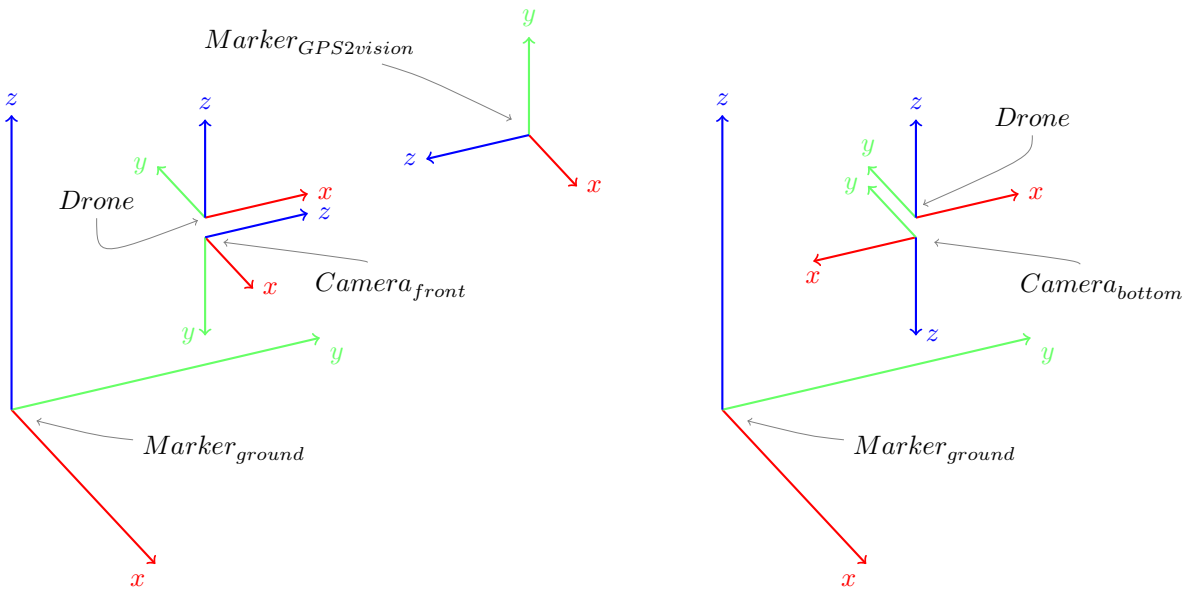
It may be noticed that the GPS2Vision marker board and all landing marker boards have the same orientation and differs only in translation. In this configuration, the bottom camera of the UAV is used for pose estimation of the ground marker board and front camera for the GPS2Vision and landing markers boards. This yields two different configurations as seen in Figure 25. In Figure 25a, the UAV is considered to hover in front of the GPS2Vision marker board. Here four coordinate systems are illustrated namely the marker board located on the ground, the UAV, the front camera attached on the UAV and the GPS2Vision marker board. To get the pose of the UAV in respect to the ground marker, the rotation and translation of the camera w.r.t the marker board must be found using Equation 3.16 and 3.17. Here the transpose of the rotation vector named r_{vec} is multiplied with the translation vector t_{vec} to give a translation from the camera to the marker board. This has to be done because the output r_{vec} and t_{vec} from the pose estimation using OpenCV are given from the marker board w.r.t the camera [29]. This can be seen in Equation 3.16 with the rotation matrix in Equation 3.17 transformed to euler angles from the Rodrigues rotation matrix.

$$t = -r_{vec}^{T} t_{vec} \tag{3.16}$$

$$r = r_{vec} \tag{3.17}$$

This initial step goes for both cameras. Now to get the position of the UAV w.r.t the ground marker, Equation 3.18 is used.

$$P_{Marker_{Ground}}^{UAV} = T_{Marker_{ground}}^{Marker_{GPS2vision}} \cdot T_{Camera_{front}}^{UAV} \cdot t \tag{3.18}$$



(a) *(b)*

Figure 25: Visualization of the coordinate systems of the ArUco marker boards in the simulation seen from a 3D view with two camera configurations as seen in Figure 25a and 25b for the front and bottom camera respectively

Here a transformation of the position of the GPS2Vision marker board w.r.t the ground marker is multiplied with a transformation of the UAV w.r.t the front camera which is then multiplied with the translation vector from the front camera to the marker. The same principle goes for Equation 3.19 according to the $Marker_{landing_1}$, Equation 3.20 for the $Marker_{landing_2}$, Equation 3.21 for the $Marker_{landing_3}$ and Equation 3.22 where the later uses the bottom camera for pose estimation of the ground marker.

$$P^{UAV}_{Marker_{Ground}} = T^{Marker_{landing_1}}_{Marker_{ground}} \cdot T^{UAV}_{Camera_{front}} \cdot t \tag{3.19}$$

$$P^{UAV}_{Marker_{Ground}} = T^{Marker_{landing_2}}_{Marker_{ground}} \cdot T^{UAV}_{Camera_{front}} \cdot t \tag{3.20}$$

$$P^{UAV}_{Marker_{Ground}} = T^{Marker_{landing_3}}_{Marker_{ground}} \cdot T^{UAV}_{Camera_{front}} \cdot t \tag{3.21}$$

$$P^{UAV}_{Marker_{Ground}} = T^{UAV}_{Camera_{bottom}} \cdot t \tag{3.22}$$

To get the rotation the same equations are used, bot now with r substituted for t to get the proper rotation for each of the marker boards. In this case only the rotational part of the transformation matrix is considered. These equations for the transformations of the pose will be used for the different configurations in regard to the wanted action from the UAV e.g GPS to vision, indoor navigation and landing.

3.3.3 Camera calibration

Due to the possibility of tangential and radial distortion, the cameras must be calibrated which is done for both the Logitech C270 HD used as front camera and Raspberry Pi V2 used as bottom camera.

Radial distortion will have the effect of making straight lines appear curved which will be more visible further away from the center of the image. To correct for this error, Equation 3.23 and 3.24 will be used. Here k_1, k_2 and k_3 are radial distortion coefficients and $r = \sqrt{x^2 + y^2}$ is the distance to the optical axis.

$$x_{corrected} = x(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \tag{3.23}$$

$$y_{corrected} = y(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \tag{3.24}$$

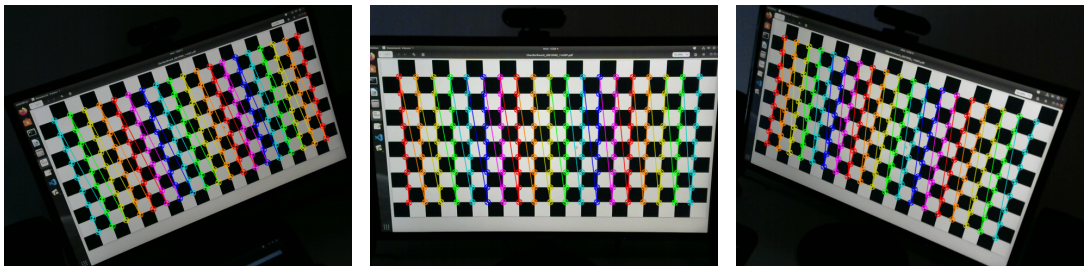
Tangential distortion occurs because the lens of the camera not being perfectly parallel aligned with the image plane. This can cause some areas within the image to look nearer than expected. Equations for handling this issue can be seen in Equation 3.25 and 3.26 where p_1 and p_2 are tangential distortion coefficients.

$$x_{corrected} = x + (2 \cdot p_1 \cdot x \cdot y + p_2(r^2 + 2 \cdot x^2)) \tag{3.25}$$

$$y_{corrected} = y + (2 \cdot p_2 \cdot x \cdot y + p_1(r^2 + 2 \cdot y^2)) \tag{3.26}$$

To find the distortion coefficients as well as the camera matrix, camera calibration based on chessboards will be used. Like ArUco markers, chessboards consists of a square pattern of black and white boxes. However, compared to ArUco markers these boxes do not contain any bit codification which makes them completely black and white. Because no separation is needed between these boxes, more corner points can be estimated. This is the reason for using these patterns for calibration which gives the possibility of a better calibration due to more point estimates.

This calibration will be based on twenty images taken from different orientations using a 9×19 chessboard as seen in Figure 26. This board has been chosen in order to fill out most of the image with boxes for corner estimations.



(a) *(b)* *(c)*

Figure 26: Illustration of some of the images taking using the front camera of the UAV in the calibration of the camera. Here circles can be seen on the detected corners of the boxes in the chessboards using images of 640×480 *pixels*

$$C_{front} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 821.7123 & 0 & 334.0420 \\ 0 & 821.0677 & 240.4507 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

$$P_{front} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} = \begin{bmatrix} -0.007 & 0.9038 & 0.0000 & -0.0027 & -2.6634 \end{bmatrix} \tag{3.28}$$

$$C_{bottom} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 501.8636 & 0 & 318.1745 \\ 0 & 502.3949 & 240.5029 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.29}$$

$$P_{bottom} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} = \begin{bmatrix} 0.1375 & 0.0961 & -0.0016 & -0.0025 & -1.1048 \end{bmatrix} \tag{3.30}$$

The outcome of the calibration can be seen in Equation 3.27 and 3.29 for the camera matrix of the front and bottom camera respectively and distortion parameters in 3.28 and 3.30 for the front and bottom camera respectively. Here the focal length

f_x and f_y has been estimated along the optical center c_x and c_y. These values will only be used on the real UAV and not in the simulation where no camera calibration will be performed.

3.4 Sensor fusion

Chances are that ArUco markers placed on the ground will not be visible at all times. This can be due to changes in light conditions and dirt on or wear of the markers. This could be fatal, because the UAV in offboard control will rely on constant updates from the pose of ArUco markers when flying indoors. To mitigate this situation sensor fusion will be implemented.

This section deals with the implementation of sensor fusion for pose estimation. Section 3.4.1 is about the theory behind the Unscented Kalman filter (UKF) and why using this would be beneficial in regard to other methods e.g. the extended Kalman filter (EKF). Section 3.4.2 deals with the implementation of the UKF.

3.4.1 Unscented Kalman Filter

The Kalman Filter (KF) requires linear models. However, these models are rarely seen in reality and especially not in the dynamics of a Aquacopter. A way to accommodate for this problem could be to approximate the linear model by linearization using an EKF. The basic idea behind this approach is to use the mean of a non-linear Gaussian function and linearize at this single point through Taylor expansion. However, because the EKF only offers first order approximations of the optimal terms based on a linearization for a nonlinear system of the estimated state distribution of a Gaussion Random Variable (GRV), large errors can occur if functions turn out to be very nonlinear. Furthermore, calculations of the Jacobian are needed which can introduce errors if not performed correctly. Because of these limitations, this method is considered suboptimal.

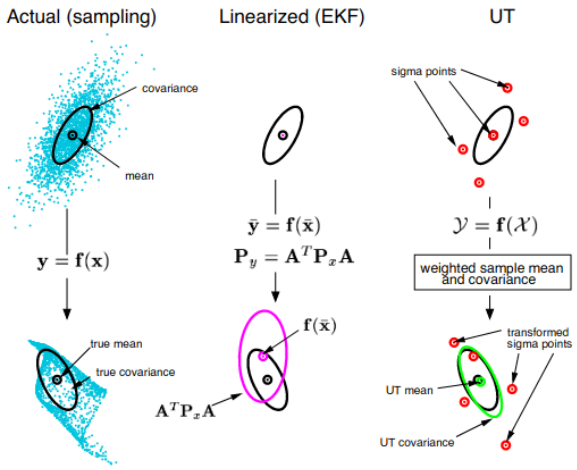


Figure 27: Illustration of the UT and how the use of this method in the UKF performas better than EKF in regard to better mean and covariance estimation [12]

Another method called UKF takes these considerations into account. The state distribution is still represented using a GRV. However, now a number of sample points around the mean called *sigma points* are chosen which better captures the true mean and covariance of the GRV of the non-linear system using the unscented transformation (UT). An illustration of the superiority of using the UT instead of a first-order linearization in the EKF can be seen in Figure 27. For Gaussian functions, the use of the UT results in approximations which are accurate to the third order and at least second order for non Gaussian functions for all nonlinearities. For a more in-depth explanation of the UKF, the paper [12] is recommended.

Because of the discussed optimizations in regard to the EKF, the UKF will be used in the implementation of sensor fusion for pose estimation of the UAV.

3.4.2 *UKF implementation*

Because the appropriate choose for parameters of sigma points may give performance improvements, these have been chosen based on a discussion of the UKF in [17, p. 354]. Three initial parameters have to be chosen namely α, β and k. α determines the amount of spread of the sigma points from the mean of $\bar{x}$ which is chosen to be 0.04, β incorporates prior knowledge of the distribution of x which is set to 2 and k is defined as a secondary scaling parameter set to 15. The equations for the calculations of *sigma points* can be seen in Equation 3.35, 3.36 and 3.37 in Pseudocode *UKF calculate sigma points* with associated weights seen in Equation 3.31, 3.32, 3.33 and 3.34. To calculate the weights, a scaling parameter $\lambda = \alpha^2(L + k) - L$ where L is the number of states in the system is used. The superscipt (m) and (c) in the weights defines the mean and covariance of the weights respectively. The initialization of the UKF can be seen in Pseudocode *UKF initialization*.

Pseudocode: UKF initialization

Given scalar values for initialization of α, β, k and L
$\alpha \leftarrow Chosen\,alpha$
$\beta \leftarrow Chosen\,beta$
$k \leftarrow Chosen\,k$
$L \leftarrow Number\,of\,states$
Given vectors of size L for x with a matrices of size $L \times L$ for Q and P
$x \leftarrow Initial\,state$
$Q \leftarrow Process\,noise$
$P \leftarrow Initial\,covariance$
Initialize function which predicts next state
$iterate \leftarrow iterateFunction(x, dt)$
Initialize weights for associated sigma points
$W_0^{(m)} \leftarrow \frac{\lambda}{L+\lambda}$ (3.31)
$W_0^{(c)} \leftarrow \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta)$ (3.32)
for $i \leftarrow 1\,to\,2L$ **do**
    $W_i^{(m)} \leftarrow \frac{1}{2(L+\lambda)}$ (3.33)
    $W_i^{(c)} \leftarrow \frac{1}{2(L+\lambda)}$ (3.34)

Pseudocode: UKF calculate sigma points

$\mathcal{X}_0 \leftarrow \bar{x}$ (3.35)
for $i \leftarrow 1\,to\,L$ **do**
    $\mathcal{X}_i \leftarrow \bar{x} + \left(\sqrt{(L+\lambda)}P_i\right)$ (3.36)
for $i \leftarrow L+1\,to\,2L$ **do**
    $\mathcal{X}_i \leftarrow \bar{x} - \left(\sqrt{(L+\lambda)}P_i\right)$ (3.37)
$return\,\mathcal{X}$

Besides the already defined parameters, the process noise matrix Q and measurements noise matrix R must be set. These can be seen in Equation 3.38 and 3.39 respectively. Both have been set as diagonal matrices with noise in position x, y and z, noise in velocity in x and y, noise in euler angles in roll, pitch and yaw, noise in euler angle rates in roll, pitch and yaw and lastly noise in altimeter error starting from $(0, 0)$ to $(12, 12)$ for both matrix Q and R. However, with an additional measurement noise parameter in R for altimeter height in $(13, 13)$. This results in twelve states in the system to track.

$$Q = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.0 \end{bmatrix} \tag{3.38}$$

$$R = \begin{bmatrix} 0.011 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.011 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.011 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.300 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.300 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.005 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.005 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.005 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.030 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.030 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.030 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 \end{bmatrix} \tag{3.39}$$

Because the measurement updates in regard to the pose come from the estimated ArUco marker pose, the noise in position and orientation have been decided based on the worst STD in Section 4.1.3 where the error in the estimated pose from ArUco marker boards were based on a comparison to that of the ground truth.

Pseudocode: UKF iterate function

$x \leftarrow State$
$dt \leftarrow Time\ since\ last\ update$
$x_0^{new} \leftarrow x_0 + 0.6 \cdot x_3 \cdot dt \quad (Position\ in\ x)$
$x_1^{new} \leftarrow x_1 + 0.6 \cdot x_4 \cdot dt \quad (Position\ in\ y)$
$x_2^{new} \leftarrow x_2 \quad (Position\ in\ z)$
$x_3^{new} \leftarrow x_3 \quad (Velocity\ in\ x)$
$x_4^{new} \leftarrow x_4 \quad (Velocity\ in\ y)$
$x_5^{new} \leftarrow x_5 + x_8 \cdot dt \quad (Angle\ in\ Roll)$
$x_6^{new} \leftarrow x_6 + x_9 \cdot dt \quad (Angle\ in\ Pitch)$
$x_7^{new} \leftarrow x_7 + x_{10} \cdot dt \quad (Angle\ in\ Yaw)$
$x_8^{new} \leftarrow x_8 \quad (Velocity\ in\ Roll)$
$x_9^{new} \leftarrow x_9 \quad (Velocity\ in\ Pitch)$
$x_{10}^{new} \leftarrow x_{10} \quad (Velocity\ in\ Yaw)$
$x_{11}^{new} \leftarrow x_{11} \quad (Error\ in\ altimeter)$
$return\ x_{new}$

The worst STD where found to be 0.011 meters and 0.005 radians for the position and orientation respectively. Due to the fact that the sensor noise from the accelerometer and gyro are quite noisy, these parameters have been chosen to be higher than that of the pose to put more trust on ArUco marker updates. The STD has therefore been set to 0.3 $\frac{m}{s}$ and 0.03 $\frac{r}{s}$

for the velocities in x and y and angular rates from the accelerometer and gyro respectively. Lastly, the noise for height estimates and associated errors between the height estimate and ArUco marker position in z have been set to 0.5 meters because updates from the barometer tends to be quite noisy as well. The parameters for the process noise where found appropriate by trial and error.

