NTNU
Kunnskap for en bedre verden

Department of ict and Natural Sciences

Bachelor's thesis in Electrical Engineering - Automation and Robotics

Autonomous Drone Props in Theatre

The Balance Between Artistic Design and Technical Functionality, and Different Possible Location Technologies.

By:
Jakob Andreas Amtedal

Supervisor:
Erlend Coates

Date: 20th of June, 2023

Sammendrag

Denne Bachelor's oppgaven omhandler den potensielle bruken av droner i skuespill. Motivasjonen for å ha flyvende droner i skuespill er å kunne få en mer fortryllende teateropplevelse med å ha svevende rekvisitter i skuespillet. Oppgaven ser på ulike teknologier for å kunne lokalisere dronen på scenen, samt også kontrollere den autonomt slik at dronen kan forflytte seg mellom ulike posisjoner uten å ha en ekstern pilot som styrer dronen. Oppgaven er gitt av Teateret Vårt, som ønsket å ha en svevende bok i sitt skuespill. Dermed ser oppgaven også på designkonflikter når dronen skal tildekkes som en bok for Teateret Vårt. De største konfliktene møtt er hull for luftstrøm som dronen må ha for å fly, og vekt som legges til for å gjemme dronen. To teknologier for innendørs lokalisering blir brukt for å lokalisere dronen. Pozyx's UWB (Ultra-Wideband) bruker lange radiobølger for å måle posisjon. Den andre teknologien brukt for å måle posisjonen av dronen er distribuert av Marvelmind. Marvelmind bruker ultrasonisk lyd for å lytte hvor objektet befinner seg. Begge teknologiene bruker triangluering fra ulike referansepunkter for å finne en mobil sensor som beveger seg. Hver av disse teknologiene har sine fordeler og ulemper. For styring av dronen er det to scripts som blir presentert, begge av disse har ulike spesifikasjoner. Første script er rask og effektiv, men lite presis. Andre script er treg men har høyere presisjon. Bruken av droner i teater har potesiale, men det krever både program, design og maskinvare for at dronen skal kunne fly pålitelig og trygt.

Underveis i prosjektet har en YouTube-video blitt laget. Denne videoen har som hensikt å vise frem hvordan prosjektet har blitt til så langt. Ferdig resultat er ikke med her, men viser mer til hvordan prosjektet har utviklet seg. Link: https://youtu.be/QBF4H6GgW6Y

Summary

This Bachelor's thesis focuses on the potential use of drones in theater productions. The motivation behind incorporating flying drones in plays is to create a more enchanting theater experience by having floating props in the performance. The thesis explores various technologies for locating the drone on stage and also controlling it autonomously, allowing the drone to move between different positions without an external pilot operating it.

The assignment was given by Teateret Vårt, which wanted to have a levitating book in their play. Therefore, the thesis also examines design conflicts when the drone needs to be disguised as a book for Teateret Vårt. The main conflicts encountered involve the need for air vents for the drone to fly and the additional weight added to conceal the drone.

Two indoor localization technologies are used to track the drone's position. Pozyx's UWB (Ultra-Wideband) utilizes long radio waves for position measurement. The other technology used for tracking the drone's position is distributed by Marvelmind. Marvelmind employs ultrasonic sound to determine the object's location. Both technologies use triangulation from different reference points to track a moving mobile sensor. Each of these technologies has its own advantages and disadvantages.

For drone control, two scripts are presented, each with different specifications. The first script is fast and efficient but lacks precision. The second script is slower but offers higher precision.

The use of drones in theater has potential, but it requires both programming, design, and hardware to enable the drone to fly reliably and safely.

During the project, a YouTube video was created. This video aims to showcase the progress of the project so far. The final result is not included but provides insight into the project's development. Link: https://youtu.be/QBF4H6GgW6Y

Acknowledgment

First, I would thank Bjørnar Teigen, Jonas Fuglseth and the others at Teateret Vårt for giving me this unique project.

Secondly, a big thank you to PlayOn! for taking me to Krakow and letting me in on their Digital Campus 2023 workshop. Giving me valuable insight into the immersion in art and technology.

Also a big thank you, Lars Ivar Hatledal, for valuable critique and feedback.

Finally, I want to thank Erlend Coates, who supervised this project. Giving guidance, advice, tips, and encouragement. This project would not be possible without him.

Table of Contents

List of Figures

Terminology

UAV - Unmanned aired vehicle

Quadcopter - A helicopter having 4 rotors

6DOF- 6 Dimensions of freedom

RTLS - Realtime Location System

TOA - Time of arrival, total time for a signal to emitt to recive back again

UWB - Ultra wideband, Radio waves between 3.1 and 10.6 GHz.

GPS - Global Positioning System

LPS - Local Positioning System

Library - A collection of pre-written code available to make coding more efficient and clean.

Hotspot - A connection using WiFi, but does not give access to world wide web.

MQTT - A computer communication protocol, using publish/subscribe data from a main server, called MQTT-*broker*.

1 Introduction

This report reviews the challenge with incorporating drones in live theatre performances. The motivation for the use of drones in a theatre play is to enhance the immersion experience by introducing levitating props, creating an illusion of magic and unnatural forces.

However, the implementation does have concerns and challenges. Such as the safety for both actors and audience in the theatre, reliability, and control.

Moreover, it is a difficult balance between artistic vision and technical feasibility to have the implementation of drones satisfy both illusion and reliability.

The theatre company *Teateret Vårt* had produced a new play that was in need of a flying prop. In cooperation with NTNU, the challenge was handed over to us to try to find a possible solution for their task. Teateret Vårt was specially requesting us to use a drone.

Additionally, this report also reviews the different technologies feasible for localization, different methods to control drones, and lastly, different challenges and issues with disguising the drone into the prop.

1.1 Background for the task

Teateret Vårt is Molde municipality's largest theatre. They have created a play based on the novel Hunger (original title Sult) by the Norwegian author Knut Hamsun. The novel tells a story about a struggling writer who becomes more and more consumed by his thoughts of inadequacy and failure. Hunger is widely regarded as a classic of Norwegian literature. Hamsun writes about self-doubt, creativity, and human psyche in a haunting and provoking way[20]. Teateret Vårt wanted to visualize the protagonist losing his mentality with a flying notebook on stage. Using a drone, the prop would physically fly on the stage. The finished product wanted by Teateret Vårt was three different drone interactions on stage. The interactions with the drone were changed somewhat throughout the project, but these changes did not create any conflicts for the thesis.

The three interactions were somewhat possible to do with an off-the-shelf drone available in hobby stores, the Tello drone that is used in this project was equipped with enough features to do this.

The Tello could be controlled with a mobile phone, this was somewhat enough for Teateret Vårt's usage, but then the Tello was dependent on good lighting. So for the project to be more applicable for a bachelor thesis in automation engineering, it was decided to add an indoor GPS system to try to create a closed-loop system locating and controlling the drone by itself. The objective for this would be to have the drone act on its own. Moving between locations as programmed to (Without remote human control).

The task for the thesis is to create a system that can calculate where the drone is and control it to different coordinates on stage. To track the position of the drone, different technologies are feasable. They have the same key concept. A small transmitter sends out a beam signal, that is picked up by multiple stationary receivers (called anchors). Using the time difference of when the signal is sent to when the signal is received, its possible to estimate the transmitters position.

1.2 Goal Objective

The goal of the project can be parted into 2 parts, what Teateret Vårt wanted as a finished product, but also what was needed for it to be done so it was more applicable as a automation thesis.

What Teateret Vårt wanted was to have the drone do 3 appearances during the play; First was to have the drone take off from the main characters hands. Staying in the air where they could push the drone and it would stay in one place.

Second, the protagonist throws the book away from him, but the book never lands on the ground.

Rather it begins to fly in the air.

The third and last appearance would be the drone taking off from the ground. Finishing the play. These appearances are very straightforward and can be done with a drone bought off-the-shelf. However, it was important for Teateret Vårt to have the drone resemble a book, so the task has some design challenges as well.

Furthermore, to make it more relevant for automation engineering it was decided to add a local GPS-system for the drone. Making it possible to control the drone's position on stage independent. Another reason for the GPS system is that the Tello drone was dependent on good lighting to maintain its position. Creating a closed loop system so it stays in its position reading of the *GPS*. The possibilities for utilizing an indoor GPS-system together with the drone are quite flexible and open-ended.

1.3 Structure of Thesis.

The report is structured as follows: Chapter II presents the theory needed to understand the different technologies and concepts that have been used in the thesis. The chapter also includes background information relatable to the report. How the technology and concepts are put together are presented in Chapter III, also presenting the testing and prototyping done. Chapter IV showcases the final result of the project in general. What the final product is, and if it has managed to successfully complete the challenge given. Chapter V discusses the result, assessing the results found in the previous chapter. Also, different limitations and constraints encountered, and how these were solved. The discussion chapter also assesses what could be done differently, and how it might be solved in other ways. The Final chapter, Chapter VI, summarizes the findings of the project. The conclusion also consists of what will be done in future research. Finally, what are the possible usages for the technology that has been established.

2 Theory

The theory chapter aims to give an understanding of the different technologies that are used in this study. Both how they function and how they are intended to be used.

2.1 Drones and Quadcopters

Drones and Quadcopters are two terms often used around each other; however, quadcopters are categorized as a subcategory of drones. A definition of a drone is an aerial vehicle that is unmanned and can fly autonomously. Therefore, fixed-wing airplanes equipped with GPS able to fly autonomously can also be categorized as a drone [19]. Multicopters are drones using multiple propellers for lift, the difference between the speed of the motors creates the movement for the multicopter. A quadcopter is a multicopter that has just 4 motors, this is optimal for stabilization [4]. Quadcopters are able to move and rotate in 6 dimensions of freedom (6DOF). Forwards & backward, left & right, roll, yaw, and pitch.

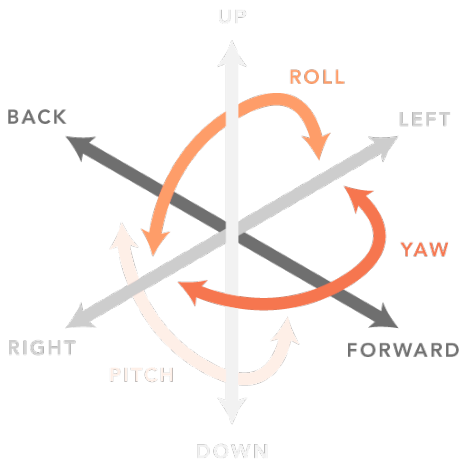


Figure 1: Illustration of 6 DOF, illustration taken from [7]

Quadcopters can change their orientation through controlling differential speeds on their motors individually. Different combinations of motor speeds change its orientation.

2.2 Tello Drone

Tello is developed by Ryze Technologies and uses a DJI flight control system. The Tello drone is lightweight and designed with the purpose to be programmable and provided with open software. The Tello drone is a quadcopter, but for further reference to avoid confusion, it will be called a drone. The drone is also equipped with tools such as a front camera and a vision positioning system. This system contains two components, a camera, and a 3D infrared module. The two components are pointing down, calculating the distance to the ground, and helps maintain the current position [18]. These properties make this the choice for the project.
The Tello drone weighs 80 grams (54 grams itself + 26 grams battery) and has a flight time of around 10 minutes. The flight time is very dependent on the weight of the drone. Therefore, it is possible to remove some parts to get more flight time and vice versa, added weight will reduce flight time [17]. The Tello drone is measured to be able to lift 70 grams of extra weight [8].

It is important to mention that the positioning system on the drone struggle when lighting is low[18]. When at the theatre scene, it will always be different lighting and it is wanted that the drone is stationary. Unaffected by the lighting. This is one of the reasons why it is implemented a indoor GPS system.

2.2.1 Tello Library

The Python library to control the drone is basic with straight-forward instructions.

start( ): - Creates a connection between Python terminal and Tello drone.

takeoff( ): - Function that instructs drone to take off the ground, going up to 70cm above ground.

land( ): - Tello drone is instructed to lower elevation and land

get_tof( ): - Get the distance to ground in millimeters

forward(centimeters): - Function takes an input value and drives the drone forward that value in centimeters.

backwards(centimeters): - Drives drone backwards the amount of input value in centimeters.

left(centimerers): & right(centimeters): - Goes left or right in centimeters with inputted value.

up(centimerers): & down(centimeters): - Goes up or down in centimeters with inputted value.

anticlockwise(degrees): & clockwise(degrees): - Rotates the drone in yaw axis using the inputted degrees. Limited between 0 and 360 degrees.

2.3 Localization Systems

GPS is a technology used to determine the position of devices globally, using satellites traveling in orbit around the globe. GPS works by using satellites that orbit the Earth. The satellites sends a continuous signal containing their current position and the time when the signal were sent.

GPS-receivers can with the received timestamp of the known position of the satellite calculate the time it took for the signal to travel from the satellite to the receiver.

Using both the satellites position and the signal's time delay, it can then calculate its relative position from the satellite. Comparing the position from a minimum of 4 different satellites the receiver calculates its position.

For a reliable continuously tracked location there are 24 operational satellites, and at any time, at least 4 are within range. 4 is the minimum for providing the current location [16]. On a global scale, the satellites are the anchors used as points of reference. The GPS tracker is the mobile node.

Furthermore, GPS is not sufficient at locating objects that are indoors, as walls and roofs disturb the signal. GPS lacks the precision needed to locate an object in a room. Additionally, GPS technology is limited in its ability to estimate precise altitudes.

In contrast, there are alternative methods to localize objects with greater accuracy in a closed area. Since the location system is not on a global scale, it is called LPS (Local Positioning System) [22]. LPS uses different technology to estimate a mobile nodes position but the principle of locating the mobile node using triangulation (see chapter 2.5) is the same.

2.4 UWB

Ultra-WideBand is a technology that uses wireless communication and applies a broad frequency spectrum to transmit and receive data over short distances. By diverging the signal over a broader range of frequencies (typically several gigahertz) the spectrum makes it possible for high data transfer rates and precise localization capabilities. Sending signals in a very short duration to transmit data. The pulses have low power cost and spread across a wide range of frequencies, which allows for lot of transmitted data[5]. Measuring the Time of flight (ToF) of the signal of the UWB pulse between devices it can estimate the distance between them [5]. Reading of the distances between multiple devices it can then create an estimate of a mobile device. Estimating the position using different perspectives is called triangulation.

2.5 Triangulation

Triangulation is a measuring technique, using multiple points of reference to estimate a new position or distance. For the positioning system for the drone, it will be used stationary nodes (called anchors). The stationary nodes know their position relative to each other. The stationary nodes can then send a signal out to the mobile node (the node that is tracked). The mobile node then sends the signal back to the anchors again. Using the TOT and TOA (time of transmission and arrival) and knowing the speed of the signal. Computers can estimate the distance from each anchor to the mobile node. And then calculate an estimation of where the mobile node is [21]. Using one anchor, it is only possible to calculate the distance between the anchor and the mobile node. However, there is not any direction. The signal is sent in every direction around the anchor, the returning signal creates a sphere around the anchor where the possible position is [21].

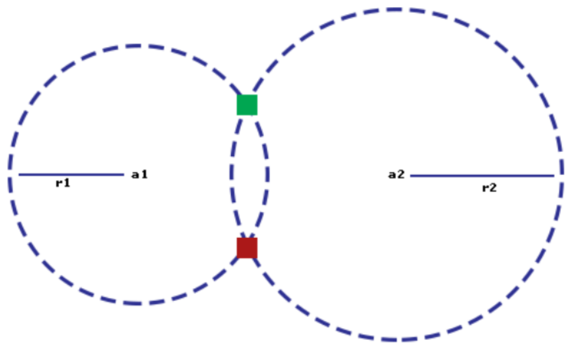


Figure 2: Using two Anchors, a1 & a2 and using the distances of r1 & r2, the possible location of the beacon is both in the green and red area.

Using two anchors to track a mobile node, however, there can be then two possible solutions for where the mobile node is, as illustrated in figure 2. If the anchors are at the end of a room, a wall can then eliminate the other position. 2 anchors are functional for 2-dimensional space, but a 3rd anchor will create better localization, and it can also give a sense of the height level of the mobile node.

2.6 Pozyx UWB Technology

Pozyx is a distributor of UWB RTLS system, they have developed an open-source kit with hardware that they develop. Creator One is a kit with both Arduino and Raspberry Pi libraries.

2.6.1 The Creator One Kit

Pozyx provides a kit for new creative development of indoor GPS systems. The kit includes 6 developer tags (mobile nodes) and 4 anchors (anchors working as reference points). One of the tags serves as a master tag that can be connected to the computer communicating with the other tags. The 6 tags make it possible to track 6 different objects within the confined space. Extra technical equipment that is supplied is 5 battery packs for the tags and anchors and 4 power adapters [13].

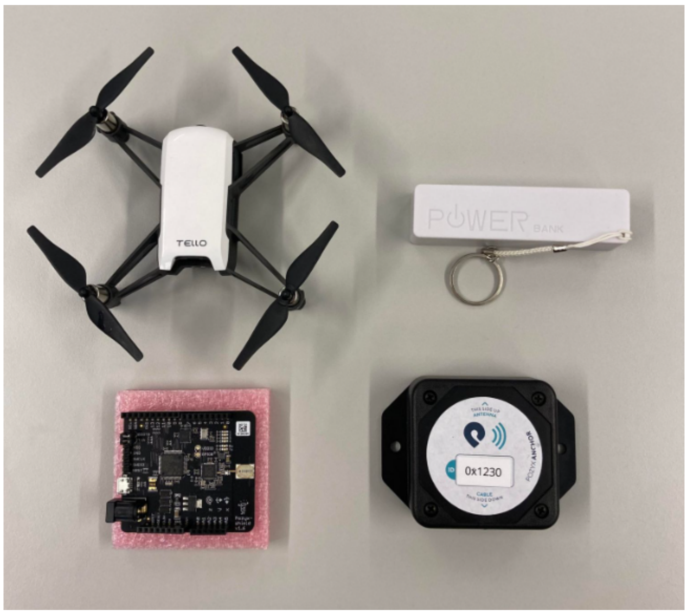


Figure 3: Creator One kit + Tello Drone, top left is Tello drone, Battery pack, mobile beacon, and anchor.

If it was to be used to track the drone, the drone would then need to carry both the power bank and the mobile beacon. The power bank is weighing 61 grams, and the mobile beacon weighs 17 grams.

2.6.2 Companion Software

Pozyx does distribute software with the hardware, the software gives real-time location visualization. In this software it is also possible to adjust settings and calibrate the system. The software available as a standalone software or available on the web in a browser. From this software its possible to collect the location data using MQTT.

2.7 Marvelmind Indoor Navigation System

Marvelmind has developed tracking technology with the same technique as UWB RTLS, the main difference however is the use of sound instead of radio waves. The speed of sound and radio waves are very different, but the same method still applies. Marvelmind supply software for their components, making it more straight forward to set up and read data of it. Marvelmind Starter set NIA-SmallDrone is a tracking kit designed for small drones for indoor tracking. The set contains 4 Super-Beacons, 1 Mini-TX Beacon and 1 Modem [9].

2.7.1 Super-Beacons

The Super-Beacon can be used as a double feature, with both being a stationary anchor but also a mobile node. This is possible because the Super-Beacon is capable of transmitting and receiving ultrasonic sound.

The Super-Beacons can have a range of 50 meters between them, but Marvelmind recommends up to 30 meters. The super-Beacon measures at 55x55x65mm (with antenna) and weighs 59g (including 1000 mAh battery) [11].

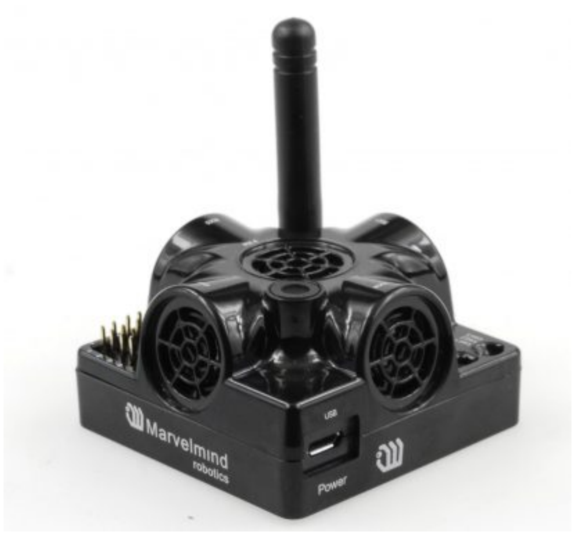
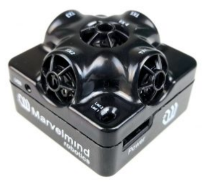


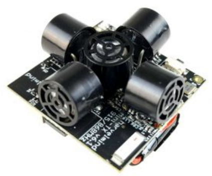
Figure 4: Marvelmind Super Beacon, picture taken from Marvelmind.com [11]

2.7.2 Mini-TX Beacon

The mini beacon is as suggested a smaller version of the other tracking beacon, ideal to attach on a drone because it weighs less. The biggest differences are that the Mini beacon is a TX, meaning it can transmit ultrasound but cannot receive it. Common Beacons can receive and transmit ultrasound. The Mini-TX is 47x42x15mm and weighs 25 grams. Comparing size to Super-Beacon, which is 55x55x33mm (not counting the antenna) and 62 grams. The battery of the Mini-TX is a 250mAh, but since it also only has TX function, the battery time lasts longer [10] . (The Mini TX will be referred as the *mobile beacon* or *beacon number 7*)



(a) Picture of the Marvelmind Mini TX.



(b) Marvelmind Mini TX without its casing

Figure 5: The mini TX can remove its casing to make it smaller. Pictures taken from Marvelmind.com [10]

2.8 Computer Assisted Design

Computer assisted design, CAD for short, is computer software that assists in designing models. The designs are then created with great precision and accurate measurements. The software that has been used to create models is called Autodesk Fusion 360, a versatile tool for model making. Fusion 360 also has tools to optimize the 3D printing of models.

In addition, Blender was used to create computer models. Blender is a computer program that mainly focuses on character modeling and animation, but it was chosen for its familiarity and efficiency. Ultimately, Blender is not an optimal option for creating designs that crave high precision and measurements.

2.9 Additive Manufacturing (3D-Printing)

Parts needed for the product, such as the drone's disguise as a book and holder for the mobile sensor were made with the usage of additive manufacturing. There are different technologies for additive manufacturing, however, final products were made using material extrusion (MEX) 3D printers. The 3D printers used, are called Prusa Mini+. The Prusa printers provide an opportunity to prototype parts and small models. The Mini+ Models have a printing volume of 18cm^3. Which is sufficient for this project [14].

2.9.1 Material for Printing

The models are printed in PLA (Polyactic acid), a plastic polymer created from renewable sources. These sources can be fermented plant starches such as corn or sugarcane. The material is strong and stiff enough for the project.

2.9.2 Prusa Slicer and Slicing

Software used to make instructions to the printer. Here it's also decided what size and orientation the model should be printed. Different orientations need different supports and can demand more time. Prusa Slicer is the software to create the instructions for the printers [15].

3 Methodology and Materials

The methodology and materials chapter goes deeper into the approach of the task, and what was done from research at the beginning of the project to the final product.

3.1 Project Organisation

3.1.1 Supervisor Meetings & Meetings with Teateret Vårt

The student and supervisor had a meeting usually every Wednesday. At the beginning of the project, it was mostly discussed how to approach the task given by Teateret Vårt. As there are multiple ways to locate the drone, a comparison of technologies was made to look at what is needed for the task and was is possible to do. While doing research, it would also be discussed what kind of drone would be accessible. An idea was to build a drone with the needed gadgets from scratch, however, this would need a lot more research and time.

It was also discussed design ideas for the book, such as how the drone could be hidden, and safety measures for both actor and audience.

After finding hardware that could be used for locating the drone, it was ordered online. The drone for the project was already in inventory, it was the correct size and was equipped with the needed tools.

3.1.2 Meetings with Teateret Vårt

There were 3 meetings with Teateret Vårt. The first meeting was with them presenting their objective of their using the drone. Also telling the setup of the play, the scene, and where the audience would be sitting. The second meeting was at Teateret Vårt, where we got to see the scene setup and perceive how the play would unfold. Here we showed our drone and the indoor GPS system that we ordered. It was tested to see how the drone noise would play out with music. Here it was learned how much the Tello drone struggles to keep its position in air with the different lighting conditions that the theatre use on stage.

The third meeting was also at Teateret Vårt, this time the drone was implemented in the play, tested with an audience.

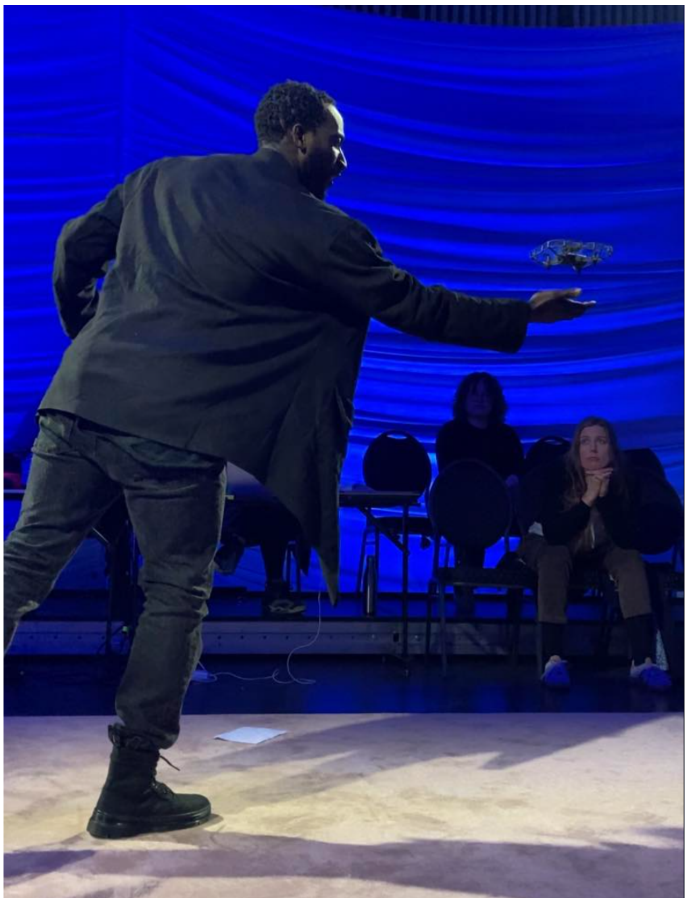


Figure 6: Testing the drone in the play, the drone is only tested with its safety cage. (Note the amount of light on the ground.)

3.2 Research for hardware

First phase of approaching the task was to do research for how to localize the drone. There were multiple systems using different methods. There where different suppliers that was considered. Comparing what the different suppliers where offering, as of software, hardware and price points. Evaluation of the different techs were compared in a table:

3.3 Setup and testing hardware and software

When the positioning systems arrived, the first approach was setting the systems up and creating a connection between the anchors, the beacons, and the laptop to read the position. The main reason for testing was to check if everything worked, hardware and software-wise.

First the Tello drone, then both the location systems, POZYX and Marvelmind. There were different theories for optimal setup with the location systems to get the best results. One theory is to have all in the same plane, at a level z value in the room. Second theory is to have two diagonal anchors placed lower on the wall, and the two other higher up. This would give the anchors better cross-reference when calculating distances.

3.3.1 Tello Drone

To control the Tello Drone, a library is available. Equipped with easy scripts to decide movement and orientation. Also, some scripts to get status information. Functions mentioned are listed in chapter2.2.1. The Tello Drone is connected to a computer with Wi-Fi, a *hotspot* the drone sets up itself. With the connection to the computer, it is then given instructions with Python code. A complimentary Python library is given with the Tello drone for basic maneuvers. Such as rotating in the yaw axis to move in different directions. The Tello drone could also be controlled with an app on a smartphone, the smartphone connects via the *hotspot* by the drone. When the connection is successful, the drone was tested with simple lines of code. Such as takeoff and move 100 cm forward. These are scripts already supplied from Tellos code library.