Pseudocode: UKF prediction update

Prediction step for sigma points
for $i \leftarrow 0\,to\,2L$ **do**
    $\mathcal{X}_i \leftarrow irerateFunction(\mathcal{X}_i, dt)$ (3.40)
Prediction step for state
for $i \leftarrow 0\,to\,L$ **do**
    for $j \leftarrow 0\,to\,2L$ **do**
        $x_i \leftarrow x_i + W_j^m \cdot \mathcal{X}_{ij}$ (3.41)
Prediction step for covariance matrix
for $i \leftarrow 0\,to\,2L$ **do**
    $P \leftarrow P + W_i^c \cdot (\mathcal{X}_i - x_i)(\mathcal{X}_i - x_i)^T$ (3.42)
$P \leftarrow P + Q \cdot dt$

Pseudocode: UKF measurement update

$\mathcal{Y} \leftarrow Sigma\,points\,for\,measurements$
$\mathbf{y} \leftarrow Measurements$
$\mathbf{R} \leftarrow Measurement\,noise$
Covariance of measurements
for $i \leftarrow 0\,to\,2L$ **do**
    $P_{yy} \leftarrow P_{yy} + W_i^c \cdot (\mathcal{Y}_i - y_i)(\mathcal{Y}_i - y_i)^T$ (3.43)
$P_{yy} \leftarrow P_{yy} + R$
Covariance of measurement with states
for $i \leftarrow 0\,to\,2L$ **do**
    $P_{xy} \leftarrow P_{xy} + W_i^c \cdot (\mathcal{X}_i - x_i)(\mathcal{Y}_i - y_i)^T$ (3.44)
Calculate Kalman gain
$\mathcal{K} \leftarrow P_{xy} \cdot P_{yy}^{-1}$ (3.45)
Update state
$x \leftarrow x + \mathcal{K}(\mathbf{y} - \bar{y})$ (3.46)
Update covariance
$P \leftarrow P - \mathcal{K} P_{yy} \mathcal{K}^T$ (3.47)
Update sigma points
$\mathcal{X} \leftarrow Calculate\,sigma\,points$

In Pseudocode *UKF initialization*, one may notice the initialization of the $iterateFunction(x, dt)$. This function defines the update of the state in the prediction step of the UKF algorithm. This function can be seen in Pseudocode *UKF iterate function*. Here it may be noticed that the values from the accelerometer x_3 and x_4 are directly passed to the updates in position for x and y with only a single integration. To be completely mathematical correct, these values should have to be put through a double integration from where one goes from acceleration to velocity and velocity to position. The reason for doing this is because of the accumulation in error when integrating twice. To mitigate this issue, only a single integration is used with multiplication of a constant with a value of 0.6 which is found appropriate. Hence, the values from the accelerometer is interpreted as a velocity which reduces error and let go of the need for implementation of *movement end checking* [5, p. 6]. The need for this implementation would have been likely because the area under the integration

curve due rarely align when the acceleration *flips* the opposite way until reaches zero. This may create an accumulated velocity even thought the vehicle is not moving. This approach is only possible because the implementation of sensor fusion is to be used in indoor environments where the UAV is not exposed to wind. Otherwise the integration of the values from the accelerometer when the UAV having an angle in pitch or roll due to wind, would lead to errors in the position estimation.

In regard to the updates of the altitude of the UAV when no new ArUco marker pose is available, a barometer is used. Because barometers uses atmospheric pressure to calculate the altitude, errors in height can occur when pressure or temperature changes. To take into account these scenarios, the error between the altitude estimated from the ArUco marker pose and that of the barometer is calculated every time new pose estimates from ArUco markers are available. This estimated error is then subtracted from the height estimate from the barometer when the barometer is used to update the altitude of the UAV.

Pseudocode code for the prediction and measurement update can be seen in *UKF prediction update* and *UKF measurement update*. In the prediction step, the sigma points and state will be updated along with covariance. In the measurement update, the state to be updated is used, where the Kalman gain is calculated based on covariance of the measurement and measurement with states. This Kalman gain is then used to update the state and covariance of the system.

In regard to the sensor noise in the accelerometer and gyro, they are found by taking the mean of the first 10 seconds of measurements when the UAV is initiated and then subtracting this value from new measurement updates. These measurements will then have to be aligned with the ArUco marker pose estimation, which is done by multiplying the data from the accelerometer and gyro by a rotation matrix from the current IMU sensor placement on the UAV. Moreover, the barometer error will be estimated as soon as new ArUco pose estimates are available to the system.

3.5 Companion computer

This section deals with the setting up of the onboard computer used on the UAV. A number of different companion computers were considered like the Jetson nano, Banana Pi M3 and Raspberry Pi 4. The Jetson nano offers many features for computer vision algorithms and do have a much better graphic card than the Raspberry Pi 4. This is because this computer includes a Nvidia graphic card specifically build for deep learning and other computationally heavy algorithms. The Banana Pi M3 is generally more powerful than the Raspberry Pi 4. However, both the Jetson nano and Banana Pi M3 are more expensive. Hence, because the Raspberry Pi 4 yields good computational performance at an acceptable price, this unit will be used as the companion computer on the UAV.

The operating system used on the Raspberry Pi will be discussed in Section 3.5.1. Section 3.5.2 will explain the procedures for wireless communication of the Raspberry Pi 4 to the laptop. Section 3.5.3 will explain the disabling of UART between the Raspberry Pi 4 and PX4 at boot time, which can be essential for the system to boot. For performance optimizations on the Raspberry Pi 4, overclocking is used which is shortly discussed in Section 3.5.4. In Section 3.5.5 and 3.5.6 the procedures for serial communication between the Raspberry Pi to PX4 and OptiTrack system for ground truth estimation of the UAV will be discussed respectively.

3.5.1 Operating system

For onboard computing a Raspberry Pi 4 is used with an Ubuntu 18.04.5 LTS (Bionic Beaver) 64-bit server operating system. The 64-bit version is used because the Raspberry Pi 4 is build on a 64-bit architecture and the server edition is chosen to minimize overhead because the GUI is not needed as the interface used for communicating to the Raspberry Pi will be through ssh from the terminal. The mentioned operating system can be fetched from the official Ubuntu website [30]. The Raspberry Pi 4 gives a number of possibles options in regard to system boot, but in this case an SD card will be used. Now etcher is used to flash the Ubuntu 18.04.5 image to the SD card which offers a secure way for image flashing.

The reason for choosing this operating system compared to the default use of Raspbian, which is made specifically for Raspberry Pi, is that the installation of software can be done in the same manner as already described in Section 3.2.2.5. If one uses Raspbian, some libraries or software have to be installed from source which could be time consuming. Using this

setup of Ubuntu 18.04, the exact same procedures can be implemented on the Raspberry Pi as onboard computer which is beneficial.

3.5.2 Wireless communication

To enable ssh for a wired communication between laptop and Raspberry Pi, the following lines of code from the terminal are executed as seen in Listing 8 to the SD card.

```
#Change directory to system-boot
$ cd /media/<user>/system-boot/
#Make a file called ssh
$ touch ssh
```

Listing 8: How to enable ssh communication by making a file called ssh to system-boot of the sd card

To access the Raspberry Pi from the terminal, the correct IP-address of the Raspberry Pi must be known. This can be done using the protocol nmap which searches for all possible IP-addresses in a defined sub-network in the local network. This is done using Listing 9. When the correct IP-address is found an ssh connection can be established.

```
#Find the IP-address of eth0 (Raspberry Pi)
$ ifconfig
#Find hosts in specific address range
$ nmap -sn <ip-address-eth0>/24
#SSH into Raspberry Pi
$ ssh ubuntu$<ip-address-raspberry-pi>
```

Listing 9: Find the IP-address of the Raspberry Pi for ssh communication

Because the goal is to have a wireless connection to the Raspberry Pi from an access point using WiFi, the netplan has to be configured from /etc in the system configuration files of the Raspberry Pi. The file to be updated can be seen in Listing 10. Here the SSID (name of WiFi network) must be configured along with the password of the WiFi network.

```
#File /etc/netplan/50-cloud-init.yaml
network:
    ethernets:
        eth0:
            dhcp4: true
            optional: true
    version: 2
    wifis:
        wlan0:
            optional: true
            access-points:
                "<YourSSID>":
                    password: "<YourPassword>"
            dhcp4: true
```

Listing 10: File to configure to enable wireless connection

Using vim or another text editor, the mentioned file can be updated using Listing 11 from the established wired ssh connection.

```
#Configure access point changing the wlan0 settings
$ sudo vim /ect/netplan/50-cloud-init.yaml
#Apply settings
$ sudo netplan apply
```

Listing 11: Create an access point from a wireless connection by configuring netplan

Now the outgoing IP-address through WiFi from the Raspberry Pi can be found. This IP-address can now be used to ssh into Raspberry Pi if both the Laptop and Raspberry Pi are connected to the same WiFi network.

3.5.3 Disabling UART at boot time

Because the system boot of the Raspberry Pi can be interrupted if transmissions are observed on the TX and RX pins on the Raspberry Pi, UART communication must be disabled during auto boot. These transmissions occur because the fan shim connected to the Raspberry Pi, used to cool down the CPU, may send data to the system at boot up. If this happens, the system will not boot which would be critical.

To mitigate this issue, the following lines in Listing 12 can be added to the /system_boot/config.txt on the Raspberry Pi which is based on a discussion on stack exchange for disabling UART at boot time. Another method is to disable UART communication from the TX and RX pins completely by setting enable_uart=0 in the same file.

```
[all]
kernel=vmlinuz
initramfs initrd.img followkernel
#device_tree_address=0x03000000
```

Listing 12: How to disable UART at boot time

3.5.4 Performance optimizations

The Raspberry Pi is based on a Quadcore CPU which utilizes four cores with 1.5 GHz each. However, these settings can be altered by adding the lines in Listing 13 to the file /system_boot/config.txt on the Raspberry Pi.

```
[pi4]
over_voltage=2
arm_freq=1750
```

Listing 13: Increase clock speed on the Raspberry Pi by overclocking

However, care must be taken because an increased number of cycles on the CPU increases the heat generated. The maximum overclocking is approximately 2.1 GHz, but a 1.8 GHz overclocking will be used in this case which is found appropriate for the cooling system available (fan shim). The reason for boosting the performance of the Raspberry Pi is to enable more calculations per time unit, which results in more pose updates to the system.

3.5.5 Serial communication to PX4

As mentioned in Section 2.1, serial connection between the Raspberry Pi and PX4 is achieved by connecting the 6 pin connection cable to a TTL 232R cable to the USB port of the Raspberry Pi. However, for communication to be initiated some additional steps have to be performed. In these steps QGroundCrontrol is used. Here one can change parameters in the configuration files of the PX4. The parameters *MAV_1_CONFIG* is set to *TELEM/SERIAL 4* to enable UART through

the USB port, *MAV_1_MODE* is set to *onboard* for telling PX4 that this Mavlink connection is used to comminicate to the onboard computer (Raspberry Pi) and *MAV_1_RATE* and *SER_TEL4_BAUD* is set to *80000* and *921600 8N1* for the date rates respectively.

3.5.6 Communication with OptiTrack for ground truth of UAV

As discussed in Section 2.1, an extra WiFi adapter is connected to the Raspberry Pi. This is done so that the UAV can be controlled from a wireless connection between the laptop and Raspberry Pi meanwhile the Raspberry Pi gets ground truth updates of its pose using the other WiFi module connected to the OptiTrack system. The OptiTrack system is discussed in Section 3.6.

To get the ground truth from the OptiTrack system instead of the Gazebo simulation, the ROS package *vrpn_client_ros* is used which enables communication between the onboard computer and the OptiTrack system using WiFi. This can be accomplished by running the command in Listing 14 after setting up the Optitrack system.

```
roslaunch vrpn_client_ros sample.launch server:=<server-ip-adress>
```

Listing 14: How to get pose estimation of the UAV from the OptiTrack system

This way the exact same code can be used as in the simulation, but instead of subscribing to the ground truth pose of the UAV from Gazebo, the pose estimated from the OptiTrack system can be used from the topic published using Listing 14.

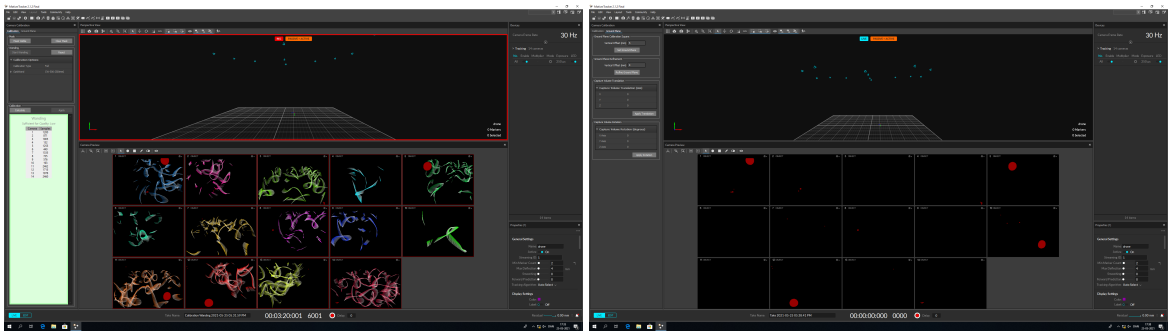
3.6 OptiTrack system

For tracking the UAV in real flight, the OptiTrack system is used which enables high precision tracking of the pose of the UAV. This is done using fourteen cameras placed on a truss which are attached to the ceiling as seen in Figure 28. These cameras transmits pulsed infrared light using attached infrared LEDs. The light will be reflected back to the cameras from small markers attached to the UAV as seen in Figure 30a. This way the pose of the UAV can be tracked using triangulation.

For setting up the OptiTrack system a getting started guide from the OptiTrack homepage can be used. However, in this case three changes to the default setup have been used. Here the exposure time has been set to 250 μs and the camera frame rate to 30Hz. The exposure time has been set to a lower value because the markers located on top of the UAV are quite small which results in that the exposure time must be set lower in order to properly track the UAV.

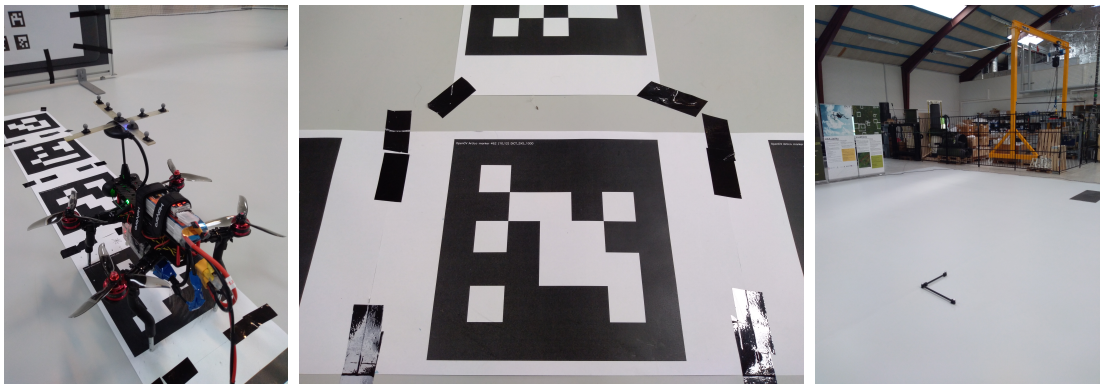
(a) *(b)* *(c)*

Figure 28: Illustration of the setup of the vision based navigation area inside of the OptiTrack system. Here a simplified version of the original design used in the simulation from Section 3.1.1 has been used. This is because only parts of the planned tests have been conducted due to the limited access to the airport discussed in section 1.4. Small changes to the landing boards had been made to fit the size of the A0 paper. Now the distance between markers and marker length are 6.5 cm and 13 cm, 3.1 cm and 6.1 cm, 2.85 cm and 5.858 cm for landing board board one, two and three respectively



(a) (b)

Figure 29: Illustration of the setup of the OptiTrack system using motive. The fourteen cameras placed at the truss are calibrated in Figure 29a and the calibration square in Figure 30c is used to set the ground plane in Figure 29b



(a) (b) (c)

Figure 30: Small reflective markers have been placed on top of the GPS module on the UAV which is used to track its pose as seen in Figure 30a. In Figure 30b, a close view of one of the ground markers can be seen. It may be noticed that the row and column number of the marker is labeled in the top left corner which is used to easily keep track of each marker that is part of the 25×25 ArUco marker board. The calibration square can be seen in Figure 30c with the used orientation and origin

The camera frame rate has been set to 30 which matches the update rate from Gazebo. These changes can be seen in Figure 29a and 29b. Lastly, the ground plane has been set as illustrated in Figure 30c. Here the longer leg represents the z axis and shorter leg the x axis which means the y axis will point upwards. In this setup the x axis points towards the ArUco marker landing boards where motive will automatically change the ground plane so that the z axis will point up. This way, the coordinate system matches exactly the coordinate system from the Gazebo simulation which means that no changes in the code have to be made in regard to the ground truth.

The reason for placing the markers on top of the GPS module is that cameras could have a hard time tracking the UAV if these markers were placed other places on the UAV. Using this asymmetrical design of the marker placements, the UAV can be tracked without problems using the OptiTrack system. In Figure 30b, a close view of one of the ground markers can be seen. The ArUCo marker have been printed in the middle of the A4 paper. This way the papers can be aligned side-by-side to achieve the best accuracy. However, small changes to the size have been made. The distance between the markers are now set to 10.5 cm and the size of the marker is 19.2 cm. These where found appropriate for the giving size of the paper.

Results of pose estimation using ArUco markers can be seen in Section 4.2.1, UAV flight in Section 4.2.2 and vision based landings in Section 4.2.3. These are only some of the planned tests of real flight of the UAV where a thorough description of the others will be explained in Section 7 for future work.

4 Results

This section contains the data of the results from simulations using Gazebo in Section 4.1 and real flight in the airport using the OptiTrack system as ground truth in Section 4.2. The results will be discussed and analyzed in Section 5.