3.3.2 Marvelmind

Testing the Marvelmind location system was done by first uploading firmware to both the anchors and the mobile node provided by Marvelmind. Then connect them to the modem. In the first effort just to try the tracking of the mobile node. The anchors were set up with no determined distance between the anchors, but when they showed up in the software and the ranges seemed to be accurate. However, in the testing, the correctness of ranges and the readout of data is not so important. It will be tuned later. Most important is to check that there is no big errors. Also important for testing is to see if the mobile node is located and the estimated location is somewhat accurate. Which it is.

When setting up a permanent environment, the beacons are mounted on each side of the classroom using Velcro. All beacons are placed 2.5 meters high, so that the signal has as the minimal objects disturbing the line of sight. The beacons are each given a number to mark each of them.

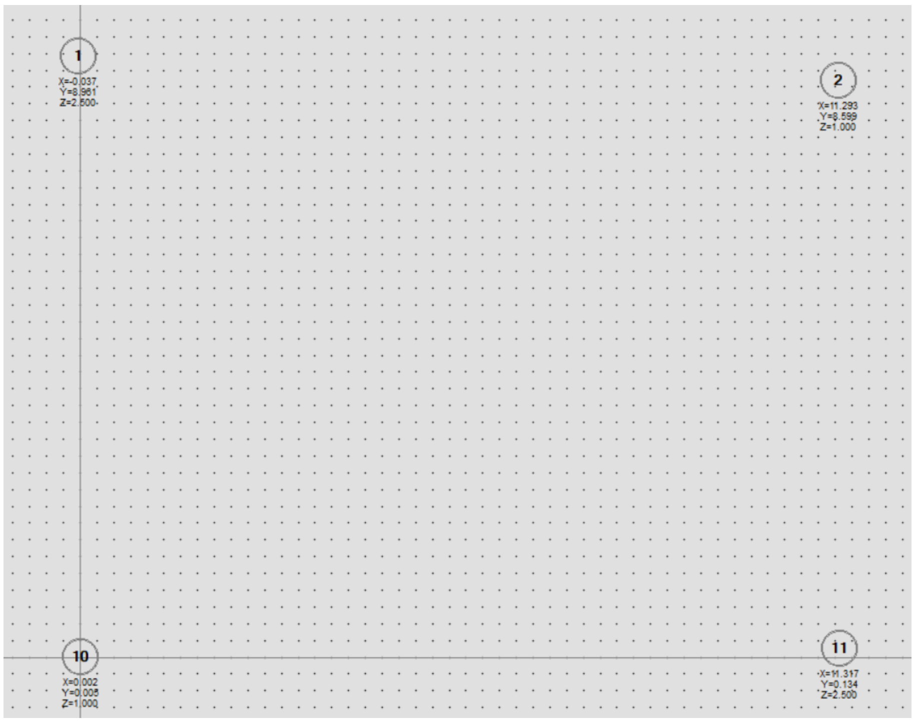


Figure 7: This is how the beacons are oriented in the room, ignore x, y and z values for now. Only note that beacon 10 is in origo (center) of the map. This is to have positive coordinate values in the future.

When measuring the actual distance between the beacons. The ground truth distances between the beacons are measured below. It is not measured diagonally. Measured distances are 1

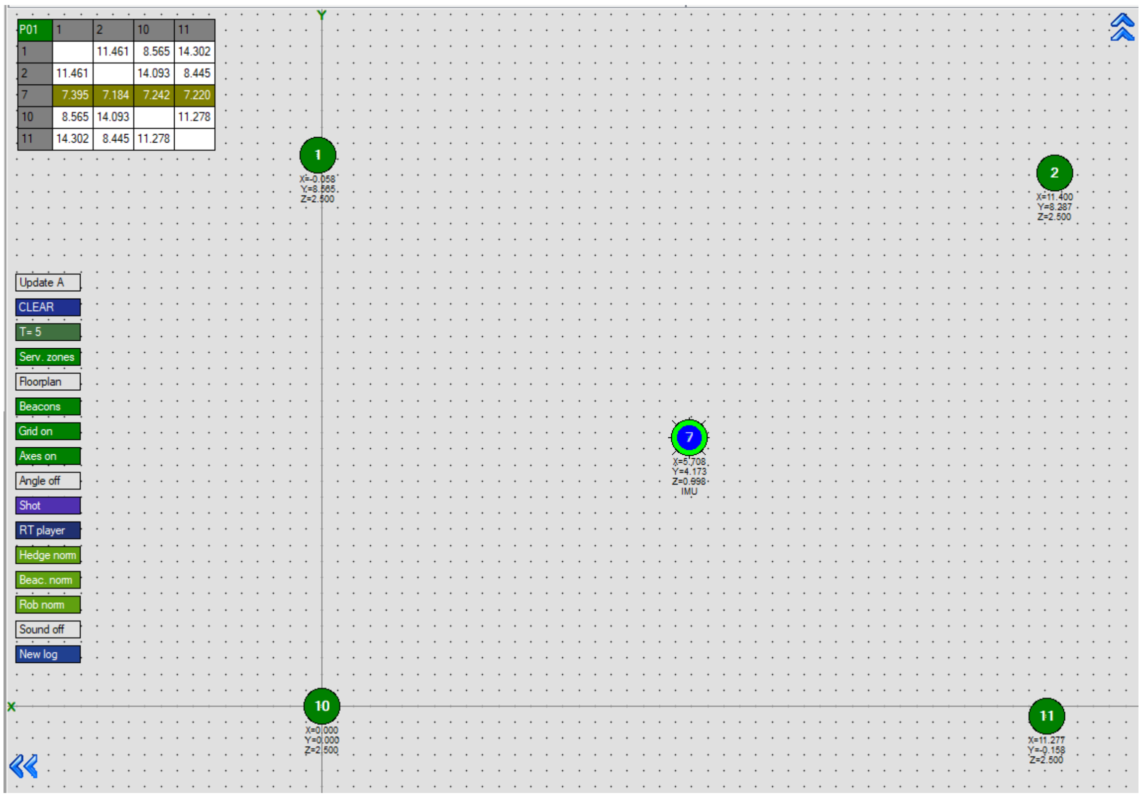
Mind the distance between 1 to 10 and 2 to 11 is not exact same length, neither 1 to 2 and 10 to 11 even when 1 & 10 or 2 & 11 are mounted on the same wall.

Software supplied by Marvelmind is used to set up the indoor location system. The beacons send

	Beacon 1	Beacon 2	Beacon 10	Beacon 11
Beacon 1	-	11.40m	8.60m	-
Beacon 2	11.40m	-	-	8.43m
Beacon 10	8.60m	-	-	11.30m
Beacon 11	-	8.43m	11.30m	-

Table 1: Ground Truth Matrix (Marvelmind)

out a signal to each other and then estimate their distances based on the time it takes for the signal to return.



P01	1	2	10	11
1		11.461	8.565	14.302
2	11.461		14.093	8.445
7	7.395	7.184	7.242	7.220
10	8.565	14.093		11.278
11	14.302	8.445	11.278	

Figure 8: The Beacons are running, listening to the the mobile node nr 7 (in center). The coordinates are adjusted, note that the Z value is defined and not calculated by Marvelmind.

In figure 8 the distances of the beacons are in the top left. These are the readings of the Marvelmind data.

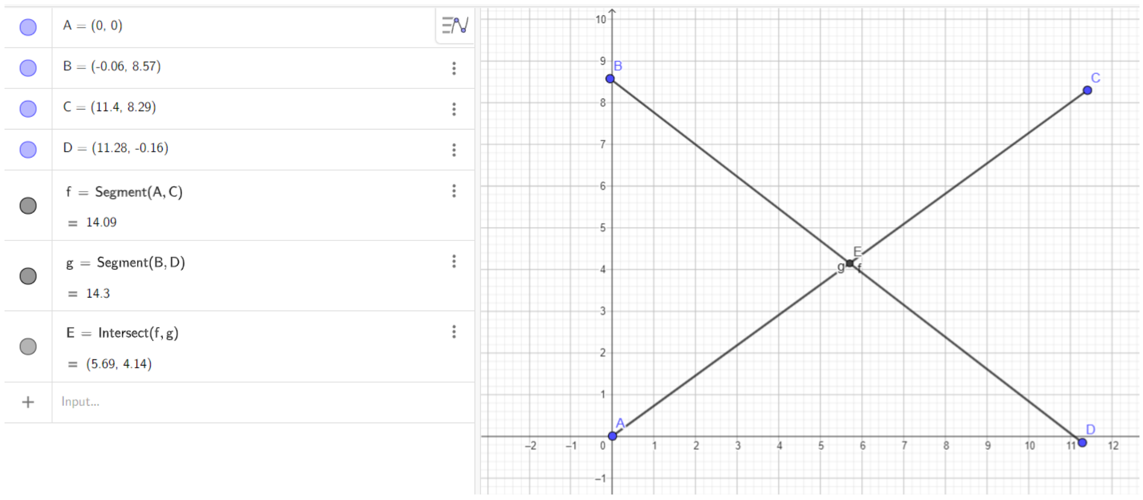
	Beacon 1	Beacon 2	Beacon 10	Beacon 11
Beacon 1	-	11.461m	8.565m	14.302m
Beacon 2	11.461m	-	14.093m	8.445m
Beacon 10	8.565m	14.093m	-	11.278m
Beacon 11	14.302m	8.445m	11.278m	-

Table 2: Marvelmind Distance Matrix

The values in matrix 2 is the measured distances done by the Marvelmind software. These values change somewhat when in *search mode* actively measuring distances. Compared to the first matrix (Matrix 1), the diagonal distances are measured in matrix 2.

When anchors are done calculating their distances between themself, the map *freezes* their position and begins to actively find the mobile beacon in the room. Placing the mobile beacon as center as possible in the room so each anchor can get as equally strong signal as possible. Thereafter reading the position of the mobile beacon from the software and from ground truth, comparing the results give an estimation of reliability and precision of the local GPS.

Finding center of the the beacons is done by putting the coordinates of the anchors in geogebra and see where the diagonal line cross:



```
A = (0, 0)
B = (-0.06, 8.57)
C = (11.4, 8.29)
D = (11.28, -0.16)
f = Segment(A, C)
= 14.09
g = Segment(B, D)
= 14.3
E = Intersect(f, g)
= (5.69, 4.14)
Input...
```

Figure 9: Measuring X and Y of coordinate E, we place the Marvelmind mobile beacon 5.7 meter from the left wall, and 4.14 meter from the bottom wall

3.3.3 POZYX, Creator one kit, First setup

Testing the Pozyx location system was done similarly to Marvelmind, here there was an issue with the software that reads all the data to work. Luckily Pozyx also had the software available online in the web browser, the data could then be extracted out as with MQTT to Python. Here again, testing it is mostly for checking if everything is working, tuning will be done later. The first setup was done in one of the classrooms. Figure 10 shows the first setup. When setting up, each anchor is given a reference coordinate of where it is, then when starting to scan where they are, they have a greater understanding of their relative position from each other. Note that Anchor *0x1233* is connected to my laptop. This communicates with the other anchors. Optimally this would be in a corner of the room.