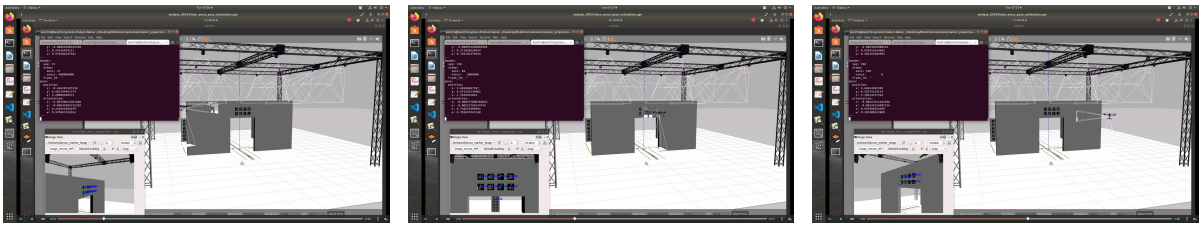
4.1 Simulation

This section includes simulations of the system in Gazebo for a number of different configurations. This will give an indication of the performance of the implementation which is critical before moving on to implement the solution on the real UAV.

Section 4.1.1 will investigate the performance of the ArUco pose estimation algorithm based on the GPS2Vision marker board on the wall to be compared to the ground truth pose of the UAV. This will yield an error which is expected to increase as the UAV moves further away from the ArUco board. Thus, the maximum allowed distance away from the board can be estimated to insure a reliable GPS to vision transition. To further analyse the reliability of the ArUco pose estimation based on the GPS2Vision board, the rolling average of a sequence of pose estimates is analyzed for fluctuations in the data in Section 4.1.2. Section 4.1.3 will deal with performance evaluation of the control algorithms for pose control by having the UAV to hold its current pose in front of the GPS2Vision and landing boards. This will give an indication if further optimizations of the pose have to be performed. Moreover, the ArUco pose estimations will be compared to that of the ground truth to see if the error in the pose estimation is reduced when the number of visible ArUco markers are increased by using different setups of ArUco boards. In Section 4.1.4, the actual GPS2Vision implementation will be analyzed to see how the UAV performs in locating the ArUco board, navigate to the board and make the GPS to vision transition in windy conditions for stress analysis. Vision based navigation will be analyzed in Section 4.1.5 using a number of different ArUco marker boards located on the ground for pose estimation. To stress test the system, boards with missing markers will be used, which means that the UAV has to rely on sensor fusion from IMU and barometer data when no ArUco markers are visible in the image. Further performance evaluations will be based on increased horizontal speed of the UAV. Finally, the vision based landings will be analyzed in Section 4.1.6. Here all landing stations will be used to compare the precision and accuracy of the landings to evaluate if the number of ArUco markers in the image has an impact on precision and accuracy for both the landing procedure and the actual pose estimation.

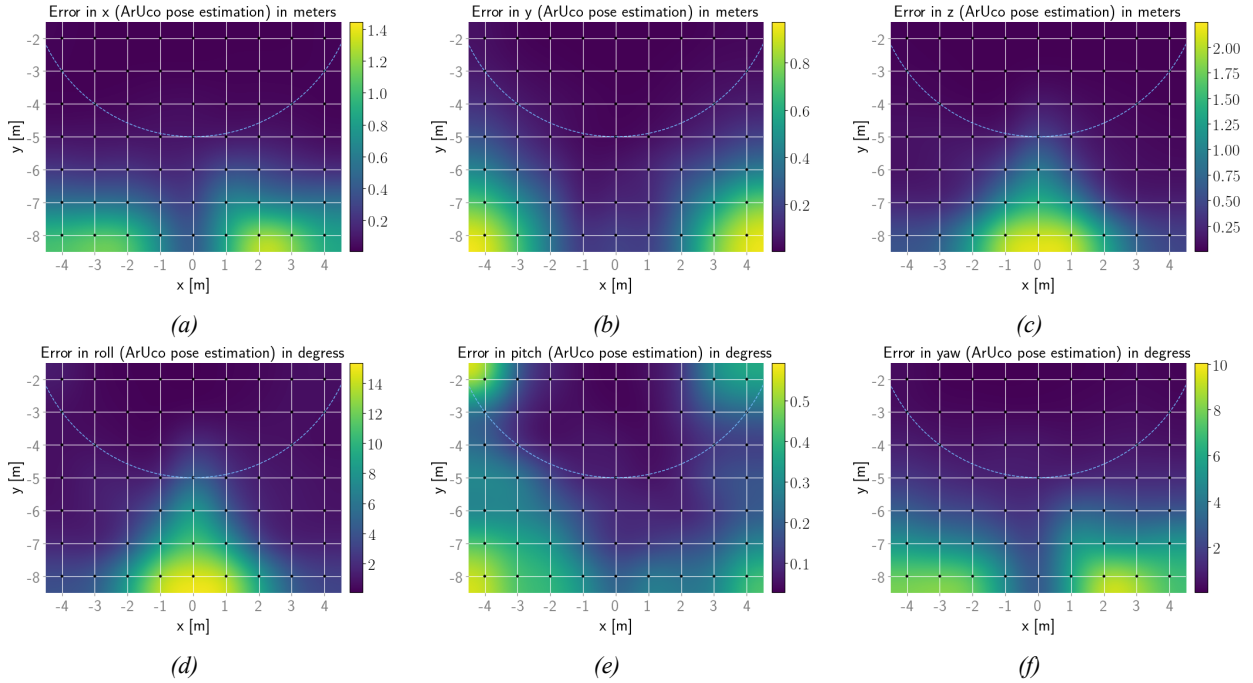
4.1.1 GPS to vision ArUco pose estimation

The GPS2Vision pose estimation test is based on a predefined number of waypoints which the UAV has to visit while keeping its orientation towards the GPS2Vision marker. The altitude will be 2.5 meters above the ground which is approximately the same as the origin of the GPS2Vision board. At each waypoint, the UAV estimates the pose of the board for five seconds which will be compared to the ground truth of the UAV and the mean error computed. Cubic interpolation will then be used to smooth out the errors along a 2D top view of the area in front of the GPS2Vision board as seen in Figure 32. Illustrations of the procedures can be seen in Figure 31.



(a) *(b)* *(c)*

Figure 31: Illustrations of the GPS2Vision pose estimation test in three different positions with the UAV facing the GPS2Vision board. A video of the first couple of iterations can be seen on GitHub using GPS2Vision pose estimation



(a) *(b)* *(c)* *(d)* *(e)* *(f)*

Figure 32: Black dots indicates where the UAV has estimated the pose of the ArUco board and the circle colored sky blue indicates a safe GPS2Vision transition area. It may the noticed that the error in the pose becomes quite high when the UAV has a distance of 7-8 meters away from the GPS2Vision board

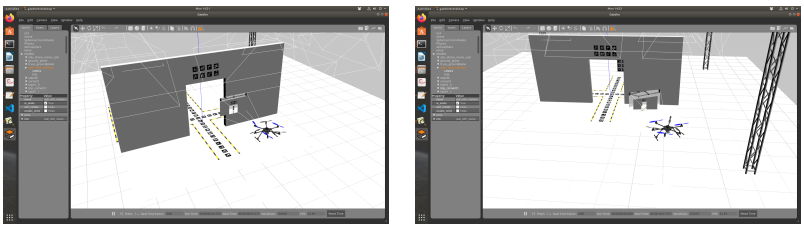
In order to replicate the test, the command in Listing 15 can be run from the terminal.

```
#For execution of the test
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world
  ↪ drone_control_args:="GPS2Vision_aruco_pose_estimation_test" x:=-3.0 y:=0.0
```

Listing 15: Command to be used to replicate the test

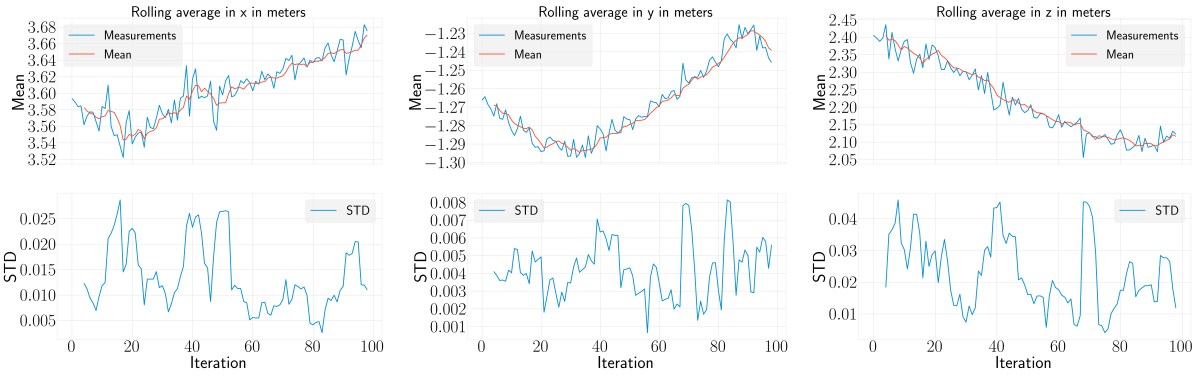
4.1.2 Rolling average from ArUco pose estimation

To analyze the possibility of fluctuations in the ArUco pose estimates, the rolling average is used for calculating the mean and STD in two different positions for one-hundred iterations. This can be seen in Figure 33. This is done as an extension to the maximum allowed distance from the GPS2Vision marker board from Section 4.1.1 to ensure that the GPS to vision transition is only performed when the pose estimates are stable and reliable.



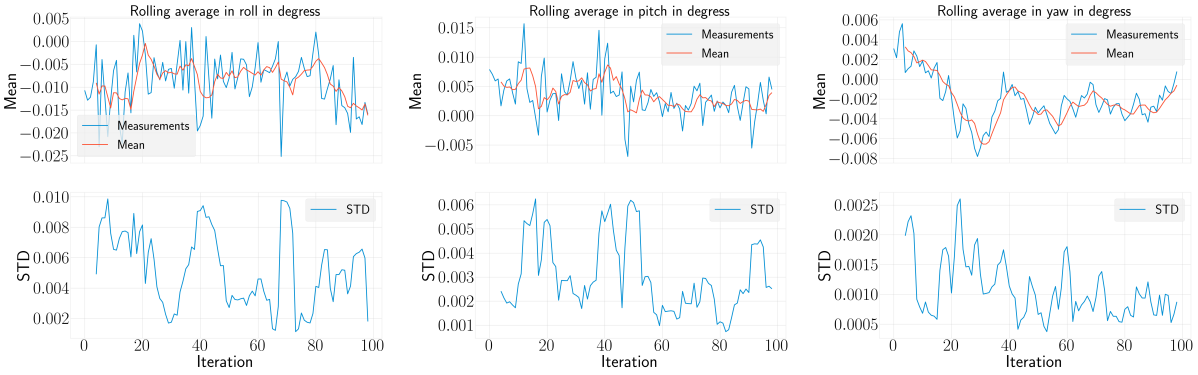
(a) *(b)*

Figure 33: Illustrations of the two positions used to evaluate the rolling average for one-hundred iterations. The positions of the UAV in Figures 33a and 33b are (0,-5) and (3,-7) respectively from the coordinate system in Figures 32. However, the poses in Figure 34, 35, 36 and 37 are in regard to the ArUco marker board located on the ground



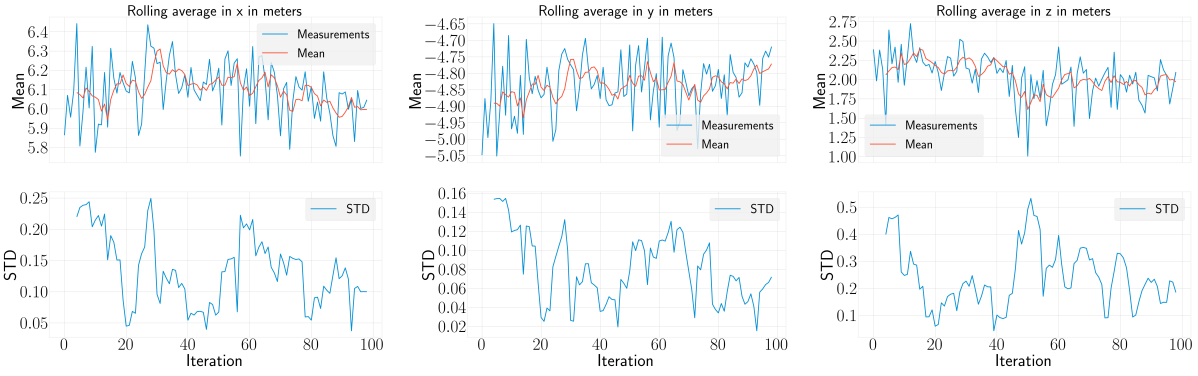
(a) *(b)* *(c)*

Figure 34: Illustrations of the rolling average from the configuration in Figure 33a. As it can be seen, the fluctuations of the position estimates are quite steady with only minor fluctuations of a few centimeters



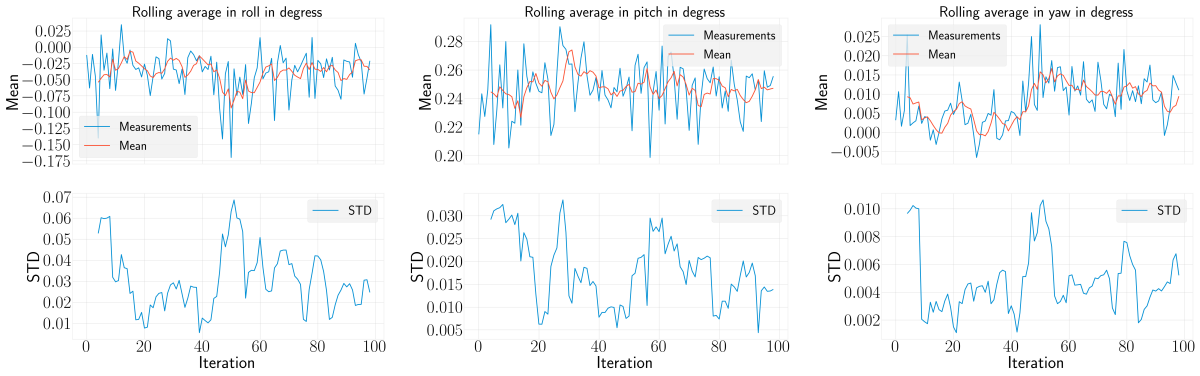
(a) *(b)* *(c)*

Figure 35: Illustrations of the rolling average from the configuration in Figure 33a. As it can be seen, the fluctuations in the angle estimates are steady with only negligible deviations



(a) *(b)* *(c)*

Figure 36: Illustrations of the rolling average from the configuration in Figure 33b. As it can be seen, the fluctuations of the position estimates fluctuates a lot especially in the altitude

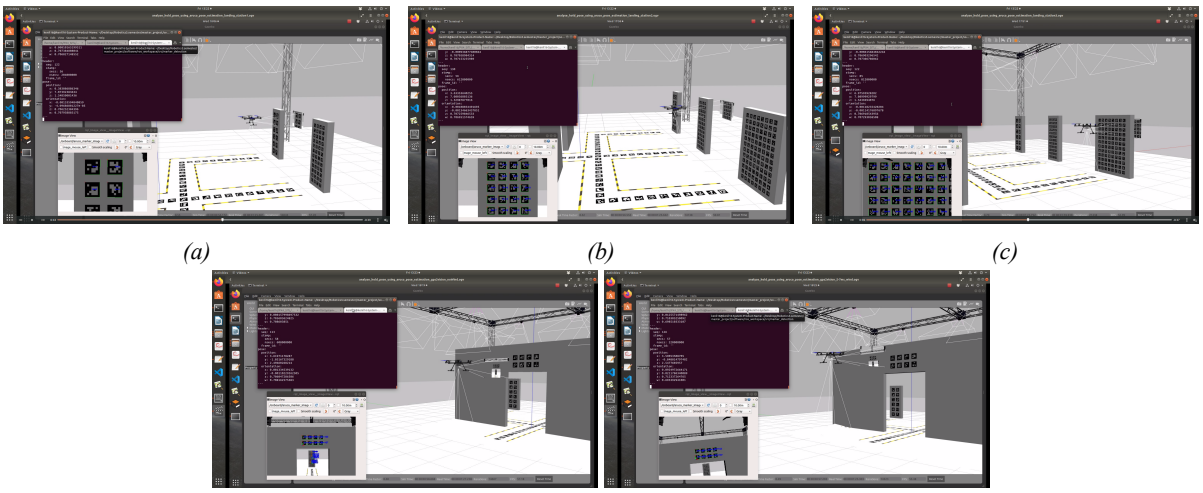


(a) *(b)* *(c)*

Figure 37: Illustrations of the rolling average from the configuration in Figure 33b. As it can be seen, the fluctuations in the angle are steady even in this configuration

4.1.3 Hold pose using ArUco pose estimation

To analyses how the ArUco pose estimations effects the algorithms for pose control, the UAV was set to hold its pose for 30 seconds in a number of different configurations as seen in Figure 38.



(a) *(b)* *(c)*

(d) *(e)*

Figure 38: Illustrations of the different configurations for the tests. The ArUco landing boards are used in Figures 38a, 38b and 38c for landing stations one, two and three respectively. The GPS2Vision board is used in 38d and 38e without and with simulated wind. Videos of the tests can be seen on Github using Landing station one, Landing station two, Landing station three, GPS2Vision board and GPS2Vision board with 5-7 $\frac{m}{s}$ wind

In Figures 38a, 38b and 38c, the ArUco landing boards are used where the UAV are set to keep its pose one meter in front of the board with an altitude of 1.5 meters. This is done to ensure that most of the markers in the board can be seen in the image without been too close to the board to cause damage to the UAV. In Figures 38d and 38e, the GPS2Vision board is used without and with simulated wind. Here the UAV is set to keep an altitude of 2.5 meters approximately 4 meters in front of the board.