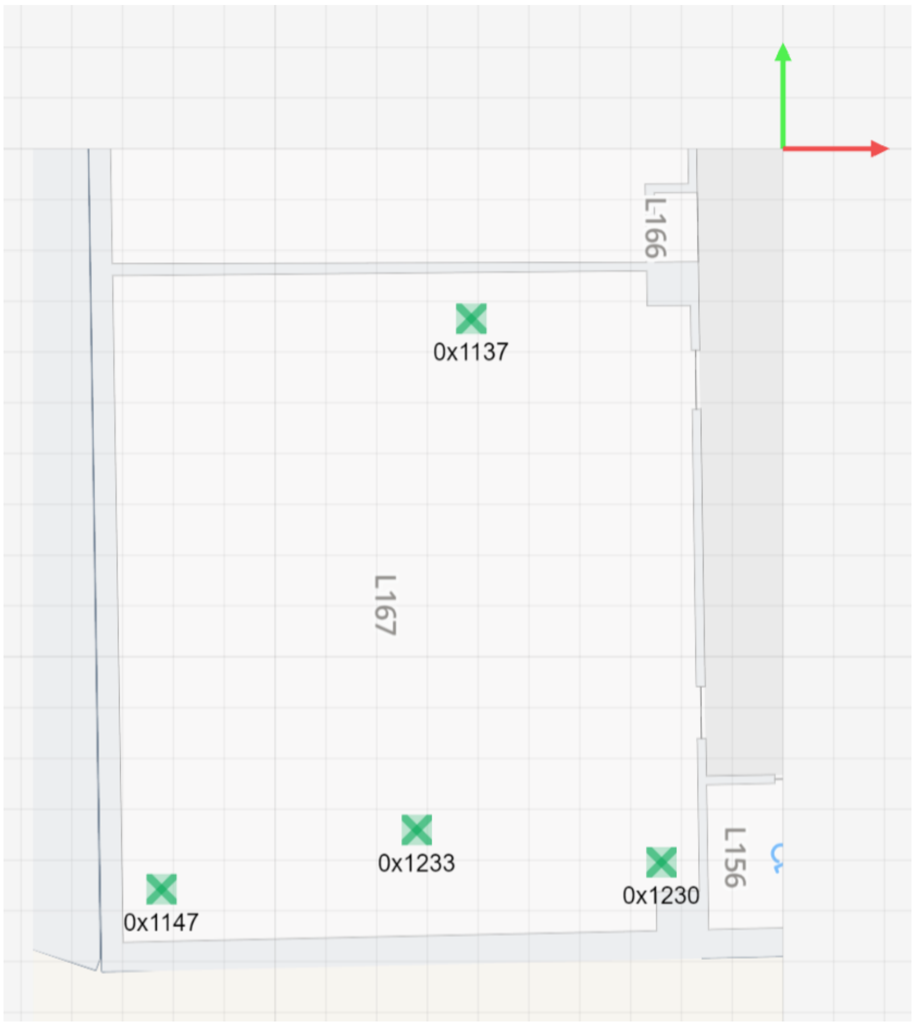
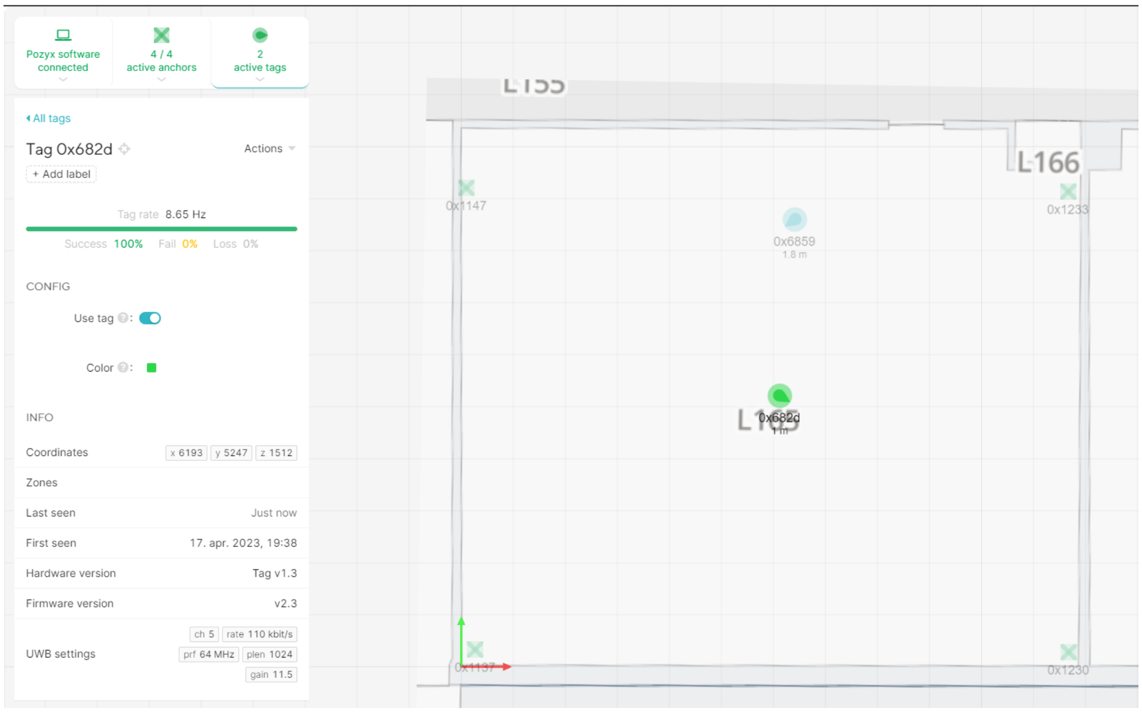


Figure 10: First setup of the Pozyx test, figure is a screen capture of the software. Note that the background is a picture of the classroom where this was tested.

In figure 10 , the mobile nodes are not visible.

3.3.4 Measuring Pozyx with setup

The Pozyx beacons were set up in classroom L165 for measuring accuracy.



Tag 0x682d

Tag rate 8.65 Hz

Success 100% Fail 0% Loss 0%

CONFIG

Use tag: on

Color: 🟩

INFO

Coordinates	x 6193 y 5247 z 1512
Zones	
Last seen	Just now
First seen	17. apr. 2023, 19:38
Hardware version	Tag v1.3
Firmware version	v2.3
UWB settings	ch 5, rate 110 kbit/s, prf 64 MHz, plen 1024, gain 11.5

Figure 11: Setup at room L165. Stationary beacons are placed in the corner of the room, two mobile beacons are visible at the map.

Figure 11 shows the setup in room L165. Similar to the Marvelmind setup, each beacon is placed in the corner of the room. In contrast, there are two visible mobile beacons. Beacon 0x6859 is the beacon connected to the laptop, working as the communicator between all beacons. The measured beacon in center of the room is labeled 0x682d.

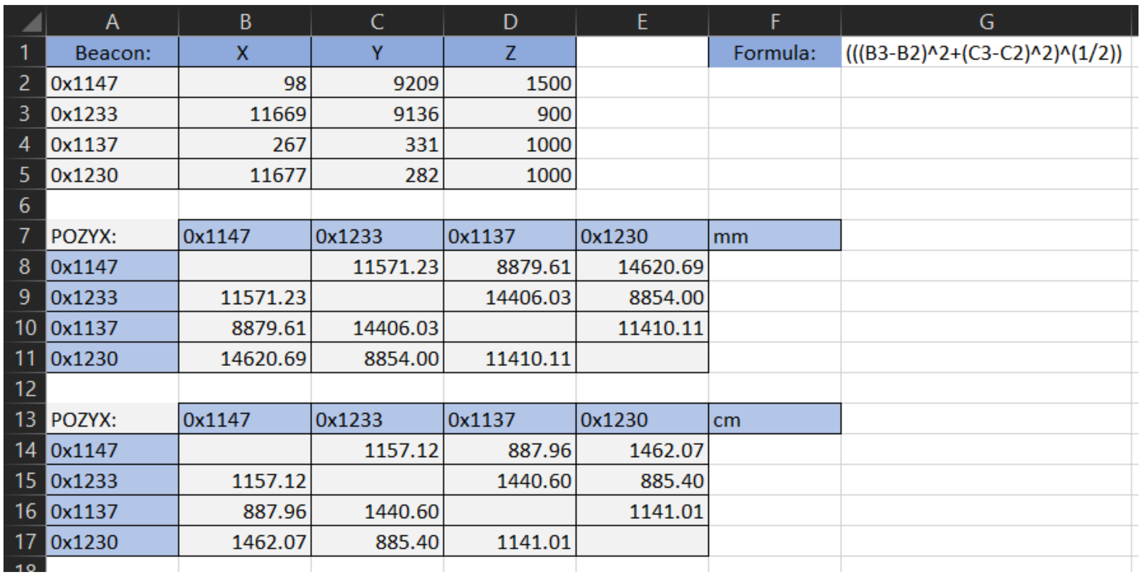
When measuring the distances using a tape measure between the beacons, the distances are shown in matrix 3:

	0x1147	0x1233	0x1137	0x1230
0x1147	-	1114mm	909mm	-
0x1233	1114mm	-	-	884mm
0x1137	909mm	-	-	1106mm
0x1230	-	884mm	1106mm	-

Table 3: Ground Truth Matrix (Pozyx)

Again, the diagonal distances are not measured.

For coordinates, values are given in millimeters. But Pozyx does not give the distances between the beacons and nodes. So reading coordinates of each beacon, it is then possible to find the distances between them.



	A	B	C	D	E	F	G
1	Beacon:	X	Y	Z		Formula:	(((B3-B2)^2+(C3-C2)^2)^(1/2))
2	0x1147	98	9209	1500			
3	0x1233	11669	9136	900			
4	0x1137	267	331	1000			
5	0x1230	11677	282	1000			
6							
7	POZYX:	0x1147	0x1233	0x1137	0x1230	mm	
8	0x1147		11571.23	8879.61	14620.69		
9	0x1233	11571.23		14406.03	8854.00		
10	0x1137	8879.61	14406.03		11410.11		
11	0x1230	14620.69	8854.00	11410.11			
12							
13	POZYX:	0x1147	0x1233	0x1137	0x1230	cm	
14	0x1147		1157.12	887.96	1462.07		
15	0x1233	1157.12		1440.60	885.40		
16	0x1137	887.96	1440.60		1141.01		
17	0x1230	1462.07	885.40	1141.01			

Figure 12: Reading the coordinates of each stationary beacon we can calculate distances between them.

First table in figure 12 shows the coordinates of each beacon measured by Pozyx's software. The Z value added externally. Using these coordinates, the distances are calculated between the anchors. Second table is the distances in millimeters, and third is in centimeters.

Here again, the mobile beacon is placed center of the room. Using the coordinates from Pozyx' software center is then found.

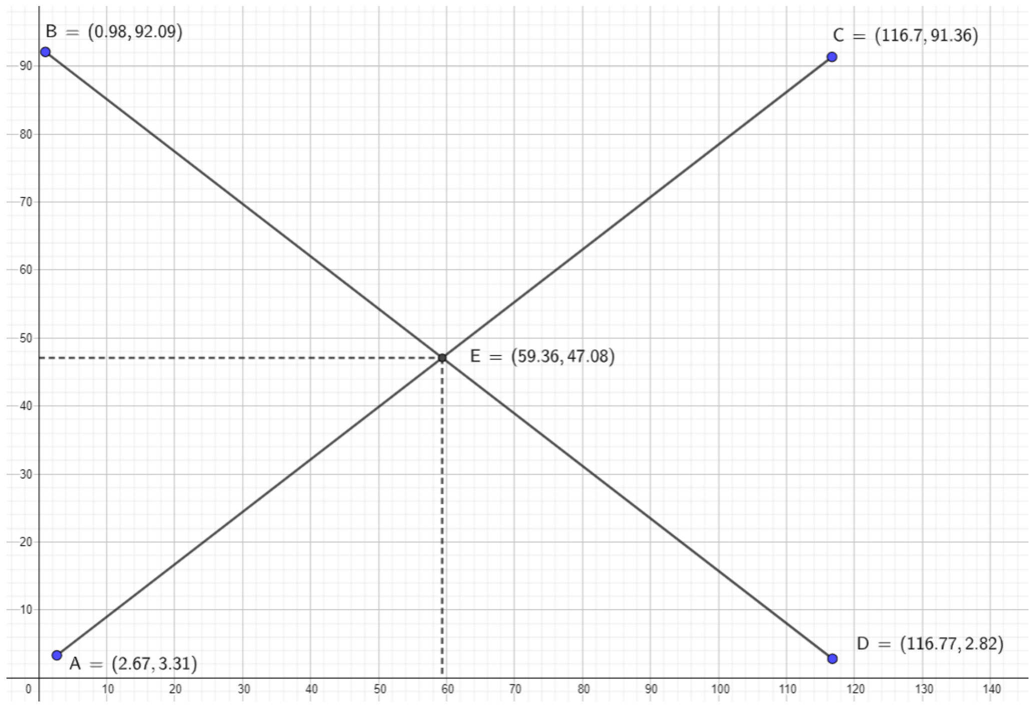


Figure 13: Finding center of the Pozyx Anchors.

Figure 13 shows where the mobile beacon should be placed to be center of the anchors. (It is scaled to a tenth so measured it is 593.6cm and 470.8cm)

3.4 Implementing Marvelmind localization with Tello drone

3.4.1 Creating a mount for the Marvelmind to the Tello

To mount the Marvelmind mobile node to the Tello drone there was a main idea that it would sit 45 degrees rotated between the propellers, however, it would be a limited area for it. This place is optimal for the drone because it would would evenly put weight in the center, and not create imbalance on the motors.

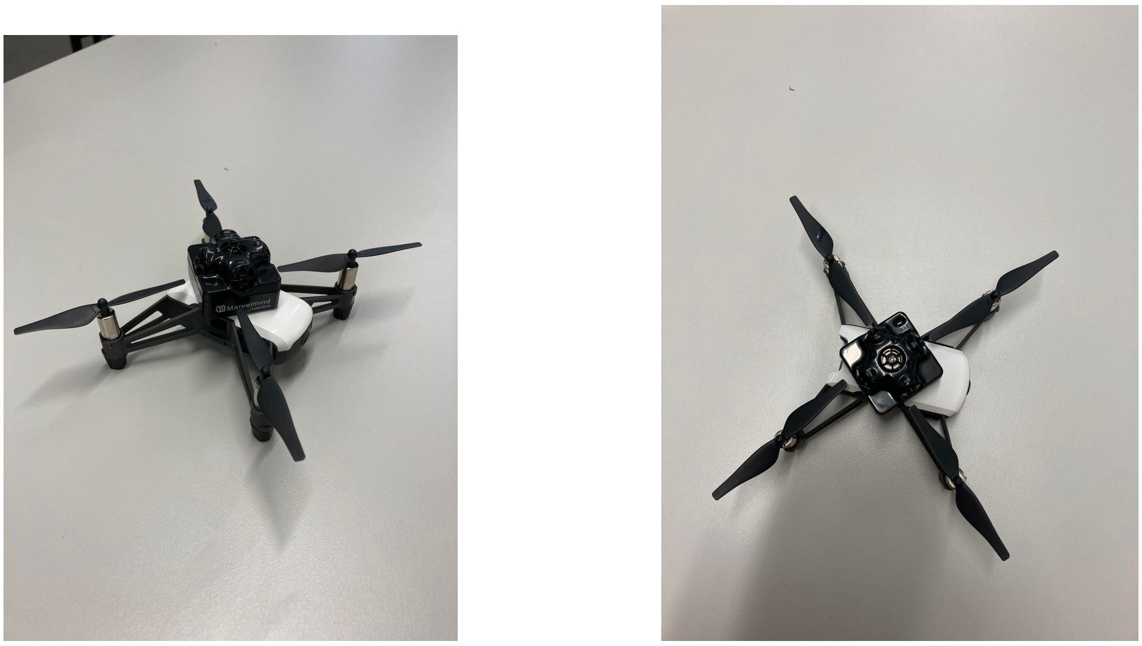


Figure 14: Marvelmind Mini TX on the Tello drone

Figure 14 above shows the ideal position for where the mobile node should be mounted. To mount the sensor, there was needed a container to hold it secure. The idea is to make a clip-on mechanism that grasp onto the Tello drone. There is a notch (see figure15) that is possible to use for the clipping. This notch is symmetric to both sides of the Tello so a 3D printed part could then slightly bend over the top. Then the sensor could be mounted to the grasp. A mount was created using CAD software and then 3D printed.