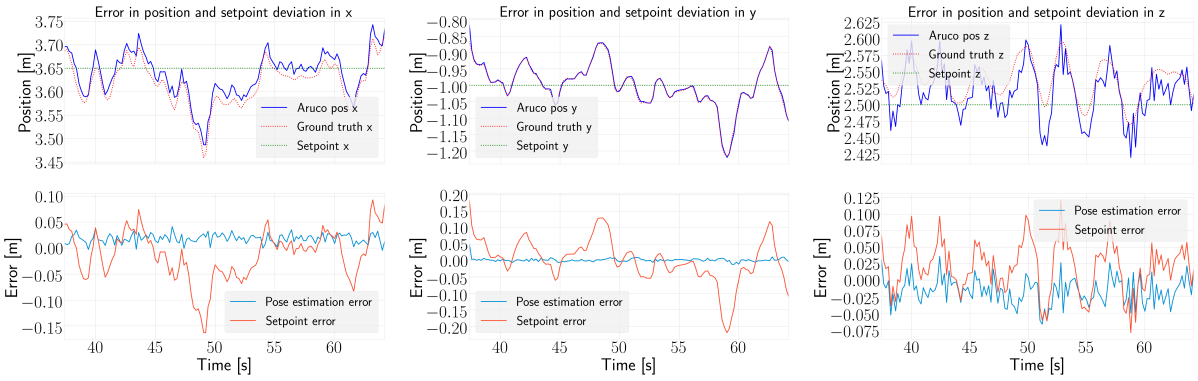
This distance in front of the GPS2Vision board is chosen on the basis of the results in Section 4.1.1 and 4.1.2 which was found to be a safe transition area when going from using GPS to vision based navigation.

Table 7: Statistics of the results of holding the pose using the GPS2Vision boards. It may be noticed that adding wind to the simulation, hence giving the UAV an angle in roll and pitch, does not seem no effect the pose estimation error significantly. Only the target setpoints are effected

Estimation error	Runs	Wind	Mean position	STD position	Mean angle	STD angle
			Test 1			
ArUco pose GPS2Vision board	10	No	$1.43cm$	$1.10cm$	$0.18°$	$0.13°$
Setpoint	10	No	$3.12cm$	$2.58cm$	$0.28°$	$0.22°$
			Test 2			
ArUco pose GPS2Vision board	10	5-7 $\frac{m}{s}$	$1.47cm$	$0.95cm$	$0.49°$	$0.31°$
Setpoint	10	5-7 $\frac{m}{s}$	$4.04cm$	$3.35cm$	$4.45°$	$1.49°$

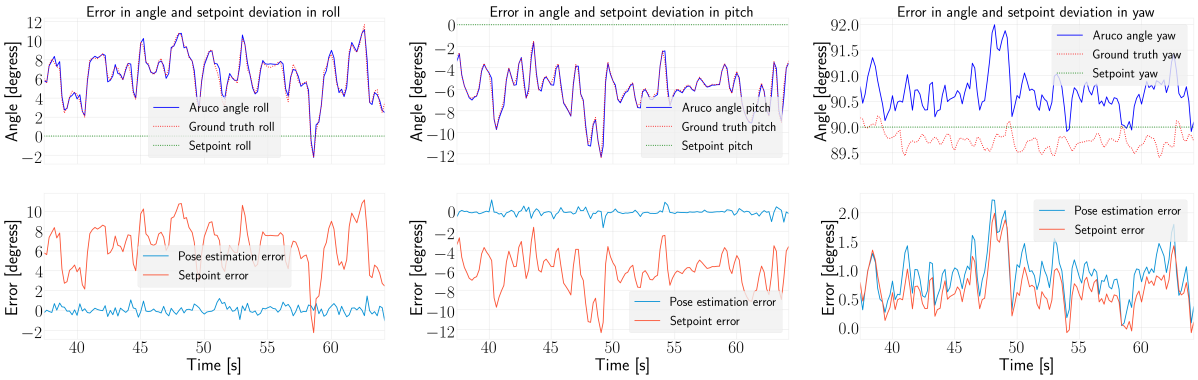
Table 8: Statistics of the results of holding the pose using the ArUco landing boards. It may be noticed that as more ArUco markers are present in the image, the lesser is the pose estimation error. However, no significant decrease in the pose estimation error is seen between using landing board one and three even though the entire image is filled with ArUco markers as seen in Figure 38c

Estimation error	Runs	Wind	Mean position	STD position	Mean angle	STD angle
			Test 3			
ArUco pose landing board one	10	No	$0.12cm$	$0.23cm$	$0.03°$	$0.05°$
Setpoint	10	No	$2.93cm$	$2.90cm$	$0.12°$	$0.12°$
			Test 4			
ArUco pose landing board two	10	No	$0.11cm$	$0.21cm$	$0.02°$	$0.01°$
Setpoint	10	No	$2.92cm$	$2.89cm$	$0.11°$	$0.09°$
			Test 5			
ArUco pose landing board three	10	No	$0.09cm$	$0.20cm$	$0.02°$	$0.01°$
Setpoint	10	No	$2.89cm$	$2.85cm$	$0.11°$	$0.09°$



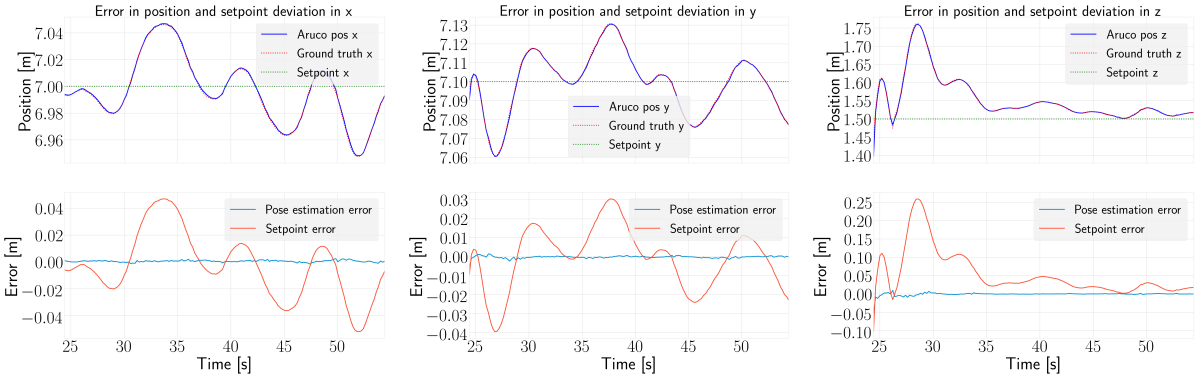
(a) *(b)* *(c)*

Figure 39: Illustration of the pose estimation and setpoint error using configuration in Figure 38e. It may be noticed that the pose estimation errors in x and y are in the orders of centimeters with an increased error in z



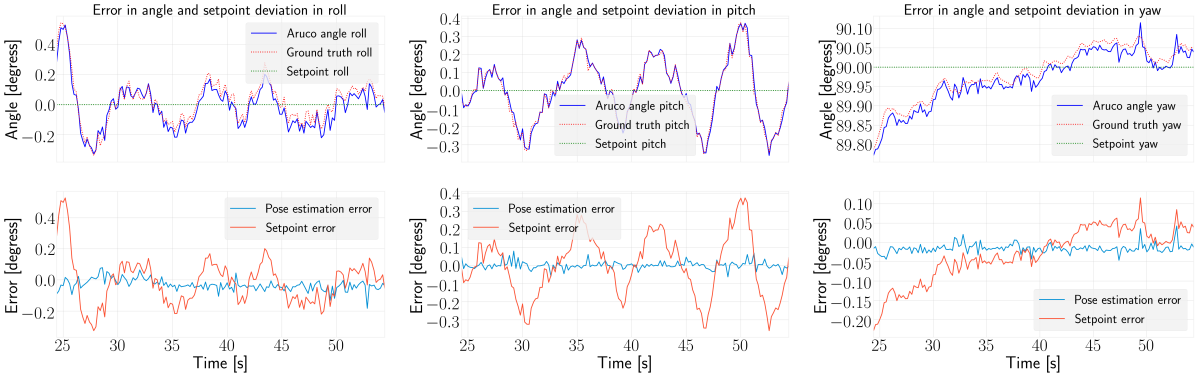
(a) *(b)* *(c)*

Figure 40: Illustration of the pose estimation and setpoint error using configuration in Figure 38e. It may be noticed that the UAV has trouble keeping roll and pitch close to zero due to the wind in the simulation in Figures 40a and 40b



(a) *(b)* *(c)*

Figure 41: Illustration of the pose estimation and setpoint error using configuration in Figure 38c. It can be seen that the error in position estimation is almost negligible down to millimeters of error which was also mentioned in Table 8. Moreover, the setpoint error has decreased significantly which illustrates that the overall stability of the system increases when the UAV operates closer to the board



(a) *(b)* *(c)*

Figure 42: Illustration of the pose estimation and setpoint error using configuration in Figure 38c. The error in the angle estimations can be seen to be very small as well as the target setpoint which again confirms that operating close to the board decreases the pose estimation error

In order to replicate the tests, the command in Listing 16 can be run from the terminal.

```
#For execution of the test
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world
  ↪ drone_control_args:="hold_aruco_pose_test" x:=-4.0 y:=0.0 headless:=false gui:=true
```

Listing 16: Command to be used to replicate the test

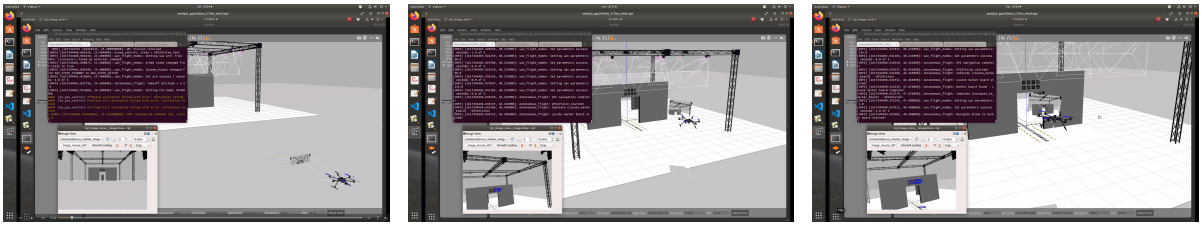
Here the position for x and y is set to (3, 3.3), (3,0), (3,-3.3), (-4,0) and (-4,0) for the test in Figure 38a, 38b, 38c, 38d and 38e respectively which is in regard to the Gazebo simulation coordinate system. The wind in the simulation can be activated by changing the followings lines in Listing 17 in the file /software/ros_workspace/PX4-software/optitrack_big_board_onepattern.world. All these values were initially set to zero to disable the use of wind.

```
<plugin name='wind_plugin' filename='libgazebo_wind_plugin.so'>
  <frameId>base_link</frameId>
  <namespace>/foo/bar</namespace>
  <xyzOffset>0 0 0</xyzOffset>
  <windDirectionMean>1 1 1</windDirectionMean>
  <windDirectionVariance>0.05</windDirectionVariance>
  <windVelocityMean>1.0</windVelocityMean>
  <windVelocityVariance>0.05</windVelocityVariance>
  <windGustDirection>1 1 1</windGustDirection>
  <windPubTopic>world_wind</windPubTopic>
</plugin>
```

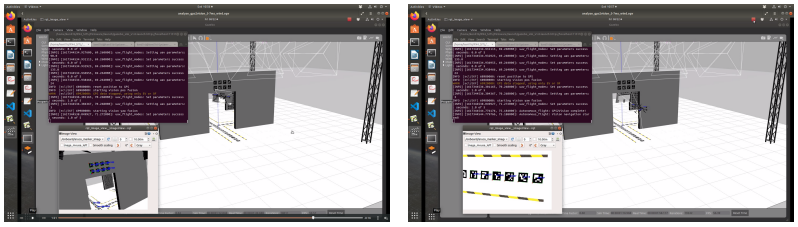
Listing 17: Activation of wind plugin for the optitrack_big_board_onepattern.world world

4.1.4 GPS to vision navigation

The most important part of the simulations was the actual transition between using GPS and vision as navigation. Because the influence of wind plays a major part of the stability of this transition, the simulated environment will include wind speeds between 5-7 $\frac{m}{s}$ and 7-10 $\frac{m}{s}$ for stress test of the system. This can be seen in Figure 43.



(a) *(b)* *(c)*

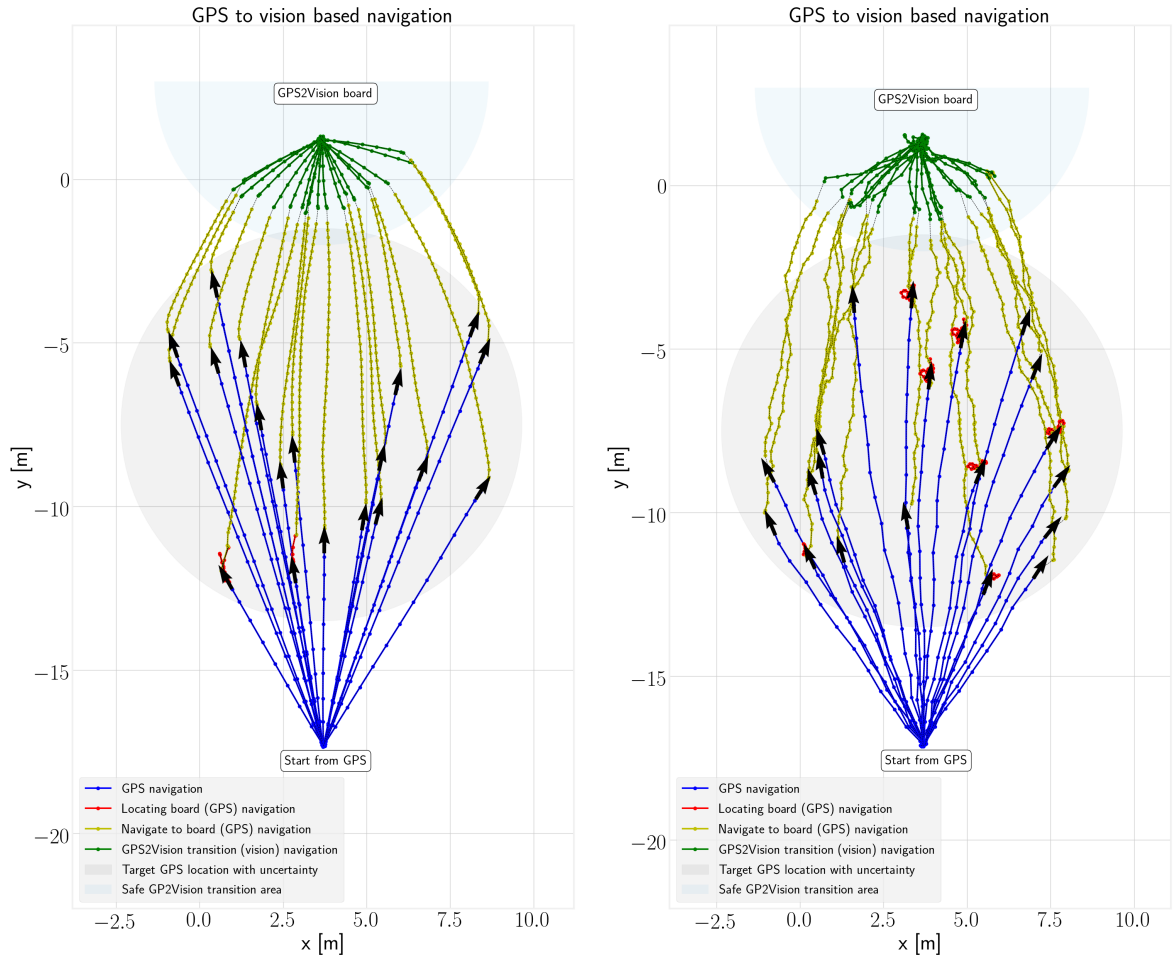


(d) *(e)*

Figure 43: Illustrations of the GPS2Vision test. Videos of the test can be seen on Github using GPS2Vision no wind, GPS2Vision 5-7 $\frac{m}{s}$ wind and GPS2Vision 7-10 $\frac{m}{s}$ wind

Table 9: Statistics of the results of the GPS2Vision tests. The GPS, locate board, navigate to board and GPS2Vision defines how much time the UAV in average spend in the different substates when executing the tests. It may be noticed that the UAV used a lot more time in GPS2Vision (gps to vision transition) in test 3. That it because the UAV had a hard time finding the ArUco markers located on the ground due to the angle in roll and pitch caused by the simulated wind. Moreover, it was only able to complete 17 out of 20 runs in test 3, because it lost sight of the ground marker in the GPS to vision transition and had to land

Wind	Runs	Completed	Vel (GPS)	GPS	Locate board	Navigate to board	Vel (vision)	GPS2Vision
				Test 1				
0 $\frac{m}{s}$	20	20	2 $\frac{m}{s}$	7.45s	0.50s	14.64s	1 $\frac{m}{s}$	5.28s
				Test 2				
5-7 $\frac{m}{s}$	20	20	2 $\frac{m}{s}$	6.66s	1.76s	16.05s	1 $\frac{m}{s}$	5.32s
				Test 3				
7-10 $\frac{m}{s}$	20	17	2 $\frac{m}{s}$	7.04s	3.71s	16.24s	1 $\frac{m}{s}$	20.39s



(a) *(b)*

Figure 44: Illustrations of test 1 in Figure 44a and test 3 in Figure 44b summarized in Table 9. The black arrows define the directions of movement of the UAV for each run along with ended GPS target location due to variations of ± 6 meters in the GPS accuracy to take into account real life conditions. The actual target GPS location is predefined to be centered in the gray circle

In Figure 43a, the UAV starts from an initial pose and moves to a predefined GPS target location. In Figure 43b, the UAV has reached the target GPS location and begins to search for the GPS2Vision board. When the UAV has found the GPS2Vision board, it begins to navigate to it still using GPS as seen in Figure 43c. When the UAV is of a radius of maximum 5 meters away from the GPS2Vision board and the estimated pose is stable, it makes a GPS2Vision transition as seen in Figure 43d. When the last setpoint is reached, the UAV now uses the bottom camera to enable the actual vision navigation which can be seen in Figure 43e.