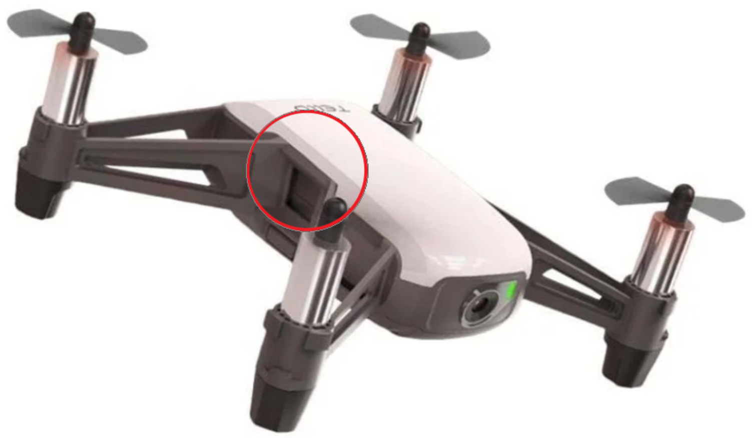


Figure 15: Possible latch point for the mount of sensor

With the latch point on the Tello, a 3 piece CAD model was created (figure 16). The bottom piece is the part that would clip on the Tello, sliding though the compartment holding the sensor. Top piece comes above to hold the sensor in place.

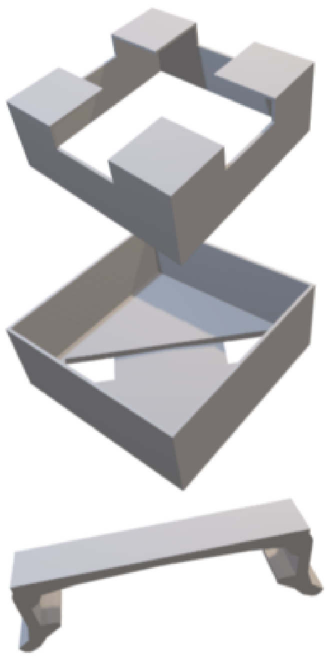


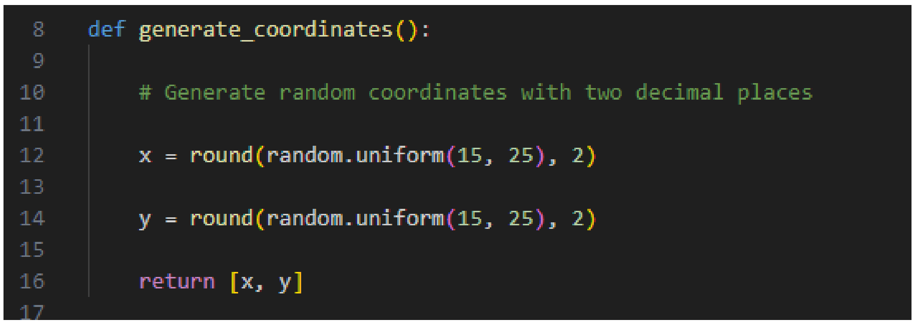
Figure 16: Computer model of the container to the Marvelmind node mounting to the drone.

3.5 Software for Tello drone

Combining the libraries from both Tello and Marvelmind, it can be used to create a script controlling the drone from its location to go to a determined position.

3.5.1 Mocking the Marvelmind GPS

Mocking refers to creating code that gives an output resembling real data, this is useful to edit the data without having to connect to hardware. First it was created a GPS mock that gives a random x and y value to further create the commands for the drone. In the mocking software its made so that it takes two input variables (x and y coordinates) that is written in the terminal. Then the code creates two random x and y coordinates. The random coordinates are supposed to be the read position of the drone, and the coordinates written in the terminal is the coordinates the drone would go to. Using these two coordinates the code then creates 3 values that is the *distance* between them, the *angle* and what *rotation* the drone should do. A function to random generate x and y- coordinates between the values of 15 - 25. For readability they are rounded to 2 decimals.

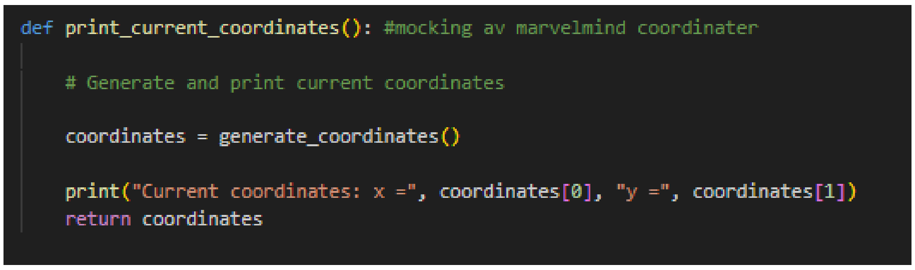


```python
def generate_coordinates():

    # Generate random coordinates with two decimal places

    x = round(random.uniform(15, 25), 2)

    y = round(random.uniform(15, 25), 2)

    return [x, y]
```

Figure 17: Function that generates random coordinates, rounded to two decimals

Figure 17 above shows the function code that creates random coordinates and mocks the Marvelmind code, the reason for the mocking is so it would be easier to create the script that calculates the orientation, rotation and orientation values without having to connect the Marvelmind setup each time.

Next function generates coordinates and prints them in the terminal. Shown in figure 18 below.



```python
def print_current_coordinates(): #mocking av marvelmind coordinater

    # Generate and print current coordinates

    coordinates = generate_coordinates()

    print("Current coordinates: x =", coordinates[0], "y =", coordinates[1])
    return coordinates
```

Figure 18: Function that generate coordinates and prints them in terminal

Last function is the code that calculates the angle, distance and orientation between the two coordinates. It is expected that *my_pos* and *wanted_pos* is coordinates with both x and y -values, therefore it declares each of the x and y values out of both coordinates. Following code shown in figure 21 using Euclidean distance formula to find the distance of difference in the x and y coordinates, but also the angle.

$$c^2 = dx^2 + dy^2$$

$$c = \sqrt{dx^2 + dy^2}$$

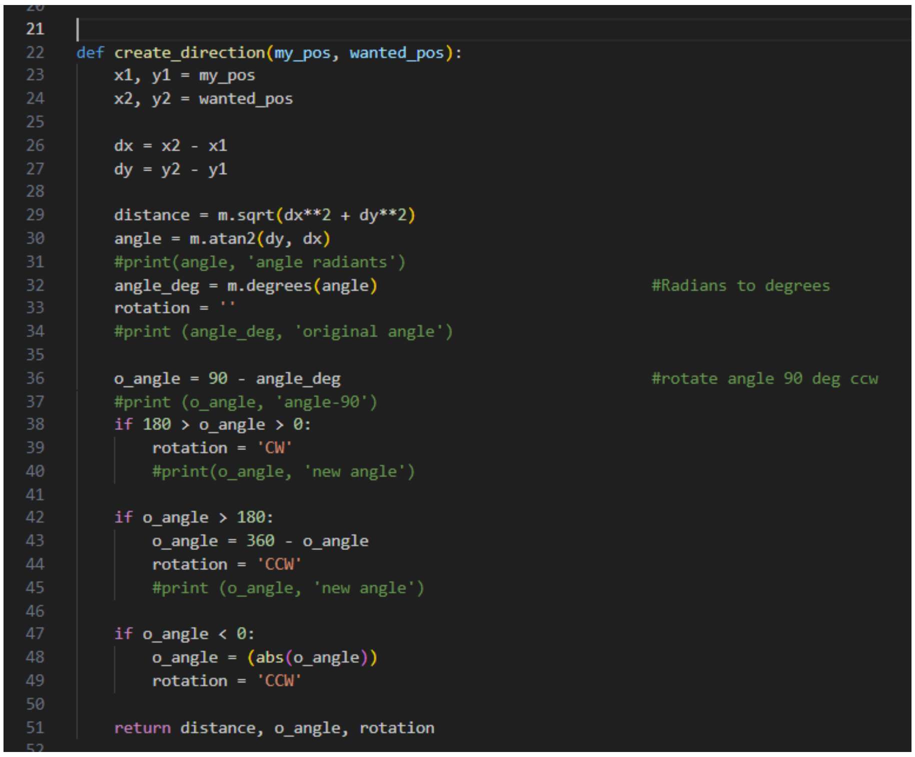
Figure 19: Euclidean Formula for length of hypotenuse,
dx and dy refers to the difference in coordinates; the length of the adjacent and opposite leg

Finding the angle of between the lengths dy and dx is used with arctan of the adjacent and opposite leg. The angle θ between the adjacent leg and the hypotenuse can be found using the arc tangent function:

$$\theta = \arctan\left(\frac{dy}{dx}\right)$$

Figure 20: Formula for angle θ, note that this value is given in radians.

However, calculating the orientation of the rotation was more challenging. The reason to find the orientation if it is clockwise or counterclockwise rotation is because the code to steer the Tello drone contains commands to steer it clockwise or counterclockwise (even though the Tello code library calls it anticlockwise). Knowing the orientation the drone should not rotate any more than 180 degrees to point towards the next coordinate it is ordered to go to.



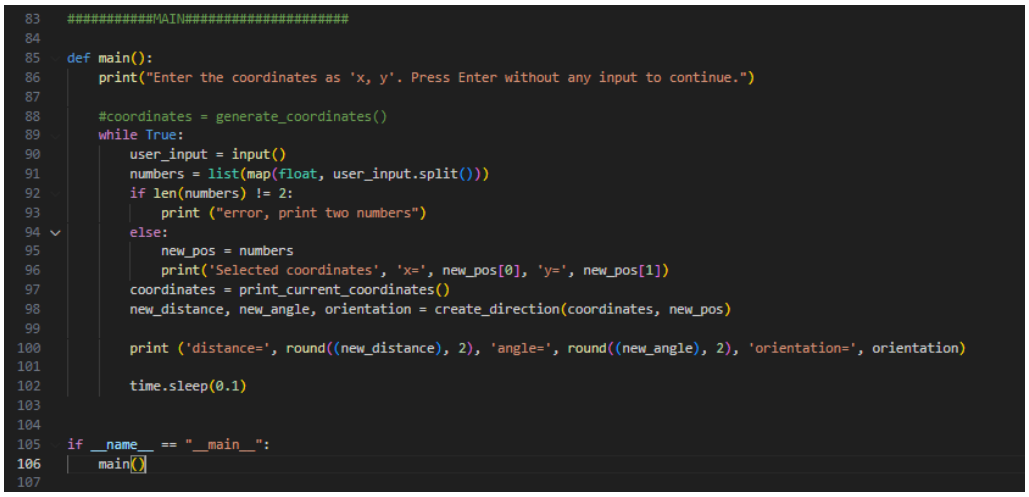
```python

def create_direction(my_pos, wanted_pos):
    x1, y1 = my_pos
    x2, y2 = wanted_pos

    dx = x2 - x1
    dy = y2 - y1

    distance = m.sqrt(dx**2 + dy**2)
    angle = m.atan2(dy, dx)
    #print(angle, 'angle radiants')
    angle_deg = m.degrees(angle)                            #Radians to degrees
    rotation = ''
    #print (angle_deg, 'original angle')

    o_angle = 90 - angle_deg                                #rotate angle 90 deg ccw
    #print (o_angle, 'angle-90')
    if 180 > o_angle > 0:
        rotation = 'CW'
        #print(o_angle, 'new angle')

    if o_angle > 180:
        o_angle = 360 - o_angle
        rotation = 'CCW'
        #print (o_angle, 'new angle')

    if o_angle < 0:
        o_angle = (abs(o_angle))
        rotation = 'CCW'

    return distance, o_angle, rotation
```

Figure 21: Function that calculates direction, rotation and orientation between two coordinates

The returned values of the function is the direct distance between the coordinates, the angle (in degrees) and if the rotation is clockwise or counter clockwise. This function should never give an angle that is above 180 degrees.

Figure 22 below shows the main part of the mocking. where the functions are put together. First it waits for an input in the terminal, its expected that the input is two coordinates split with a space. *eks: 20 20*. If input is something else, then an error is displayed.

The input numbers are put in a list, and two other coordinates are randomly created with the function from figure 17 in *line 97*.
Line 98 in figure 22, the input coordinates and random coordinates are put together to find the distance, angle and orientation. The random coordinates represents the coordinates that are read of the Marvelmind. The input coordinates represent the input coordinates that we want to go towards.

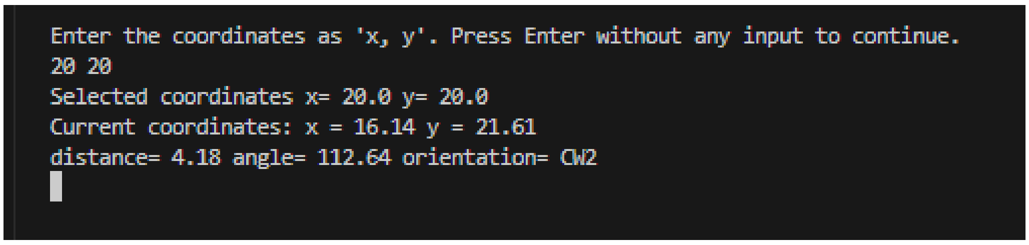


```
83   ###########MAIN####################
84
85   def main():
86       print("Enter the coordinates as 'x, y'. Press Enter without any input to continue.")
87
88       #coordinates = generate_coordinates()
89       while True:
90           user_input = input()
91           numbers = list(map(float, user_input.split()))
92           if len(numbers) != 2:
93               print ("error, print two numbers")
94           else:
95               new_pos = numbers
96               print('Selected coordinates', 'x=', new_pos[0], 'y=', new_pos[1])
97           coordinates = print_current_coordinates()
98           new_distance, new_angle, orientation = create_direction(coordinates, new_pos)
99
100          print ('distance=', round((new_distance), 2), 'angle=', round((new_angle), 2), 'orientation=', orientation)
101
102          time.sleep(0.1)
103
104
105  if __name__ == "__main__":
106      main()
107
```

Figure 22: Main code of mocking Marvelmind

3.5.2 Testing the mocking

The input given is *20 20*.



```
Enter the coordinates as 'x, y'. Press Enter without any input to continue.
20 20
Selected coordinates x= 20.0 y= 20.0
Current coordinates: x = 16.14 y = 21.61
distance= 4.18 angle= 112.64 orientation= CW2
```

Figure 23: Test of the mocking code, here the input given is *20 20*, Note: Orientation is defined as *CW2* is because code was somewhat changed, it is indifferent.