In Figure 44, a total of forty runs were performed with no wind in Figure 44a and 7-10 $\frac{m}{s}$ simulated wind speeds in 44b. The dots between the lines defines the actual point of ground truth measurement of the position of the UAV. The blue, red and yellow lines defines the use of GPS as pose estimate for navigation while the green lines defines the use of the GPS2Vision board as pose estimate. As it can be seen, the transition of using vision instead of GPS first happens when the UAV is located inside of the safe transition area visualized with a sky blue color.

In order to replicate the tests, the command in Listing 18 can be run from the terminal.

```
#For execution of the test
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world
  ↪ drone_control_args:="GPS2Vision_test" x:=-21.0 y:=0.0 headless:=false gui:=true
```

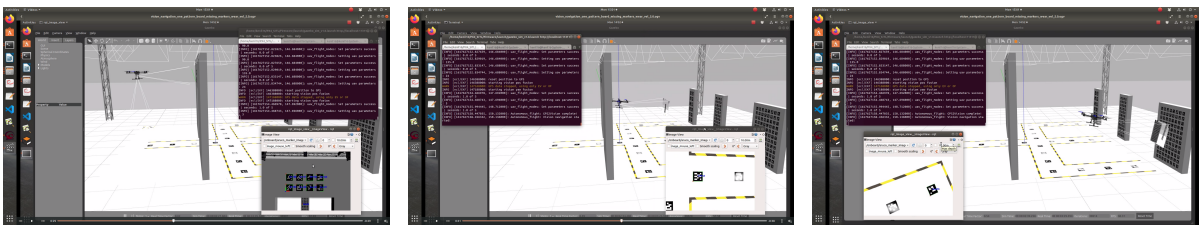
Listing 18: Command to be used to replicate the test

For activating wind, Listing 17 must be set. However, in the same piece of code the *windVelocityMean* is set to 1.0 and 1.5 for test 1 and test 2 which ensures wind speeds of approximately $5 - 7\frac{m}{s}$ and $7 - 10\frac{m}{s}$ respectively.

4.1.5 Vision based navigation

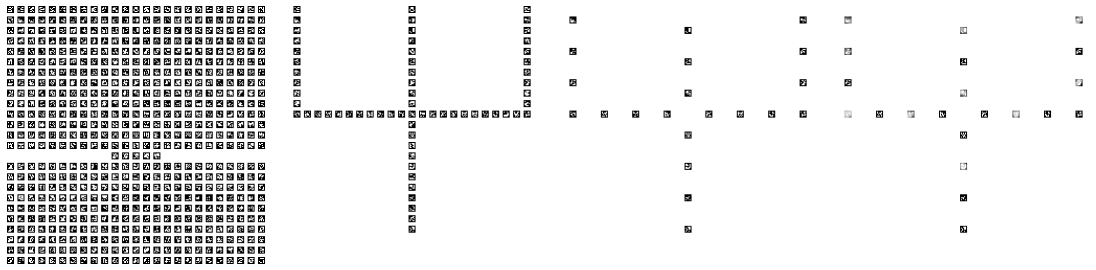
To analyze how the system performs when markers are not always visible in the image, a setup of different ArUco boards located on the ground has been proposed which can be seen in Figure 46. This is done to stress test the system and evaluate the performance using the implemented sensor fusion. The UAV is set to move from a predefined location to one of the landing stations and then back again. This can be seen in Figure 45.

The ground truth pose of the UAV will be compared to the estimates of the pose from sensor fusion which is summarized in Table 10. The waypoint error defined in Table 10, means the maximum allowed error between the target setpoint and the estimated pose before moving on to the next target location. This error is calculated using the euclidean error distance added with the error in the yaw angle.



(a) *(b)* *(c)*

Figure 45: Illustrations of the vision based navigation test. The UAV initially starts in front of the GPS2Vision board as seen in Figure 45a. Then it moves either to landing station one, two or three and then back to its initial position where the target landing station is chosen randomly at the initialization of the test. In this case, the UAV can be seen to move to landing station three in Figures 45b and 45c. Videos of the test can be seen on Github using Vision navigation (full ArUco board) and Vision navigation (ArUco board with missing markers)

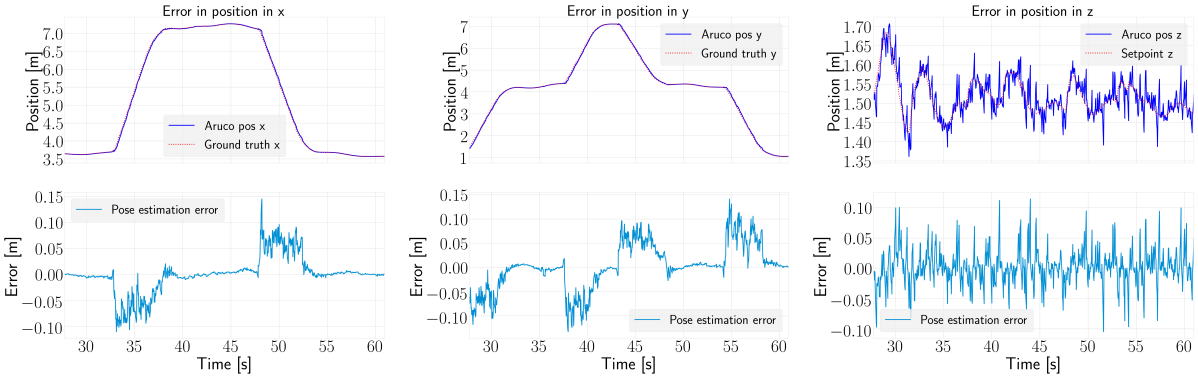


(a) *(b)* *(c)* *(d)*

Figure 46: Illustrations of the ArUco boards used which are located on the ground. In Figure 46a, a complete 25×25 ArUco board is used for test 1. A one pattern ArUco board is used in 46b for test 2. The same one pattern ArUco board is used in 46c for test 3 and 46d for test 4 and 5, but with a meter between the markers (missing markers) and with some of the markers blurred in Figure 46d to illustrate wear due to walking on the markers

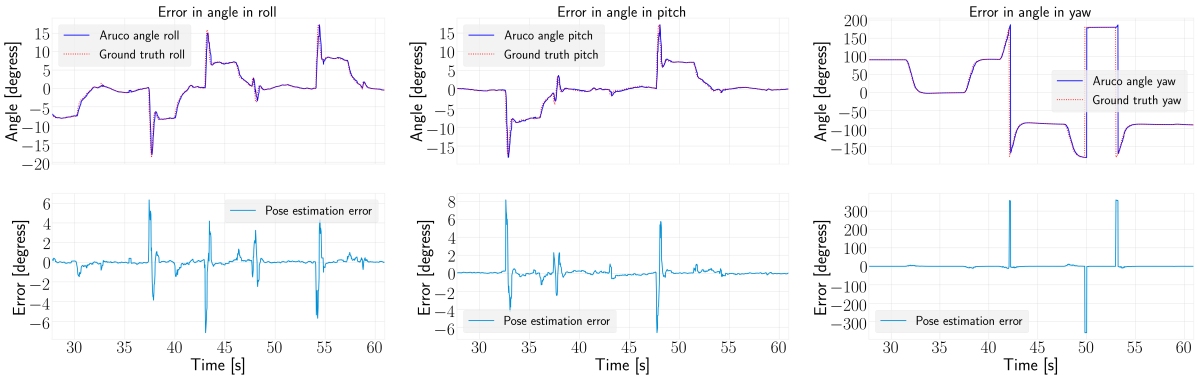
Table 10: Statistics of the results in the vision navigation tests. The mean error can be seen to be in the range of centimeters in all tests, but with an increase when the number of visible markers decreases and speed increases. The large errors are primarily caused by a time delay between the ground truth of the UAV and the estimated pose based on sensor fusion

Estimation error	Runs	Waypoint error	Vel (Horizontal)	Mean position	STD position	Mean angle	STD angle
			Test 1				
Sensor fusion	20	$0.1m$	$1\frac{m}{s}$	$2.44cm$	$2.51cm$	$1.59°$	$9.68°$
			Test 2				
Sensor fusion	20	$0.1m$	$1\frac{m}{s}$	$2.44cm$	$2.29cm$	$1.30°$	$7.44°$
			Test 3				
Sensor fusion	20	$0.1m$	$1\frac{m}{s}$	$3.19cm$	$3.28cm$	$1.99°$	$10.88°$
			Test 4				
Sensor fusion	20	$0.1m$	$1\frac{m}{s}$	$4.21cm$	$3.96cm$	$3.09°$	$11.99°$
			Test 5				
Sensor fusion	20	$0.1m$	$5\frac{m}{s}$	$5.66cm$	$5.79cm$	$6.89°$	$22.29°$



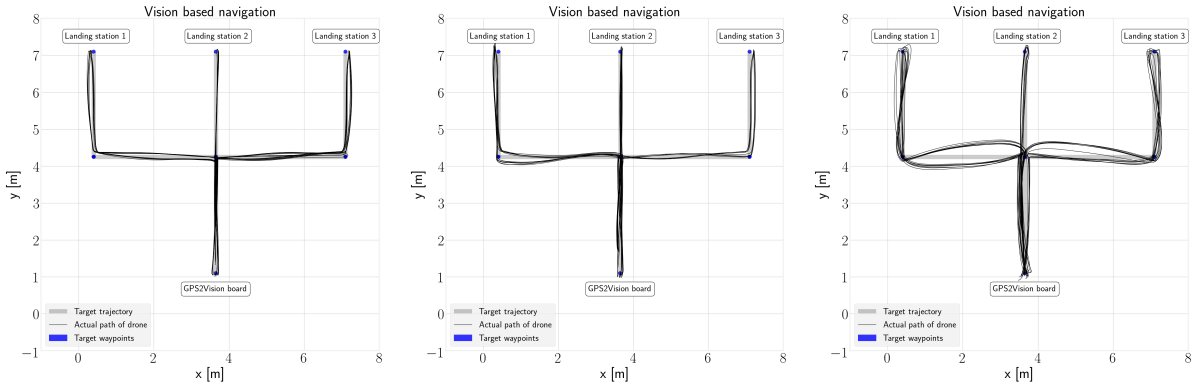
(a) *(b)* *(c)*

Figure 47: Illustration of the position estimation error using sensor fusion for test 1. It can be seen that the error increases a lot when large changes in the position occurs. This is because of a delay between the estimated position and the ground truth. This delay is even clearer in Figure 48c when zooming in on the graph



(a) *(b)* *(c)*

Figure 48: Illustration of the angle estimation error using sensor fusion for test 1. It can be seen that the error increases a lot when large changes in the angles occurs. This is because of a delay between the estimated angle and the ground truth. This delay leads to very large errors especially in yaw as seen in Figure 48c



(a) *(b)* *(c)*

Figure 49: Illustrations of the ground truth path of the UAV seen from above in Figure 49a, 49b and 49c for test 1, 4 and 5 respectively. The blue dots illustrates the target setpoints when going to either landing station one, two or three. The gray lines defines the wanted trajectories and black lines the actual path of the UAV. Each test was performed for twenty runs

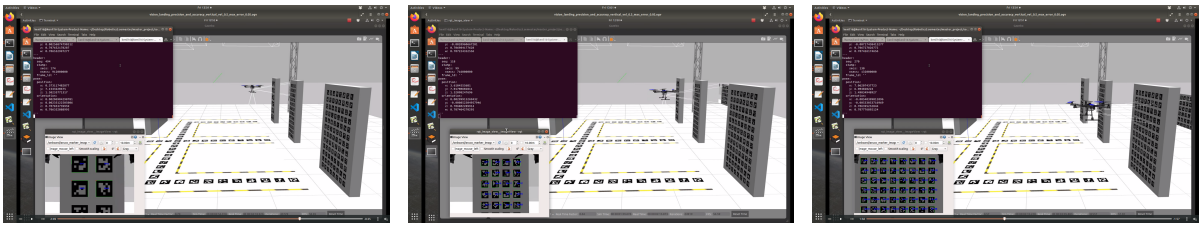
In order to replicate the tests, the command in Listing 19 can be run from the terminal.

```
#For execution of the test
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world
  ↪ drone_control_args:="vision_navigation_test" x:=-3.0 y:=0.0 headless:=false gui:=true
```

Listing 19: Command to be used to replicate the test

Here the *.world* file must be changed in Listing 19 to optitrack_big_board_full.world, optitrack_big_board_onepattern.world and optitrack_big_board_onepattern_missing_markers.world for the configuration in Figure 46a, 46b, 46c and for the configuration with added custom wear optitrack_big_board_onepattern_missing_markers_wear.world for 46d respectively.

4.1.6 *Vision based landing*

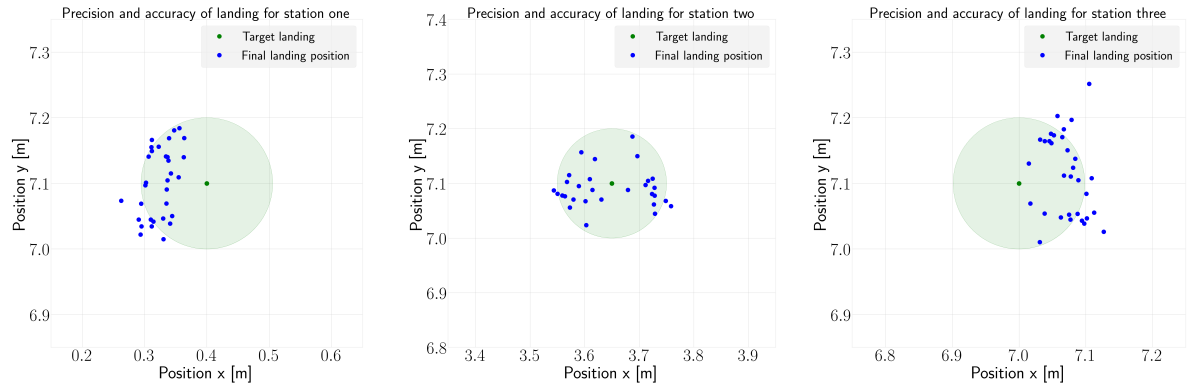


(a) *(b)* *(c)*

Figure 50: Illustrations of the landing test. The UAV is set to move between the landing stations in a random order and make a landing using either landing station one, two or three which can be seen in Figures 50a, 50b and 50c respectively. A video of the first couple of landings in the tests can be seen on Github using Vision based landing

Table 11: Statistics of the results from landing tests. Each test runs for 100 landings randomly picked between landing stations. The error between the wanted landing setpoint and actual end position is defined with minimum, maximum, mean and STD error. The stabilize time is the time it takes the UAV to settle in front of the ArUco landing board before beginning the actual landing. The waypoint error defines the maximum allowed error, as already mentioned, before moving on to the next waypoint (landing is this case). It may be noticed that a decrease in stabilize and landing time comes with a cost of a larger final landing error. Also the velocity (vertical) has an impact on the final landing error

Station	Runs	Success	WP error	Vel	Min	Max	Mean	STD	Stabilize time	Landing time
					Test 1					
One	35	32	$0.1m$	$0.1\frac{m}{s}$	$1.60cm$	$12.46cm$	$5.81cm$	$3.02cm$	$3.60s$	$11.51s$
two	30	28	$0.1m$	$0.1\frac{m}{s}$	$0.70cm$	$11.83cm$	$5.79cm$	$2.54cm$	$0.70s$	$11.76s$
Three	35	32	$0.1m$	$0.1\frac{m}{s}$	$0.62cm$	$10.75cm$	$5.86cm$	$2.50cm$	$3.22s$	$11.82s$
					Test 2					
One	33	25	$0.1m$	$0.5\frac{m}{s}$	$2.20cm$	$15.04cm$	$8.00cm$	$2.67cm$	$2.63s$	$3.00s$
two	29	25	$0.1m$	$0.5\frac{m}{s}$	$3.18cm$	$11.60cm$	$7.62cm$	$2.19cm$	$0.20s$	$3.00s$
Three	38	27	$0.1m$	$0.5\frac{m}{s}$	$1.11cm$	$14.63cm$	$7.87cm$	$3.25cm$	$2.18s$	$3.02s$
					Test 3					
One	32	19	$0.1m$	$0.9\frac{m}{s}$	$4.63cm$	$14.01cm$	$9.25cm$	$2.27cm$	$1.68s$	$2.15s$
two	36	18	$0.1m$	$0.9\frac{m}{s}$	$4.79cm$	$14.00cm$	$9.54cm$	$1.92cm$	$0.19s$	$2.13s$
Three	32	21	$0.1m$	$0.9\frac{m}{s}$	$3.35cm$	$18.45cm$	$9.45cm$	$2.88cm$	$2.09s$	$2.18s$

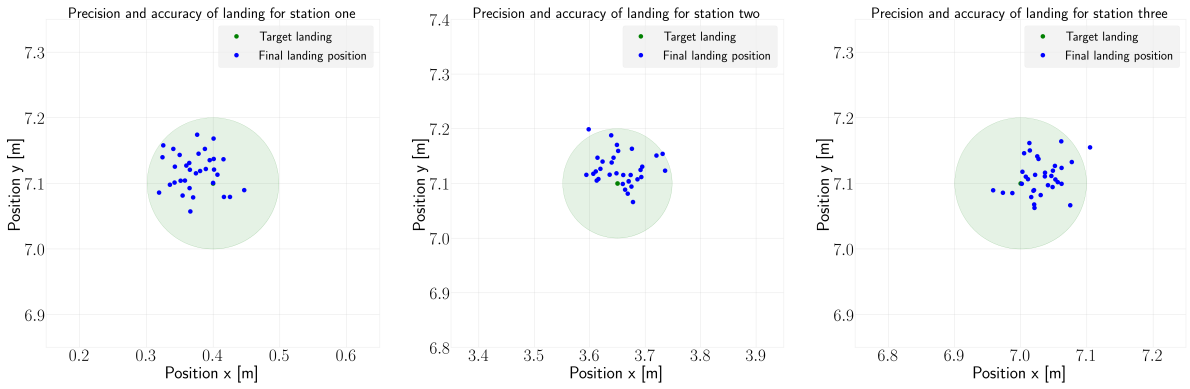


(a) *(b)* *(c)*

Figure 51: Illustrations of the final landings seen from above for test 3 in Figure 51a, 51b and 51c for landing station one, two and three respectively. It may be noticed the error is quite big where the wanted landing position should be in the center of the green circle which illustrates a threshold for acceptance for ± 10 centimeters