To better visualize the results of the code, figure 24 shows the results using Geogebra: *RandomCoor* is the Marvelmind mock, and *selectedCoor* is our input. Point A is to define direction. Angle α is the clockwise angle. And d is the calculated distance.

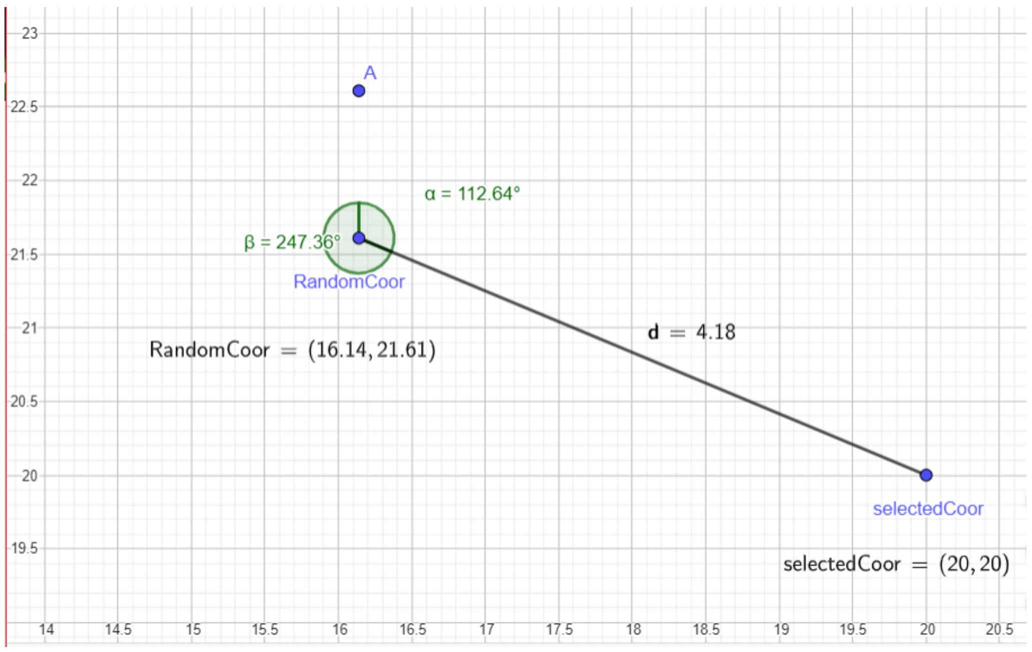
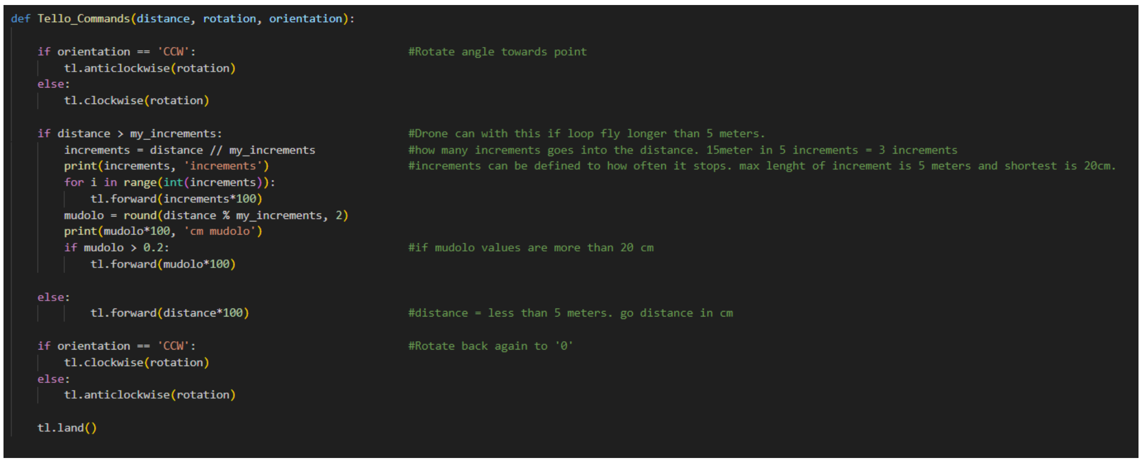


Figure 24: Visualising in Geogebra to get better understanding.

This code is the first step in enabling the Tello drone to move from one coordinate to another in a confined space. However, the drone is not able correct itself because it does not check if it is off its course. It is still tested on the drone to see if its working.

3.5.3 Controlling the drone with Marvelmind

With the function to find direction and distance, it is then created a new code to give the drone its instructions. Here its important to mention that the Tello library do not allow code that instructs the drone to fly further than 5 meters. and the following code also has a workaround this limitation.



```python
def Tello_Commands(distance, rotation, orientation):

    if orientation == 'CCW':                                #Rotate angle towards point
        tl.anticlockwise(rotation)
    else:
        tl.clockwise(rotation)

    if distance > my_increments:                            #Drone can with this if loop fly longer than 5 meters.
        increments = distance // my_increments              #how many increments goes into the distance. 15meter in 5 increments = 3 increments
        print(increments, 'increments')                     #increments can be defined to how often it stops. max lenght of increment is 5 meters and shortest is 20cm.
        for i in range(int(increments)):
            tl.forward(increments*100)
        mudolo = round(distance % my_increments, 2)
        print(mudolo*100, 'cm mudolo')
        if mudolo > 0.2:                                    #if mudolo values are more than 20 cm
            tl.forward(mudolo*100)

    else:
            tl.forward(distance*100)                        #distance = less than 5 meters. go distance in cm

    if orientation == 'CCW':                                #Rotate back again to '0'
        tl.clockwise(rotation)
    else:
        tl.anticlockwise(rotation)

    tl.land()
```

Figure 25: Code that uses the distance, rotation and orientation to make the drone move.

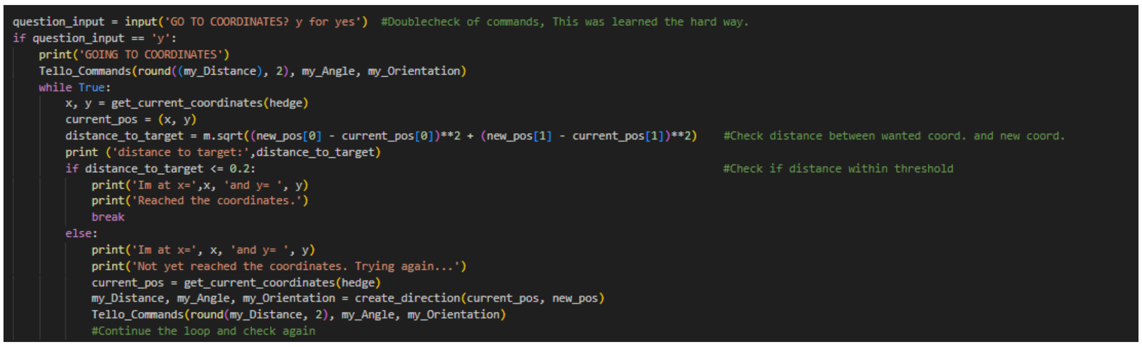
The code first orients the drone to point towards the new point of where its headed. Its decided by the if the orientation is CW or CCW (clockwise or counter clockwise). The increments is the workaround of maximum 5 meter flight distance. For easier explanation the increments are set to 5 meters, longest allowed flight. The code then check how many times the *distance* can be divided in total *increments* (ignoring the decimal). Example: If the distance is 17 meters, and increments are 5. Then the value would be 3, leaving out the decimal. Using this new value and multiplying with 100 (Tello library expects value in centimeters). The drone is then told to go 5 meters 3 times. Traveling a total of 15 meters. Calculating the last remaining 2 meters are then done with calculating the modulus. (written as % in Python). 17%5 = 2 *left over.* This value is then also multiplied by 100. The drone is instructed to go the last two meters. Totaling the 17 of 17 meters. However, the Tello library does also not let it fly shorter than 20cm, so if the modulus value is less it will ignore it.

Defining the increments as a lower value than 5 meters, it might be possible to have the a checking of new coordinates to correct itself in flight. However, the figure 25 above it is not implemented.

After the drone has taken the flight path, it then corrects its angle again so it points in the orientation it started. With this code, the drone can fly to a desired coordinate, however, there is no feedback/correction if it has failed. It follows directions blindly.

3.5.4 Self-correction of flight

After flying to a new coordinate, the drone would not know if it came to wanted position. This is counteracted by reading its position over again from the Marvelmind beacon when it has finished its flight path. Reading its coordinates from the input coordinates, it evaluates if there is a 0.2 meter error (Tello does has a minimum flight of 20 cm) in between. If there is not an error, it is in an acceptable position. If there is an error, it takes the new position and calculates the distance, rotation and orientation, then executes the new flight path. Then retries to read again, and checks if error is outside threshold.



```
question_input = input('GO TO COORDINATES? y for yes')  #Doublecheck of commands, This was learned the hard way.
if question_input == 'y':
    print('GOING TO COORDINATES')
    Tello_Commands(round((my_Distance), 2), my_Angle, my_Orientation)
    while True:
        x, y = get_current_coordinates(hedge)
        current_pos = (x, y)
        distance_to_target = m.sqrt((new_pos[0] - current_pos[0])**2 + (new_pos[1] - current_pos[1])**2)    #Check distance between wanted coord. and new coord.
        print ('distance to target:',distance_to_target)
        if distance_to_target <= 0.2:                                                                         #Check if distance within threshold
            print('Im at x=',x, 'and y= ', y)
            print('Reached the coordinates.')
            break
        else:
            print('Im at x=', x, 'and y= ', y)
            print('Not yet reached the coordinates. Trying again...')
            current_pos = get_current_coordinates(hedge)
            my_Distance, my_Angle, my_Orientation = create_direction(current_pos, new_pos)
            Tello_Commands(round(my_Distance, 2), my_Angle, my_Orientation)
            #Continue the loop and check again
```

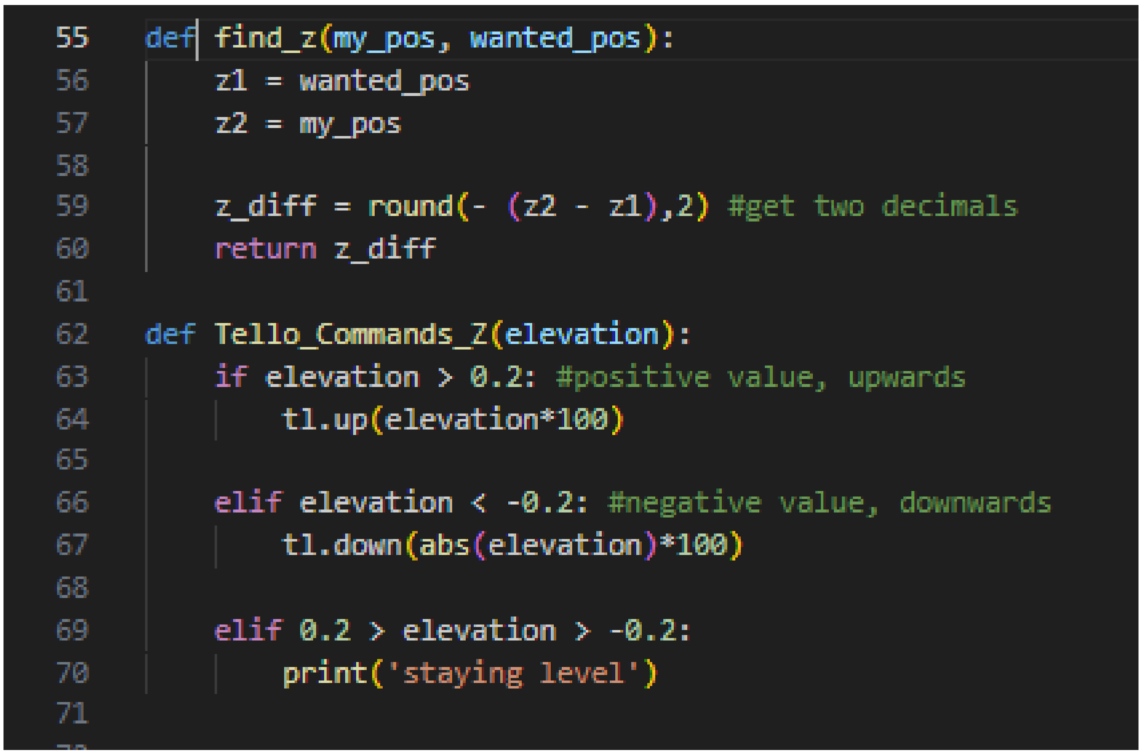
Figure 26: A loop is added after going to coordinates, that checks the new coordinates after arriving.

Figure 26 shows the loop, the loop corrects the drone if it is off the wanted coordinate. However, there is a limit on how short the drone can fly, this is defined as 20cm. Therefore the drone can be off with ± 20cm the determined position.

As of now, the drone can fly on its own to different coordinates in a closed-off room, but only in one plane. The x and y axis.

3.5.5 Implementing the Z-axis to the coordinate system

Reading the Z-axis with Marvelmind was not reliable enough to use it to control the drone (read chapter 4.1). There was a workaround this issue using the sensor on the Tello drone, reading its height by the infrared module located underneath the drone. This sensor gives reliable information about its distance to the ground beneath in millimeters.



```python
def find_z(my_pos, wanted_pos):
    z1 = wanted_pos
    z2 = my_pos

    z_diff = round(- (z2 - z1),2) #get two decimals
    return z_diff

def Tello_Commands_Z(elevation):
    if elevation > 0.2: #positive value, upwards
        tl.up(elevation*100)

    elif elevation < -0.2: #negative value, downwards
        tl.down(abs(elevation)*100)

    elif 0.2 > elevation > -0.2:
        print('staying level')

```

Figure 27: Code snippet of finding z difference and giving commands to Tello.