Table 12: Statistics of the results from landing tests. It can be seen that the waypoint check error is set lower, meaning the UAV has to settle even more before landing. The error in the final landing position decreases quite a lot. But again this comes with a cost of a longer stabilization time. In the best case as seen from test 4, the mean landing position error is decreased to be in the order of centimeters with a maximum error just above ten centimeters

Station	Runs	Success	WP error	Vel	Min	Max	Mean	STD	Stabilize time	Landing time
				Test 4						
One	33	33	$0.05m$	$0.1\frac{m}{s}$	$0.08cm$	$9.46cm$	$4.81cm$	$2.15cm$	$9.93s$	$11.45s$
two	33	32	$0.05m$	$0.1\frac{m}{s}$	$0.97cm$	$11.18cm$	$4.74cm$	$2.56cm$	$6.96s$	$11.69s$
Three	34	33	$0.05m$	$0.1\frac{m}{s}$	$0.19cm$	$11.84cm$	$4.54cm$	$2.41cm$	$9.11s$	$11.58s$
				Test 5						
One	32	32	$0.05m$	$0.5\frac{m}{s}$	$1.63cm$	$9.84cm$	$5.25cm$	$2.36cm$	$8.43s$	$3.00s$
two	32	29	$0.05m$	$0.5\frac{m}{s}$	$1.31cm$	$13.07cm$	$5.98cm$	$2.90cm$	$6.34s$	$3.00s$
Three	36	35	$0.05m$	$0.5\frac{m}{s}$	$0.45cm$	$12.22cm$	$4.98cm$	$2.88cm$	$9.33s$	$3.00s$
				Test 6						
One	32	30	$0.05m$	$0.9\frac{m}{s}$	$0.20cm$	$11.89cm$	$4.96cm$	$2.77cm$	$9.18s$	$2.28s$
two	35	32	$0.05m$	$0.9\frac{m}{s}$	$0.65cm$	$14.13cm$	$5.90cm$	$3.32cm$	$5.91s$	$2.34s$
Three	33	31	$0.05m$	$0.9\frac{m}{s}$	$1.06cm$	$13.54cm$	$5.35cm$	$2.70cm$	$8.54s$	$2.36s$



(a) *(b)* *(c)*

Figure 52: Illustrations of the final landings seen from above for test 4 in Figure 52a, 52b and 52c for landing station one, two and three respectively. It can be seen that lowering the checkpoint error as well as decreasing the landing speed (vertical velocity), the accuracy and precision of the landings increases quite a lot

To see how well the UAV is capable of performing landings when using the ArUco pose estimations, the UAV was set to start at landing station one. Then randomly move between landing stations and each time perform a landing where the landing position would be compared to the target landing position for error analysis. This can be seen in Figure 50. What goes for all the tests is a horizontal velocity of 1 $\frac{m}{s}$.

In order to replicate the tests, the command in Listing 20 can be run from the terminal.

```
#For execution of the test
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world
  ↪ drone_control_args:="landing_test" x:=3.0 y:=3.3 headless:=false gui:=true
```

Listing 20: Command to be used to replicate the test

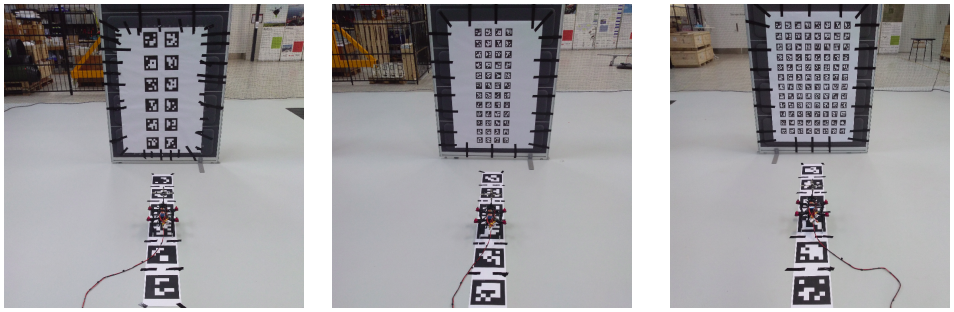
4.2 OptiTrack

This section includes real flight of the UAV in the OptiTrack system after successfully completing all initial tests in simulation from Section 4.1. Section 4.2.1 will investigate the estimated pose of the ArUco marker landing boards when the UAV is located on the ground in steady state. This is done to see how accurate the pose estimations are and to align the ArUco marker boards to that of the OptiTrack system in order to achieve millimeter precision. In Section 4.2.2, the ability of the UAV to keep its pose in front of the landing boards using ArUco pose estimations will be performed. Lastly, the vision based landings of the UAV are performed in Section 4.2.3.

One may notice that the order of the tests does not follow the ones in Section 4.1 exactly. This is done to reduce the risk of failures because implementation on real hardware always has to be taken with care. Hence, the UAV has to work perfectly using computer vision for pose estimation using both the front camera for vision based landings and the bottom camera for vision based navigation using sensor fusion before the actual GPS to vision procedure is executed. However, because of time constraints and the limited access to the airport, tests with sensor fusion using the bottom camera and the GPS to vision based procedure have not been performed. These will be discussed for future work in Section 7.

4.2.1 Steady state ArUco pose estimation

The UAV were placed approximately 1.5 meters in front of the ArUco marker landing boards as seen in Figure 53. The reason for placing the UAV 1.5 meters and not 1.0 meter in front of the boards, which was done in the simulations, was because of blind angles in the OptiTrack system from where no ground truth data was returned. Hence, to avoid this scenario, this pose were chosen. Each run lasted approximately 50 seconds where the pose estimations were compared to the ground truth for performance evaluation.

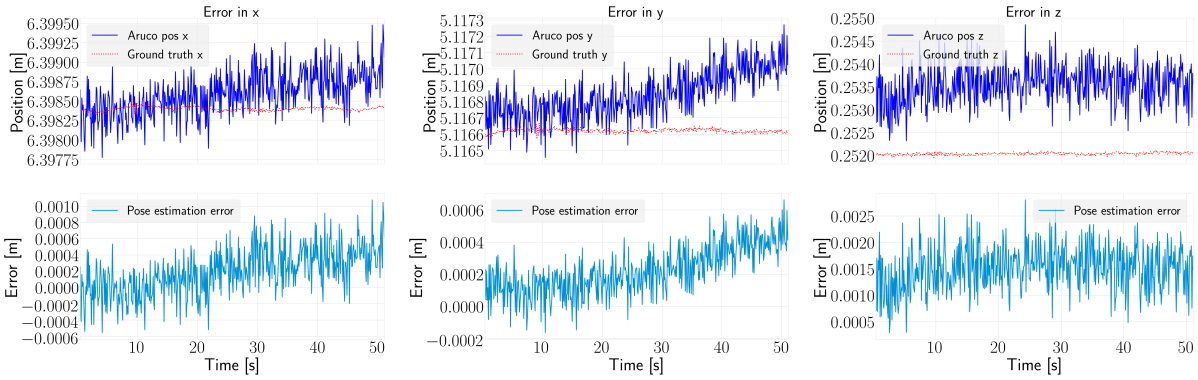


(a) *(b)* *(c)*

Figure 53: Illustration of steady state ArUco pose estimation test. Here the UAV can be seen in front of the landing marker board one, two and three in Figure 53a, 53b and 53c respectively

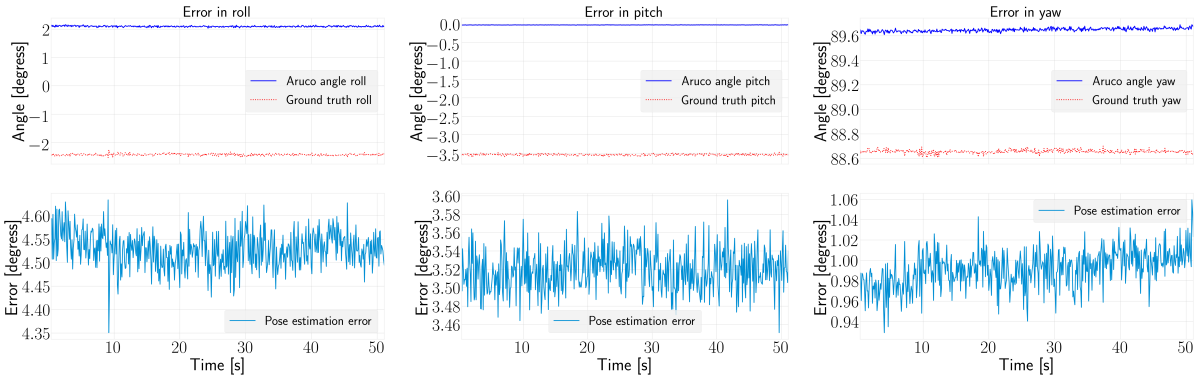
Table 13: Statistics of the results of the steady state ArUco pose estimation test. Here it may be noticed that a mean error in the order of a couple of degrees is present. This is because of a misalignment of the ground truth from the OptiTrack system to that of the estimations from the ArUco marker boards. The reason for this is the legs attached to the UAV not being perfectly equal in height and the stand, in which the ArUco markers boards is placed, does tilt just a little which means the board not being totally perpendicular to the ground. These offsets can be seen in Figure 55

Estimation error	Runs	Wind	Mean position	STD position	Mean angle	STD angle
			Test 1			
ArUco pose landing board one	5	No	$0.60cm$	$0.05cm$	2.0°	0.04°
			Test 2			
ArUco pose landing board two	5	No	$0.21cm$	$0.10cm$	1.89°	0.06°
			Test 3			
ArUco pose landing board three	5	No	$0.13cm$	$0.01cm$	3.15°	0.02°



(a) *(b)* *(c)*

Figure 54: Illustration of the ArUco position estimation error using the configuration in Figure 53c. It may be noticed that the error is quite small and only negligible deviations in the pose estimations exist compared to the ground truth from the OptiTrack system

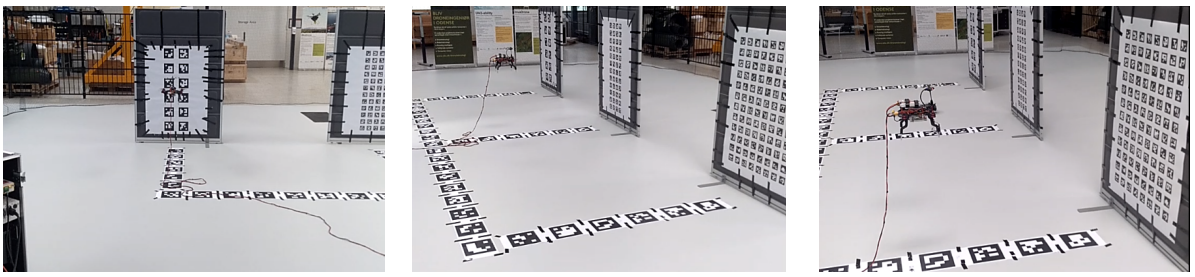


(a) *(b)* *(c)*

Figure 55: Illustration of the ArUco angle estimation error using the configuration in Figure 53c. It may be noticed that an offset exist which is more noticeable in roll and pitch in Figure 55a and 55b due to a misalignment of the OptiTrack system and the estimated ArUco marker boards

4.2.2 Hold pose using estimated ArUco pose

The UAV was placed approximately 1.5 meters in front of the landing markers with the objective to keeps its pose for 30 seconds where the estimated ArUco marker pose were compared to that of the ground truth for performance evaluation.

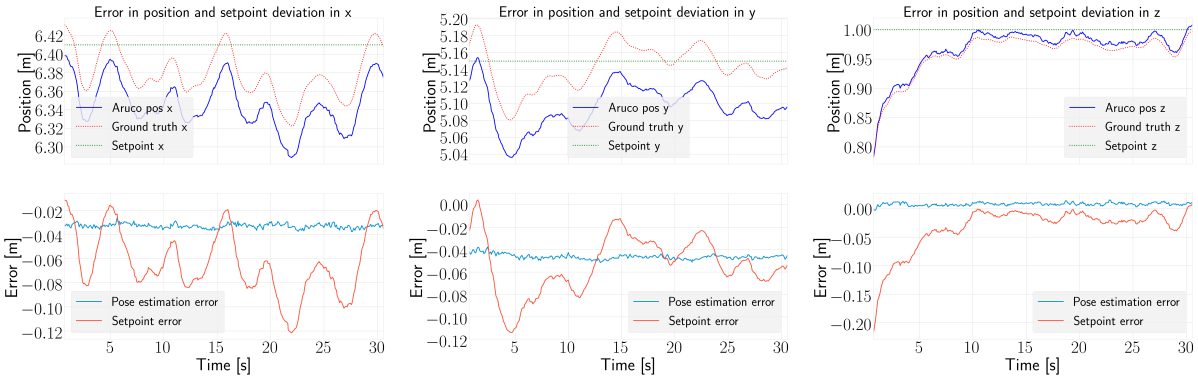


(a) *(b)* *(c)*

Figure 56: Illustrations of the different configurations of the tests. The ArUco landing boards are used in Figure 56a, 56b and 56c for landing station one, two and three respectively. Videos of the tests can be seen on Github using Landing station one, Landing station two and Landing station three

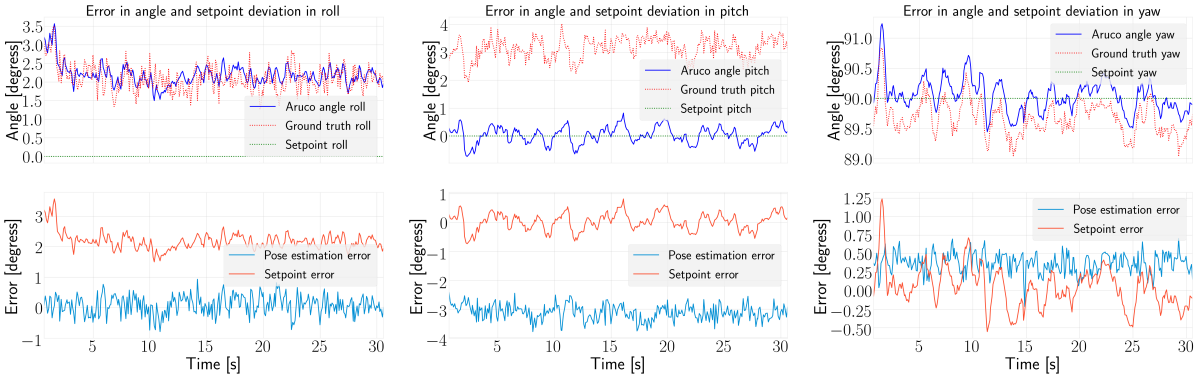
Table 14: Statistics of the results of holding the pose using the ArUco landing boards. It may be noticed that the error is in the order of centimeters for the position. This error is primarily due to an offset in the ArUco boards in regard to the OptiTrack system. This is clearly seen in Figure 57 and 58. Hence, the actual difference between the pose estimated from the ArUco boards and Optitrack system is quite small. Moreover, negligible fluctuations in the pose estimations yields a small STD which agrees with the results in the simulations in Section 4.1.3

Estimation error	Runs	Wind	Mean position	STD position	Mean angle	STD angle
			Test 1			
ArUco pose landing board one	5	No	$2.99cm$	$0.35cm$	0.83°	0.18°
Setpoint	5	No	$3.80cm$	$3.19cm$	0.69°	0.27°
			Test 2			
ArUco pose landing board two	5	No	$2.56cm$	$0.31cm$	0.97°	0.17°
Setpoint	5	No	$5.36cm$	$3.13cm$	0.28°	0.21°
			Test 3			
ArUco pose landing board three	5	No	$2.97cm$	$0.23cm$	1.17°	0.22°
Setpoint	5	No	$4.54cm$	$3.03cm$	0.87°	0.22°



(a) *(b)* *(c)*

Figure 57: Illustration of the position estimation and setpoint error using configuration in Figure 56c. The ArUco position estimation can be seen to follow the ground truth of the OptiTrack system quite nicely and only deviates in an offset

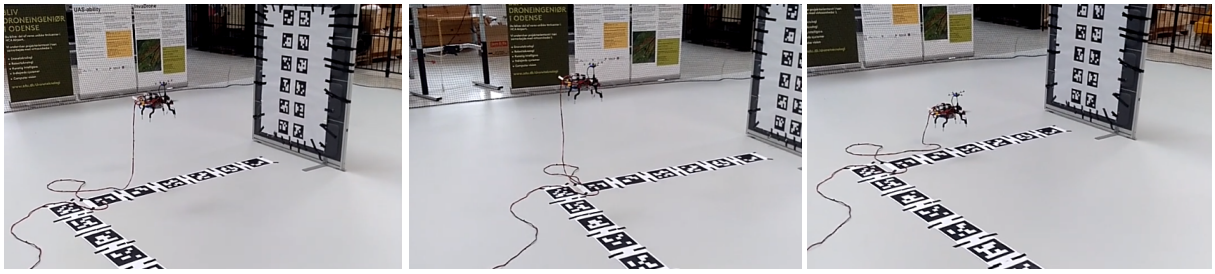


(a) *(b)* *(c)*

Figure 58: Illustration of the angle estimation and setpoint error using configuration in Figure 56c. The ArUco angle estimation can be seen to follow the ground truth of the OptiTrack system quite nicely and only deviates in an offset

4.2.3 *Vision based landing*

An overall number of sixty vision based landings were performed. An illustration along with the procedure can be seen in Figure 59. This is not the exact same procedure as in the simulation in Section 4.1.6. The changes have been made to make the landings using the three different landing boards as equal as possible because a reduced number of landings have been performed due to time constraints.

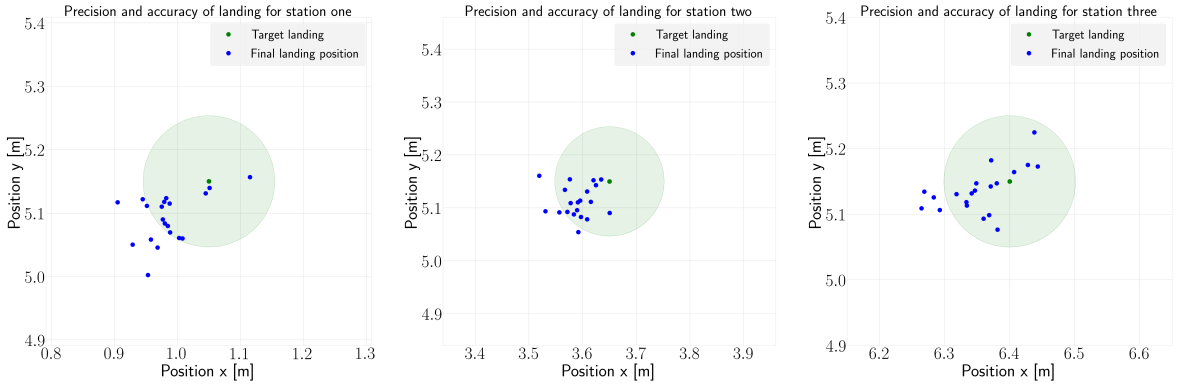


(a) *(b)* *(c)*

Figure 59: Illustrations of the landing test. The UAV is set to takeoff as seen in Figure 59a, then move approximately a meter backwards as seen in Figure 59b and then back to its initial pose for execution of a vision based landing seen in Figure 59c. This is repeated for twenty runs in each test. A video of the first two iterations of test one can be seen on Github using Vision based landing using landing board one

Table 15: Statistics of the results from the vision based landing tests. Each test runs for 20 landings all with a WP error of 0.1 and vertical velocity of 0.2 $\frac{m}{s}$. As it may be seen, the UAV makes decent results in test two and three using landing boards two and three respectively. However, the landings deviates quite a lot in test one. This is clarified in Figure 60

Station	Runs	Success	WP error	Vel	Min	Max	Mean	STD	Stabilize time	Landing time
				Test 1						
One	20	11	$0.1m$	$0.2\frac{m}{s}$	$1.04cm$	$17.68cm$	$9.76cm$	$4.07cm$	$1.45s$	$6.05s$
				Test 2						
Two	20	16	$0.1m$	$0.2\frac{m}{s}$	$1.58cm$	$13.21cm$	$7.66cm$	$3.22cm$	$1.85s$	$6.25s$
				Test 3						
Three	20	16	$0.1m$	$0.2\frac{m}{s}$	$1.57cm$	$14.19cm$	$6.99cm$	$3.56cm$	$1.20s$	$6.55s$



(a) *(b)* *(c)*

Figure 60: Illustrations of the final landings seen from above for test one, two and three in Figure 60a, 60b and 60c respectively. The wanted landing position should be in the center of the green circle which illustrates a threshold for acceptance for ± 10 centimeters

5 Discussion

Initial pose estimation using the GPS to vision ArUco board

From the results in Section 4.1.1 a circular region of poses where the ArUco pose estimation error is quite low was found to be within a maximum distance of approximately 5 meters away from GPS2Vision board. The errors can be seen to be high when the UAV is approximately 5-8 meters away, or more, from the GPS2Vision board which goes for both position and angle. From these observations, a safe area for the transition between using GPS to vision as navigation would be within a radius of approximately five meters away from the GPS2Vision board which can be seen in Figure 32.

A possible reason why especially the z position deviates a lot when moving away from the board is that the z component in the coordinate system moves with an angle in roll which triggers the altitude of the UAV to go from positive to negative and hence increases the error in z more than x and y which are less sensitive according to the analysis. It is also quite interesting to see that the position in z is more sensitive when moving away from the board when being straight in front of it. This can be seen in Figure 32c. By being further away from the board does not yield the same kind of sensitivity for the position in x and y, but they are more sensitive when facing the board when the UAV is not directly in front of the it as seen in Figure 32a and 32b.

Based on the results, the maximum distance away from the GPS2Vision board when executing the GPS to vision transition should not exceed five meters with the current configuration of the ArUco marker board and resolution of the camera which was set to 640×480. However, it is possible to reduce the errors by having a better resolution because the detection of markers would be more precise due to the fact the more pixels would yield a better separation of the borders of the markers which in turn produces better pose estimates. This could yield a better performance of the system, but may also stress the Raspberry Pi in regard to calculations performed. A way to accommodate for that is to use a stronger companion computer e.g Jetson nano, which comes with the expense of a higher price.

Analyzing rolling average for fluctuations in the pose estimation

As it can be seen in Figure 34 and 35 in Section 4.1.2, the fluctuations in the estimates are only of minor concern when the UAV is of a maximum distance of 5 meters from the GPS2Vision board. However, when the UAV changes to a position being more than five meters away from the GPS2Vision board, the estimations in position fluctuates a lot as seen in Figure 36. This could be fatal to make a transition in this scenario and lead to an unstable system. The estimation of the orientation of the UAV is affected less as seen in Figure 37. Hence, the outcome of the rolling average has been used alongside the maximum distance to the GPS2Vision board for stability analysis before initiating the GPS to vision transition.

One may argue that being close enough to the ArUco marker board should be a valid criteria for this transition, but for safety precautions this extra criteria has been considered in the implementation. Nonetheless, the calculations of the rolling average puts more pressure on the Raspberry Pi and could be neglected if found redundant.

Hold pose using ArUco pose estimations

It is interesting to see the performance boost in the reduction of the pose estimation error in Section 4.1.3 when more ArUco markers are visible in the image. Especially by analyzing Table 7 and 8, by having the UAV closer to the ArUco marker board decreases the overall pose estimation and target setpoint error significantly. Furthermore, it results in a more robust system because of fluctuations in the pose estimations are less.

As it can be seen in test two in Table 7, introducing wind in the simulation only gives negligible changes in the error for pose estimation. However, because wind affects the placements of the UAV, the setpoint error is bigger. This is also seen in Figure 40a and 40b where the UAV has a constant offset in roll and pitch due to the applied wind. From this analysis, the UAV is capable of keeping its pose even in windy conditions using the implemented control schemes in the PX4 autopilot. The important thing to note here is that the stability of the system depends on a constant flow of pose updates from the ArUco pose estimation and the amount of pose updates at each time instant. Because of the wind, the UAV may come in a situation where it looses sight of the ArUco board which puts a limit to the maximum allowed wind from where the

system becomes unstable. From tests, this has been found to be in the order of 7-10 $\frac{m}{s}$ from results in Section 4.1.4.

By analyzing Table 8, it may be noticed the negligible error in the mean position and angle. By examining test five, the mean error in x, y and z is only 0.9 millimeters and a mean angle error of 0.02 degrees. Moreover, the mean error in position in test three only adds 3 millimeters to the position and 0.01 degrees to the angle from that of test five. This concludes that the number of markers in the image only adds little gain to the pose estimation. The biggest gain comes by having a sufficient area of the image covered by markers as seen in Figure 38a. The great performance of the ArUco pose estimation can be seen in Figure 41a, 41b and 41c for the positions and 42a, 42b and 42c for the angles.

These conclusions can be supported by the real flight tests from the OptiTrack system in Section 4.2.2 where the UAV was set to keep its pose 1.5 meters in front of the landing boards with an altitude of 1.0 meter. As seen in Table 14 and Figure 57 and 58, similar results were accomplished. However, as seen in Figure 57 and 58, an offset do exist because of misalignment of the OptiTrack system. The reason for this is because that small changes to the markers located on top of the GPS module of the UAV could yield small changes in the estimated ground truth. Although these were set to be aligned in Section 4.2.1, minor deviations to this configuration could occur when moving and touching the UAV. This was a general problem because it was pretty hard to make the ArUco marker boards be perfectly aligned with the OptiTrack system for millimeter precision. Nonetheless, though there exist a small offset, the estimated ArUco marker pose is still in the range of millimeters with only negligible fluctuations in the pose estimations. Furthermore, by analyzing the videos of the flight tests, the UAV can be seen to keep its pose quite nicely which confirms the working of the implementation for offboard control.

GPS to vision transition

The results of testing the implementation of the GPS to vision transition can be seen in Section 4.1.4. In general it yielded quite good and reliable performance as seen Table 9 where almost all runs were completed successfully in a reasonable amount of time. The velocity of the UAV was set to a higher value when using GPS compared to that of using vision (using the GPS2Vision marker board for pose estimates), because instabilities of the system could occur when the UAV was flying towards the GPS2Vision marker board in the last step of the transition as seen in Figure 44. This instability was due to the UAV having a large angle in roll or pitch when flying fast which caused the UAV to loose sight of the GPS2Vision marker board located on the wall. This puts a limit to the current implementation of the system in regard to horizontal velocity in the GPS to vision transition which must not exceed 2.0 $\frac{m}{s}$.

The issue of losing sight of the markers was also observed when the UAV changed the use of camera which can be seen in Figure 43e, where the UAV begins to use data from the bottom camera for pose estimation of ArUco markers located on the ground. In this case, the reason for losing sight of the markers located on the ground was due to a strong wind applied to the simulation in test three in Table 9. Here it may be noticed that the UAV were only able to complete 17 out of 20 runs. This is because the UAV lost sight of the markers located on the ground which forced the UAV to land on the ground. As already mentioned, this puts a limit to the implementation where wind speeds of more than 7-10 $\frac{m}{s}$ may cause instabilities to the system and should be avoided. Even though strong winds were applied, the UAV had no problems of navigating to the GPS2Vision marker board as seen in Figure 44b. Here it can be seen that the ground truth of the routes that the UAV takes from an initial position through the substates of the GPS to vision transition, is more disturbed than the one in Figure 44a due to the wind. This concludes that the system works, but has its limitations because of the fixed cameras attached to the UAV. A way to mitigate this issue, is to use gimbal cameras instead. This type of camera is equipped with motors and sensors which makes it capable of stabilizing its pose. By implementing this solution to the UAV, the just described issues could be solved and thereby making the system more robust to external influences e.g wind.

Vision based navigation

Results of vision based navigation utilizing sensor fusion can be seen in Section 4.1.5. The performance of the implementation can be seen in Table 10 where the used ArUco marker boards can be seen in Figure 46. It is quite remarkable that the error in pose estimation does not seem to grow between test one and two using the configuration in Figure 46a and 46b respectively. Here the error is 2.44 centimeters for the mean position and 1.59 and 1.30 degrees for the mean angles.

One would expect that the error should increase as the number of ArUco markers decreases in the image. A possible explanation to this is that when the UAV is flying, an hereby having an angle in either roll or pitch, the outermost markers visible in the image gives bad results because of the perspective in the image which causes the resulting error in the ArUco pose estimations to increase. The possibility for this scenario is reduced in the configuration in Figure 46b, but now a lesser number of markers in the image yields the same corresponding error in the pose estimation. The error first seems to increase when a minimum amount of ArUco markers are located on the ground as seen in the configuration in Figure 46c and 46d. This increased error is due to the system does not get the same amount of updates from the ArUco pose estimation and has to rely more on the noisier measurements from IMU and barometers used in the sensor fusion algorithm. However, it must be mentioned that the biggest contributor to the increased error comes from a time delay between the calculations of sensor fusion and the ground truth pose of the UAV. This can be seen in Figure 47 and 48. Here it can be noticed that the sensor fusion pose estimation lacks behind that of the ground truth which is quite easy to see when the UAV changes its pose in a fast manner. This results in big errors especially in the yaw angle when shifting from 180° to -180° which is seen in Figure 48c. Moreover, the estimation of the altitude seen in Figure 47c can be seen to fluctuate quite a lot. This is due to the fusion with barometer data which is more noisy then that of the ArUco pose estimation. Here the measurement noise matrix for barometer measurements could be fine tuned in the UKF to yield better performance by adding more noise to this measurement in the matrix.

Even thought a time delay due exist it does not cause the system to be unstable because the UAV was capable of finishing all runs in all tests in Table 10. Visualizations of some of these tests can be seen in Figure 49. Here a top view of the ground truth path of the UAV can be seen, where it starts in front of the GPS2Vision marker board and then randomly moves to one of the three landing stations and back again to its initial position. From Figure 49a, the UAV can be seen to follow the wanted trajectory quite nicely. Here it may be noticed that the UAV does not always end up at the target waypoint which is illustrated by a blue dot. However, this is due to the waypoint check error being 0.1 meters. Lowering this threshold would yield the UAV being closer to the target waypoints, but this would make the UAV wait longer at each waypoint for the system to settle before moving on to the next target.

What goes for the test in Figure 49a and 49b, is that the horizontal velocity of the UAV was set to 1 $\frac{m}{s}$. However, in Figure 49c, a stress test was made to see how well the UAV was able to manage high velocities which in this case was 5 $\frac{m}{s}$. As it can be seen, the UAV does a great job even though a minimum amount of markers are visible. In this case the system relies heavily on measurements from IMU and barometer date in sensor fusion to give updates for new pose estimates to the UAV. Like already mentioned, the reason that the UAV does not follow the trajectory completely, is because the system does not have time to settle at each waypoint before moving on to the next due to the choose of waypoint check error.

It must be mentioned, that no blur was added to the images in the simulations. Blur in the images can make considerable performance decreasement if the camera cannot handle fast changes in the image. Ways to take this into account is to alter the settings in aperture or exposure time in the camera configuration. This controls the opening of the lens in regard to aperture and how much light the sensor absorbs before the image is processed in regard to exposure time. Especially the latter can be altered which effectively removes blur in the images, but comes with the expense of the darker image due to lesser amount of light being absorbed in each frame. These considerations much be taken into account for the operational environment in question where a camera with a very high frame rate must be considered if the UAV has to move with great velocities when using the bottom camera to detect markers for pose estimation.

Another thing to mention is the implementation of the noise estimation in the accelerometer and gyro. As discussed in Section 3.4.2, this noise is estimated by taken ten seconds of measurements when the UAV is steady and then finding the average of these measurements to be subtracted from new measurements of IMU data to effectively remove noise. However, this method could be effected by errors if the UAV does not stand on a completely flat surface. This is no problem in the simulation, but in real life it could lead to errors in the estimation. This could also be due to the PX4 attached to the UAV having a small angle which could lead to the same error. A possible implementation to mitigate this issue is to use the markers to estimate the error in the IMU data as a sort of error state filter. This was actually considered, but not implemented due to time constrains. An implementation proposed by Isaac Skog [15] where an EKF propagates and estimates the error states using GPS and IMU data for correction could have been considered implemented in the

already created UKF from Section 3.4.2. This way the error could be estimated in flight when reliable pose estimates from ArUco markers were present.

Vision based landings

The results of the landing tests performed in simulation can be seen in Section 4.1.6. A total of six hundred landings were performed in different configurations for testing the stability of the system. A summarization of the first three hundred landings can be seen in Table 11. In this configuration, the waypoint check error (WP error) is set to 0.1 for test one, two and three. However, the vertical velocities were set to 0.1 $\frac{m}{s}$ in test one, 0.5 $\frac{m}{s}$ in test two and 0.9 $\frac{m}{s}$ in test three. These changes in velocity can clearly be seen to have an effect on the minimum, maximum, mean and STD error. This can be seen by looking at the mean error which is quite acceptable in test one where the mean error is approximately the same for all landing stations with the smallest being a 5.79 cm offset to the target. The increased accuracy comes at the cost of a longer landing time. Here the smallest landing time is 11.51 seconds and smallest stabilization time 0.70 seconds. It may be noticed that the stabilization time in general is much shorter for landing station two for both test one, two and three. A possible reason for this is that the UAV does not have a long flight time before reaching landing station two from landing station one and three. This will yield a shorter stabilization time before the UAV begins to land. In test three, the landing time is at the lowest. This yields a general landing time in a few couple of seconds. In this test the mean error is just below the maximum wanted criteria of 10 centimeters. However, the worst landing error is in the order of 18.45 centimeters.

A visualization of this test can be seen in Figure 51, where the target landing is in the middle of the green circles with an acceptance region of 10 centimeters in radius. It may be noticed that the final landing positions are primarily located to the left of the target in Figure 51a and to the right in Figure 51c. This is because the UAV comes from right to left in Figure 51a and left to right in Figure 51c where the UAV is not given time to properly stabilize before the landing is initiated as already mentioned. In this scenario, it yields both a bad accuracy as well as precision. In Figure 51b the accuracy is much better but still with a bad precision. The overall success rate of the final landings inside of the acceptance region was found to be 92% for test one, 87% for test two and 58% for test three based on the number of runs and successes defined in Table 11.

In test four, five and six in Table 12, the waypoint check error can be seen to be decreased which means that the error in waypoint checking must be below 0.05. Using this configuration yields a longer stabilization time, but gives a much better mean error along with a better accuracy and precision in the final landings as seen in Figure 52. The best performance can be seen in test four with a mean error of 4.54 cm and overall landing time of 21.38 seconds. The overall success rate of the final landings inside of the acceptance region was found to be 98% for test four, 96% for test five and 93% for test six based on the number of runs and successes defined in Table 12.

An interesting point to note is that the amount of markers visible in the image do not seem to yield a performance improvement in this configuration with the current setup of waypoint check error and velocities. However, a reason for this could be that the amount of ArUco markers visible in the image for the configuration in Figure 50a, 50b and 50c yields very reliable pose estimates and hence the PID controllers of the UAV would have to be tuned in order to achieve even better performance.