First function in figure 27 *find_z* uses the measured distance above ground first then the determined Z value. Comparing the values and finding the distance between. Returning the value *z_diff*. The value is positive or negative number, In respect of up and down.

Second function, *Tello_Commands_Z* takes the input *z_diff* value and checks if its more than the 20 cm threshold. If outside the threshold, the function also checks if the value is positive or negative, commanding the drone to go the distance. Note that the drone can not go down a negative value, therefore it is changed to a positive number using *absolute* function. This method works well if there is nothing beneath the drone disturbing the read-off to the ground. If a table that is 1 meter tall beneath the drone when it measures. It will correct itself from the table.

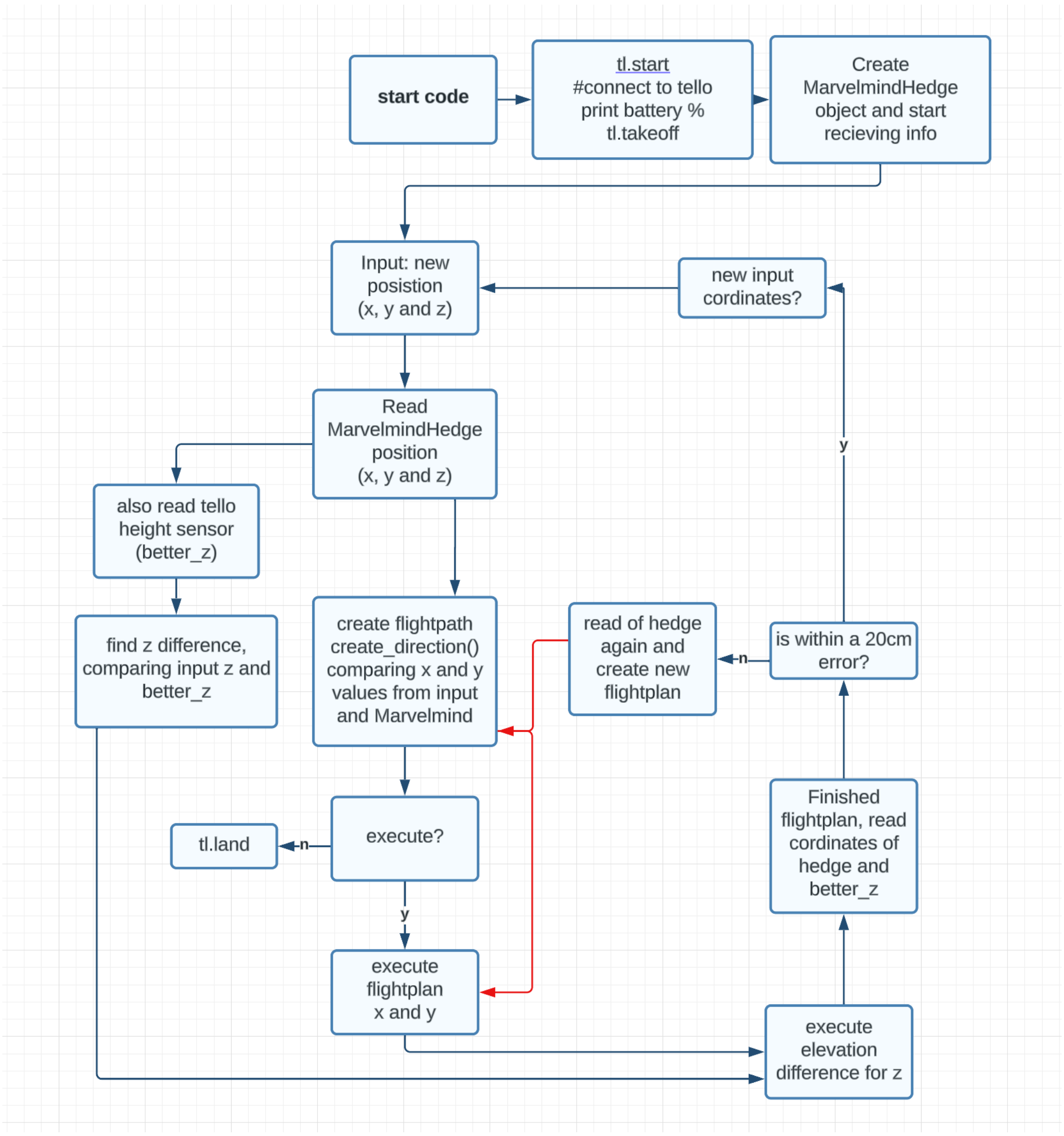

start code
tl.start
#connect to tello
print battery %
tl.takeoff
Create MarvelmindHedge object and start recieving info
Input: new posistion (x, y and z)
new input cordinates?
Read MarvelmindHedge position (x, y and z)
also read tello height sensor (better_z)
find z difference, comparing input z and better_z
create flightpath create_direction() comparing x and y values from input and Marvelmind
read of hedge again and create new flightplan
n
is within a 20cm error?
y
execute?
n
tl.land
y
execute flightplan x and y
Finished flightplan, read cordinates of hedge and better_z
execute elevation difference for z

Figure 28: Flowchart of Python code

Figure 28 gives a visualization of how the first code performs. Starting with creating a connection with Tello drone and the Marvelmind Hedge (The mobile beacon). After creating a connection, the drone takes off hovering stationary in air, waiting for coordinates to go to. When coordinates are written in the input terminal, it then reads off its own location with the Marvelmind beacon. Creating a flight path from its current location to the inputted location. The drone reads of its height position and uses this as the Z value. Using this as reference to find way to the new Z position given.

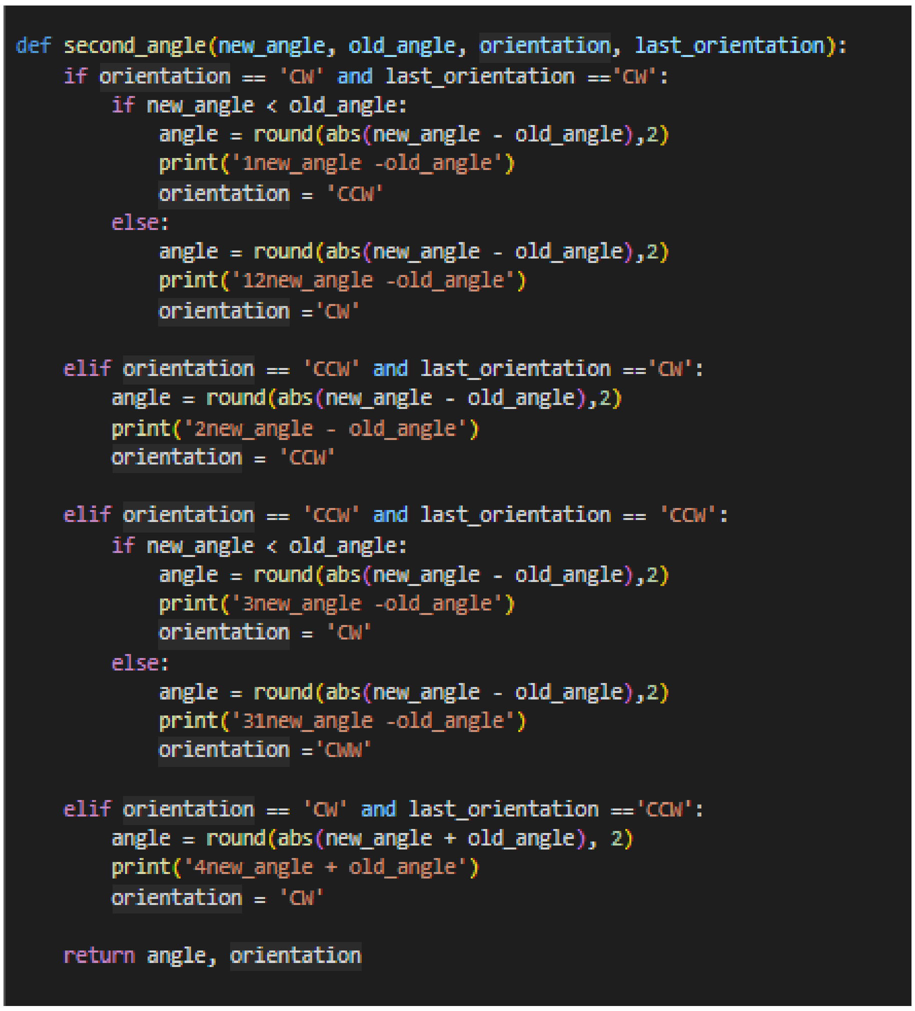
Then the code presents a flight path, asking for execution. When confirmed execution, the drone rotates towards the new position. Flies the direct difference in position A to B. When in position, it rotates back again to point in original direction. Then changes altitude in Z after the difference of measured from the drone and the input given.

When the drone has arrived at its new location (hopefully the inputted coordinate), it then first checks the Marvelmind coordinates and then the Z value from the drone. If the new location is within a 20 cm error. It is then in an acceptable position. If not an acceptable position; it then reads of again its coordinates and creates a new flight path and checks again. This loop continues until the drone is in the coordinates given in the beginning. When arriving at its wanted coordinates, it then asks for new input coordinates. Ready to start over.

3.5.6 Different Approach

A different approach to go from A to B is to part the distance up in more segments, correcting the angle midway. Resulting in always flying straight toward the input coordinate. This uses the same method for reading position and elevation, but instead of rotating back for every check it calculates a new angle from the angle of its position compared to the last angle it rotated. However, it is also important to look at the previous way the drone rotated since it can not get a negative angle value.

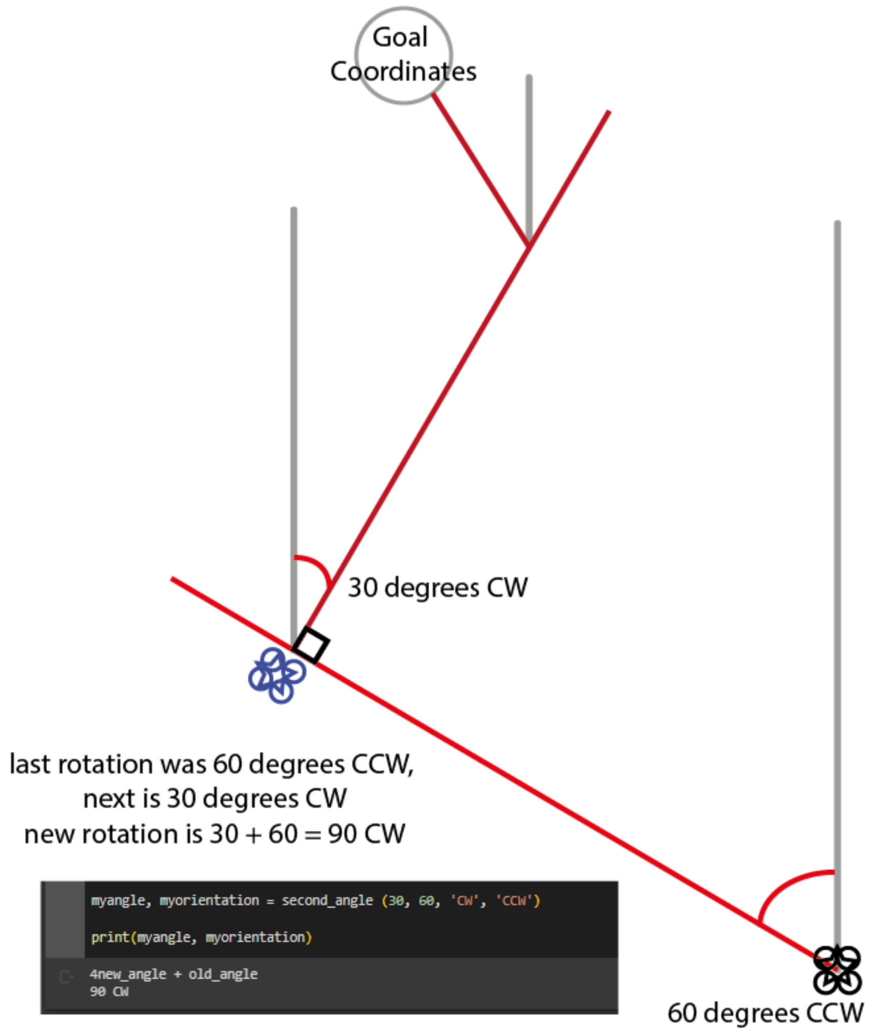
A new function is added as such:



```
def second_angle(new_angle, old_angle, orientation, last_orientation):
    if orientation == 'CW' and last_orientation =='CW':
        if new_angle < old_angle:
            angle = round(abs(new_angle - old_angle),2)
            print('1new_angle -old_angle')
            orientation = 'CCW'
        else:
            angle = round(abs(new_angle - old_angle),2)
            print('12new_angle -old_angle')
            orientation ='CW'

    elif orientation == 'CCW' and last_orientation =='CW':
        angle = round(abs(new_angle - old_angle),2)
        print('2new_angle - old_angle')
        orientation = 'CCW'

    elif orientation == 'CCW' and last_orientation == 'CCW':
        if new_angle < old_angle:
            angle = round(abs(new_angle - old_angle),2)
            print('3new_angle -old_angle')
            orientation = 'CW'
        else:
            angle = round(abs(new_angle - old_angle),2)
            print('31new_angle -old_angle')
            orientation ='CWW'

    elif orientation == 'CW' and last_orientation =='CCW':
        angle = round(abs(new_angle + old_angle), 2)
        print('4new_angle + old_angle')
        orientation = 'CW'

    return angle, orientation
```

Figure 29: Function calculating relevant angle from previous angle, with orientation