Due to time constraints only sixty vision based landings were performed in the OptiTrack system in Section 4.2.3. Here the WP error were set to 0.1 and vertical velocity to 0.2 $\frac{m}{s}$ for test one, two and three in Table 15. This configuration was chosen based on the results in Section 4.1.6 which yielded better results when the vertical velocity for landings was decreased. When analyzing Table 15 along with Figure 60, it can be seen that the UAV performs better using ArUco marker landing boards two and three. The best performance can be seen in test three with a mean error of 6.99 cm and overall landing time of 7.75 seconds. By comparing to the results in Table 11, similar results was achieved when the WP error were set to 0.1. This confirms that the implementation of the vision based landing performs well in real flight although two different UAVs were used. For better performance, the WP error could be decreased as well as the vertical velocity. The number of landings in the acceptance region for the results in Table 15 were 55% for test one and 80% for test two and three. However, thought these results are decent, improvements could be made to achieve even better performance.

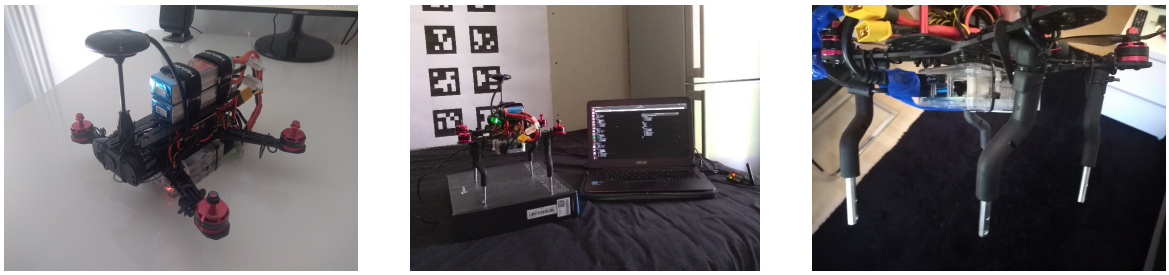
The implementation could be improved by altering the PID parameters in the pose control scheme of the PX4 autopilot. This could make the pose controller respond more quickly to changes in the pose of the UAV and hence give better pose control when new setpoints were passed to the system.

Another improvement to the vision based landing procedure could be to take inspiration from Guanya Shi et al. [7] who proposed a deep learning based robust nonlinear controller. In this implementation a deep neural network (DNN) is used. This setup makes it possible for the system to learn the parameters of the control system which makes it capable of flying very close to the ground. This is because the implementation reduces the aerodynamic ground effect. That way the control system learns to handle the extra airflow caused by the aerodynamic ground effect which yields a smother and precise landing.

Compared to similar results from other work the implementation is quite acceptable. In [18, p. 5] they achieve an average error of 6 centimeters for a number of ten vision based landings. Here the result is based on real flight tests of a UAV in indoor environments. In [16, p. 56] they achieve a 10 centimeter offset from the wanted landing position. However, it must be mensioned that the latter result was based on outdoor tests on a real implementation of a UAV which was also the case in [22, p. 6] who achieves a mean error offset from the wanted landing position of 18 centimeters.

General challenges

A general challenge has been that access to the airport was granted quite late in the period of the master thesis. Hence, much of the work with the Holybro Pixhawk 4 mini QAV250 Quadcopter was done from home where the cameras and the landing gear were attached to the UAV. An illustration of the initial testing can be seen in Figure 61b. The used landing gear was actually a suboptimal solution because it was not properly attached to the UAV a all times as seen in Figure 61c. This can clearly be seen in the video from Section 4.2.3 , where the UAV does not stand properly after the second iteration of the landings.



(a) *(b)* *(c)*

Figure 61: Illustration of the UAV without the attached landing gear in Figure 61a. Some of the initial testing of the vision based pose estimation and working with the UAV from home can be seen in Figure 61b. One of the legs can be seen to be bent after a number of vision based landings in the airport which can be seen in Figure 61c

However, without the landing gear there was no room for the Raspberry Pi as seen in Figure 61a. Hence, this implementation was chosen thought not being optimal. As mentioned in Section 3.2.1, the use of the SDU drone would be the preferred solution, but due to time constrains, this was not implemented in the flight test in the OptiTrack system. Moreover, the UAV had a hard time flying with the two batteries attached as seen in Figure 61a. This was because the center of mass being above the actual center of the UAV which led to instabilities. Hence, a wired power connection to the PX4 was attached to the UAV as seen in Figure 56 which solved the problem. Furthermore, it may the noticed that the UAV takes off quite slowly in the videos in Section 4.2.2 and 4.2.3. This is due to the extra load on the UAV e.g Raspberry Pi, landing gear etc. Improvements could be made by altering the default settings of the PID controllers in the PX4 autopilot. However, due to time constraints and the fact that the UAV was actually flying pretty well, though being slow, no change to the controllers was performed.

6 Conclusion

The implementation of the offboard control system was based on ROS and the PX4 autopilot. This way a number of nodes in the robot operating system for drone control, autonomous flight, pose estimation using ArUco marker boards and sensor fusion could be run concurrently in order to achieve autonomous flight of the UAV for Requirement **R.1**. Furthermore, this software was tested in Gazebo simulations where a number of missions could be set to be executed autonomously using keyboard commands. This software was fetched to a Raspberry Pi 4B which was used as companion computer for the Holybro Pixhawk 4 mini QAV250 Quadcopter. Hence, using the wireless module of the Raspberry Pi 4B for wireless communication to the laptop, these missions could be send through a wireless connection to be executed on the UAV. This satisfies Requirement **R.3**.

In order to use computer vision for pose estimation, a Python implementation of OpenCV was used. This gives the opportunity of detecting ArUco marker boards, where the pose estimation error were in the range of millimeters if the UAV was placed approximately one meter in front of the ArUco marker boards. Moreover, it can be concluded that if more markers are visible in the image, better pose estimations can be achieved. For navigation between ArUco marker boards, transformations of the estimated ArUco marker boards for landing and the board used in the GPS to vision transition were performed to align these coordinate systems to the one located on the ground. This ensured that only a single coordinate system would have to be used for vision based pose estimation. This satisfies Problem **P.1**, **P.3** and **P.5**.

For satisfying Problem **P.2**, initial testing of the ArUco pose estimation was performed to see how far away the UAV could be from the GPSVision board before initiating the GPS to vision transition without causing instabilities to the vision based pose estimation. Results showed that the UAV had to be in a radius of maximum five meters away from the GPS2Vision ArUco marker board where fluctuations in the estimated pose were accessible giving that the GPS2Vision board being a (2×8) ArUco marker board with marker size of 0.2 meters and distance between markers of 0.1 meters. The resolution of the camera was 640×480. This way the UAV had to navigate to the GPS2Vision ArUco marker board using GPS until being close enough to the board to make a reliable GPS to vision transition. Moreover, the coordinate system of the ArUco marker board located on the ground was transformed to be aligned to the local coordinate system of the UAV when using GPS. This resulted in a smooth GPS to vision transition, because no large change in the pose were seen due to the fact that the original coordinate system was used. Hence, increments or decrements to the original pose would just be added yielding a stable flight control using the PX4 autopilot.

Furthermore, this implementation was tested in a number of different configurations with applied wind for stress analysis. Results showed that the implementation of the GPS2Vision transition were robust to applied wind in the simulations and managed to complete all runs with wind speeds up to 5-7 $\frac{m}{s}$. However, only seventeen out of twenty runs were completed with wind speeds up to 7-10 $\frac{m}{s}$ which puts an upper threshold to the current implementation of the system for maximum allowed wind before the system becomes unstable. This satisfies Requirement **R.4**. However, due to time constraints, the GPS to vision transition was only performed in simulations.

Sensor fusion was implemented to take into account scenarios where no ArUco markers where visible for the bottom camera of the UAV in vision based navigation. Here an unscented kalman filter were implemented. Results showed that the implementation worked as expected where pose estimations from ArUco markers, IMU and barometer data were fused together yielding sensor fusion for pose estimation. The implementation was tested in a number of different configurations where small errors in the range of centimeters and degrees were observed. This satisfies Problem **P.4**. However, due to time constraints, the implementation of sensor fusion was only tested in simulations.

The vision based landings were accomplished by having three different sets of ArUco marker boards. This was done to see if the number of visible amount of markers in the image yielded better final landings. Results showed a lowest mean error of 4.54 cm with overall landing time of 20.69 seconds in the simulation for 34 runs and 6.99 mean error with overall landing time of 7.75 seconds for 20 runs in real flight of the UAV. Thought Requirement **R.2** is satisfied, improvements to the system must be made to satisfy Requirement **R.5**. The velocity of the UAV can be increased to reduce the landing time, but it comes with an increased error in the landing position. This completes Problem **P.6**. Further analysis showed no significant improvements of using the three different landing boards in regard to the mean position landing error.

7 Future work

As already discussed in Section 1.4, not all planned executions of the tests in the OptiTrack system were accomplished due to the limited access to the airport. Hence, more work of testing the implementation with real flight of the UAV has to be done in order to properly verify that all parts of the offboard control works as expected before using the UAV in missions for autonomous flight. As discussed in Section 6, all problems and requirements to the system from Section 1.2 and 1.3 were completed. However, a landing time below 5 seconds for Requirement **R.5** was not achieved at all times.

Hence, this gives the following objectives to be completed before a complete real life implementation of the system can be initiated.

1. The vision based navigation must be completed in the OptiTrack system to see how well the implementation of sensor fusion works on real hardware on the UAV. These tests will follow the simulations from Section 4.1.5, where a number of missing ArUco markers are placed on the ground to see how well the implementation performs when a minimum amount of ArUco markers are present in the image. This should also be performed with high horizontal velocity to investigate how blur effects the detection and pose estimation of ArUco markers.
2. When sensor fusion has been properly tested on the real UAV, the GPS to vision transition can be executed. Here the idea is that the GPS2Vision ArUco marker board will be placed on top of the entrance to the OptiTrack system from where the ground truth pose of the UAV can be tracked to analyze its stability when going from using GPS to vision based pose estimations. Here the error in the GPS module must be taken into account, to ensure that the UAV is not too close to the entrance of the OptiTrack system when it starts to locate the ArUco marker board.
3. Fine tuning of the PID controllers in the PX4 autopilot may be changed to achieve better precision landings which could also yield a faster landing time when a faster response in pose changes is applied to the controller. Because the procedure is case specific e.g small changes to the load of the UAV, placements of hardware etc., this could be time consuming, but will yield better results if performed correctly.
4. Before execution of missions in autonomous flight in real life, a way to handle moisture and rain must be analyzed. This means that all electronics on the UAV e.g Raspberry Pi, must be encapsulated to avoid damaging the system. Moreover, the risk of raindrops on the front and bottom cameras should be considered which could potentially have an impact in the detection of ArUco markers.
5. Finally, a specific operations risk assessment (SORA) must be performed. This will have to be completed if the UAV is to participate in autonomous missions where the UAV have to operate beyond visual line of sight (BVLOS). This will ensure that all steps of the operations in a safety perspective point of view are satisfied.

Acknowledgement

I would like to thank Henrik Skov Midtiby for guidance throughout this master thesis where knowledge about robotics and especially computer vision has been at disposal in the implementation of the autonomous offboard control system for vision based navigation. Moreover, I will thank Jes Grydholdt Jepsen for advice in the implementation of the PX4 autopilot as well as guidance with the OptiTrack system which has been a crucial part of analyzing the performance of the system for real flight of the UAV.

References

[1] Basel Alghanem. *A general unscented kalman filter*. Available at: https://github.com/balghane/pyUKF. Accessed: 24-05-2021. 2017.

[2] *ArUco marker detection*. Available at: https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html. Accessed: 17-02-2021.

[3] Dias S. S. & Santos D. A. Dos Barbosa J. P. D. A. *A Visual-Inertial Navigation System Using AprilTag for Real-Time MAV Applications*. Available at: https://doi.org/10.1109/M2VIP.2018.8600901. Accessed: 17-08-2020. 2018.

[4] Michael Maurer & Jesus Pestana & Friedrich Fraundorfer & Horst Bischof. *Towards an Autonomous Vision-based Inventory Drone*. Available at: https://graz.pure.elsevier.com/en/publications/towards-an-autonomous-vision-based-inventory-drone-2. Accessed: 05-05-2021. 2019.

[5] Kurt Seifert & Oscar Camacho. *Implementing Positioning Algorithms Using Accelerometers*. Available at: https://www.nxp.com/docs/en/application-note/AN3397.pdf. Accessed: 27-04-2021. 2007.

[6] Tiago Gomes Carreira. *Quadcopter Automatic Landing on a Docking Station*. Available at: https://web.ist.utl.pt/~tiago.carreira/thesis/Thesis.pdf. Accessed: 05-05-2021. 2013.

[7] Guanya Shi & Xichen Shi & Michael O'Connell & Rose Yu & Kamyar Azizzadenesheli & Animashree Anandkumar & Yisong Yue & Soon-Jo Chung. *Neural Lander: Stable Drone Landing Control Using Learned Dynamics*. Available at: https://arxiv.org/abs/1811.08027. Accessed: 07-05-2021. 2019.

[8] Maria Alvarez Custodio. *Autonomous Recharging System for Drones: Detection and Landing on the Charging Platform*. Available at: https://kth.diva-portal.org/smash/get/diva2:1294201/FULLTEXT01.pdf. Accessed: 05-05-2021. 2019.

[9] Technische Universität Darmstadt. *Hector localization*. Available at: https://github.com/tu-darmstadt-ros-pkg/hector_localization. Accessed: 30-04-2021. 2018.

[10] Technische Universität Darmstadt. *Hector Quadrotor*. Available at: https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor. Accessed: 30-04-2021. 2018.

[11] *Detection of ArUco Boards*. Available at: https://docs.opencv.org/master/db/da9/tutorial_aruco_board_detection.html. Accessed: 12-02-2021.

[12] Wan & Rudolph van der Merwe Eric A. *The Unscented Kalman Filter for Nonlinear Estimation*. Available at: https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf. Accessed: 23-04-2021. 2000.

[13] *Estimate pose of board*. Available at: https://docs.opencv.org/3.4/d9/d6a/group__aruco.html. Accessed: 17-02-2021.

[14] *HealthDrone*. Available at: https://sundhedsdroner.dk/. Accessed: 18-08-2020. 2018.

[15] Skog Isaac. *GNSS aided INS*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.4816&rep=rep1&type=pdf. Accessed: 23-04-2021. 2005.

[16] Ziwen Jiang. *An Autonomous Landing and Charging System for Drones*. Available at: https://www.researchgate.net/publication/337473500_An_autonomous_landing_and_charging_system_for_drones. Accessed: 05-05-2021. 2018.

[17] Roger R Labbe Jr. *Kalman and Bayesian Filters in Python*. Available at: https://drive.google.com/file/d/0By_SW19c1BfhSVFzNHc0SjduNzg/view. Accessed: 27-04-2021. 2020.

[18] Mohammad Fattahi Sani & Ghader Karimian. *Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors*. Available at: https://ieeexplore.ieee.org/document/8270408. Accessed: 05-05-2021. 2017.

[19] Monteleone S. & Catania V. & De Paz J. F. & Bajo J. La Delfa G. C. *Performance analysis of visualmarkers for indoor navigation systems*. Available at: https://doi.org/10.1631/FITEE.1500324. Accessed: 17-08-2020. 2016.

[20] *LiPo battery guide*. Available at: https://rogershobbycenter.com/lipoguide. Accessed: 22-04-2021.

[21] Xuancen Liu & Shifeng Zhang & Jiayi Tian & Longbin Liu. *An Onboard Vision-Based System for Autonomous Landing of a Low-Cost Quadrotor on a Novel Landing Pad*. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6874798/. Accessed: 05-05-2021. 2019.

[22] Jamie Wubben & Francisco Fabra & Carlos Calafate & Tomasz Krzeszowski & Johann M. Marquez-Barja & Juan-Carlos Cano & Pietro Manzoni. *A vision-based system for autonomous vertical landing of unmanned aerial vehicles*. Available at: https://ieeexplore.ieee.org/document/8958701. Accessed: 05-05-2021. 2019.

[23] Henrik Skov Midtiby. *Robust detection of n-fold edges using convolution with a complex kernel*. Available at: https://github.com/henrikmidtiby/VideoObjectTracker. Accessed: 03-05-2021. 2020.

[24] Kenni Nilsson. *Vision based navigation and precision landing of a drone*. Available at: https://github.com/Kenil16/master_project. Accessed: 30-05-2021. 2021.

[25] Nikolaj Frederik Pihl Thomsen & Marcus Enghoff Hesselberg Grove & Muhammad Owais Mehmood & Kenni Nilsson. *Fence inspection for breach detection using drones autonomously*. Available at: https://github.com/Kenil16/fence_inspection_using_drones. Accessed: 30-05-2021. 2021.

[26] Frédéric Guinand & Jean-François Brethe & El Houssein & Chouaib Harik &François Guerin & Hervé Pelvillain. *Towards an Autonomous Vision-based Inventory Drone*. Available at: https://ieeexplore.ieee.org/document/7850056. Accessed: 05-05-2021. 2016.

[27] *PX4 autopilet*. Available at: https://docs.px4.io/master. Accessed: 13-02-2021.

[28] JIN Shaogang & ZHANG Jiyang & SHEN Lincheng & LI Tengxiang. *On-board Vision Autonomous Landing Techniques for Quadrotor: A Survey*. Available at: https://ieeexplore.ieee.org/document/7554984. Accessed: 04-05-2021. 2016.

[29] *The extrinsic camera matrix*. Available at: http://ksimek.github.io/2012/08/22/extrinsic/. Accessed: 17-02-2021.

[30] *Ubuntu 18.04.5 LTS (Bionic Beaver) 64-bit server operating system*. Available at: https://cdimage.ubuntu.com/releases/18.04/release/. Accessed: 22-04-2021.