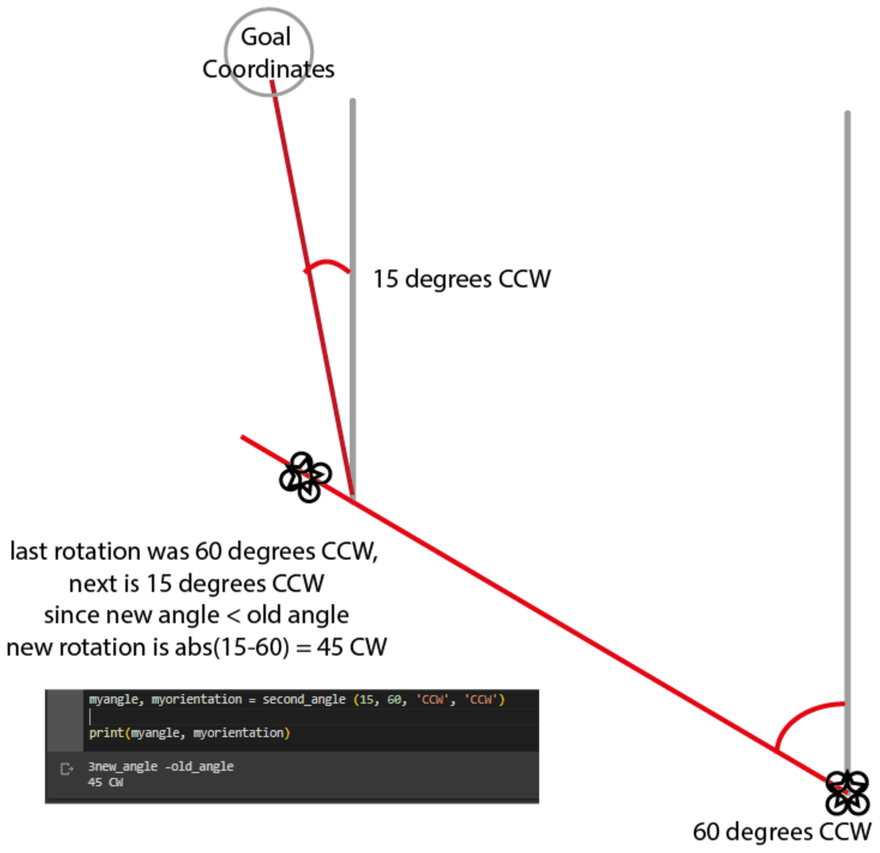
Figures drawn to give a better understanding of the code:


Goal
Coordinates
30 degrees CW
last rotation was 60 degrees CCW,
next is 30 degrees CW
new rotation is 30 + 60 = 90 CW
60 degrees CCW

```
myangle, myorientation = second_angle (30, 60, 'CW', 'CCW')

print(myangle, myorientation)
```

```
4new_angle + old_angle
90 CW
```

Figure 30: New rotation is 90 degrees CW


Goal Coordinates
15 degrees CCW
last rotation was 60 degrees CCW,
next is 15 degrees CCW
since new angle < old angle
new rotation is abs(15-60) = 45 CW
60 degrees CCW

```
myangle, myorientation = second_angle (15, 60, 'CCW', 'CCW')

print(myangle, myorientation)
```

```
3new_angle -old_angle
45 CW
```

Figure 31: New rotation is 45 degrees CW

There are different cases of how the angles should be calculated together to find the new 'relative' angle. Therefore figure 30 and figure 31 shows in the first line of the terminal what case it is in from the code in figure 29.

When pointing towards the determined coordinate, it then moves a defined amount towards it, stopping and reading again. Continuously doing this until it arrives the final coordinate. Finally, a total sum of different angles defines how much it needs to rotate back to starting position.

3.6 Creating the book model for the Tello drone

There were many ways of different approaches to create the book model. It began with using mounting points for the book to the drone. The Tello drone already has mountings for its propeller guards. The first try was to create a mount part that would fit the drone. This was done in Autodesk Fusion 360. This was first time using Fusion as a cad software, but it was a simple model to create.

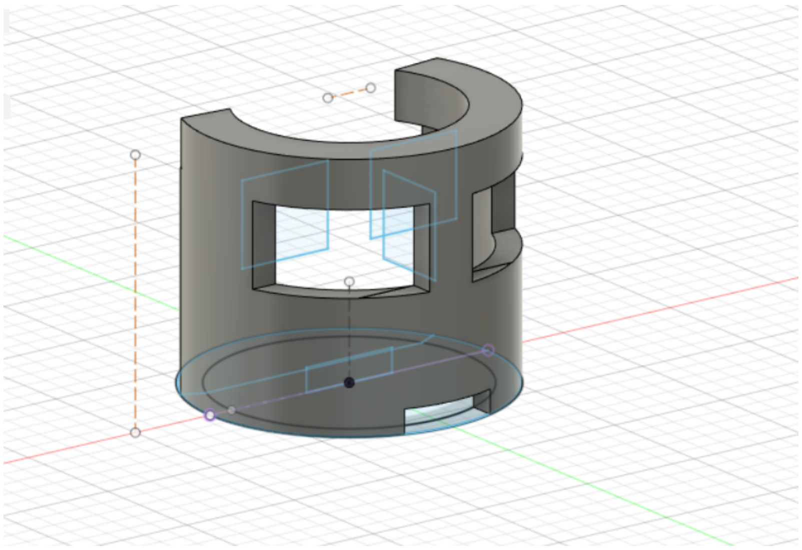


Figure 32: Mounting model created in Fusion

Figure 32 is a small part that would fit on one of the motors of the drone. This part was test to see if it could fit. After a couple of design changes it fit perfect. The design process for the book further was mostly trial and error. Testing and getting to know Fusion 360. Further design was to create a new frame around the drone.

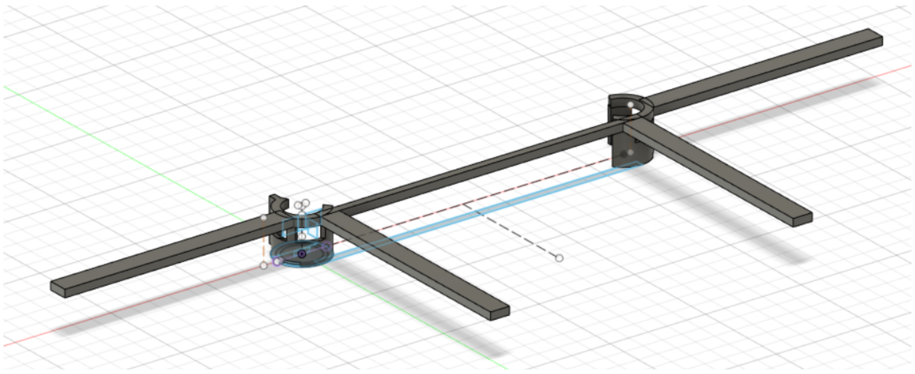


Figure 33: Further development of the frame.

The idea is that it would mount on the two front motors and the back motors Using these *arms* it was first prototype of what could be a book. Figure 34 shows the first disguise for the drone.

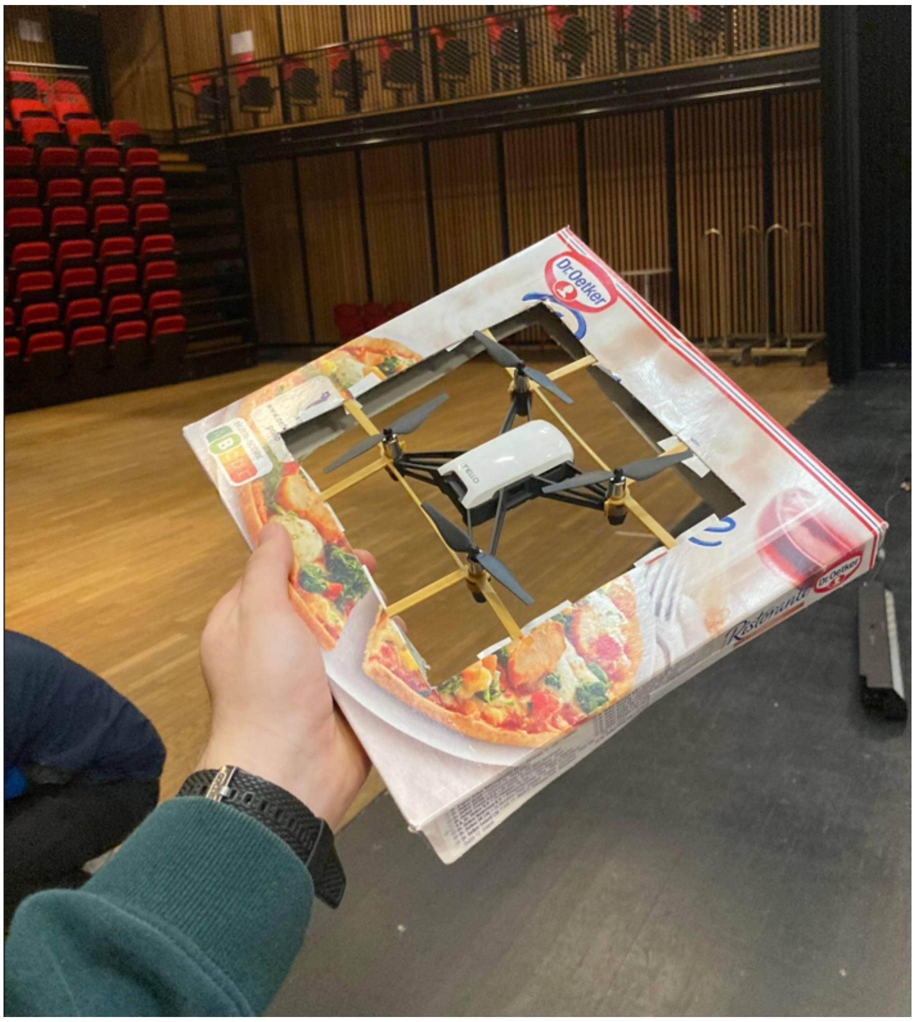


Figure 34: First test of hiding drone, using a frozen pizza box

When tested for flight it was barely able to lift itself of the ground, and poor control in air. The *book* design was not working. However it gave valuable insight in how much weight is enough. This design also did a poor job to hide the drone. The design in figure 33 was scrapped, and a new approach was tested.

Online, there was CAD models to the Tello drone and the propeller guards available for 3D printing. (Figure 35). These were modified to be the base of the mounting of the book model. Link to the original model is in the bibliography cited number: [1].

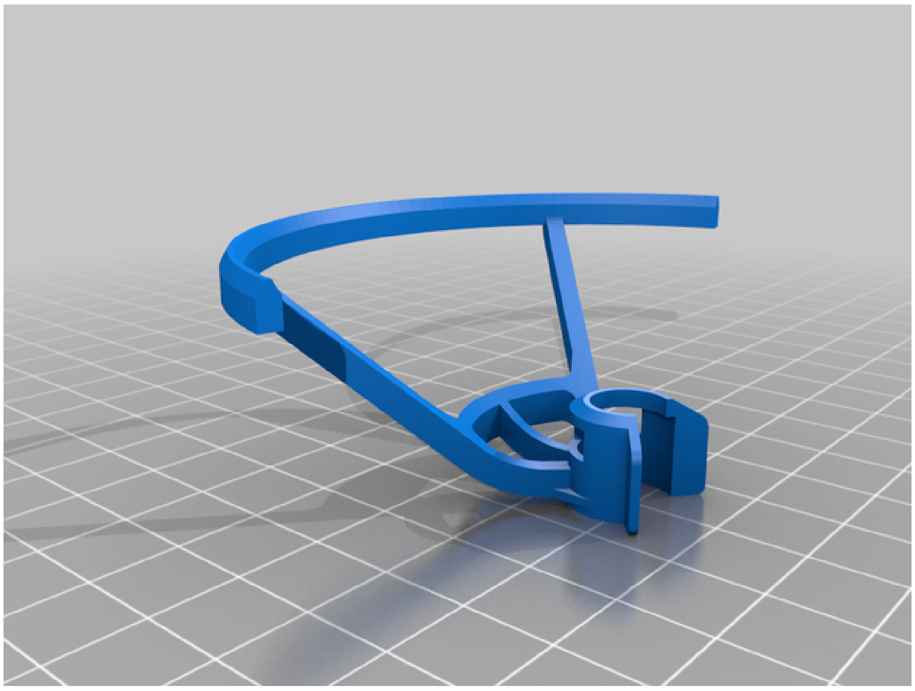


Figure 35: CAD model of the propeller guards to the Tello drone

This part is new start point for further designing the book around the drone. Making equal parts that mount diagonally from each other. The mounts were raised to cover over the propellers.

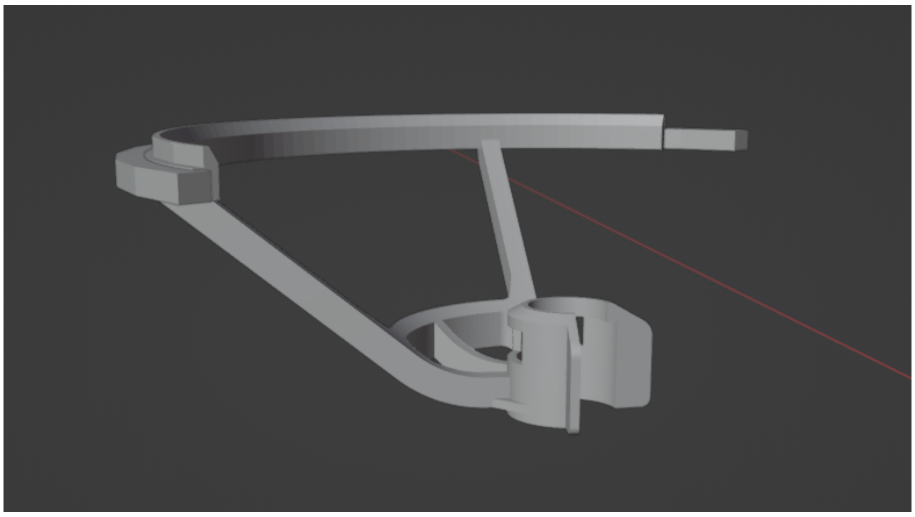


Figure 36: CAD model of the propeller guards with a thicker bezel.

Adding a thicker rim / bezel around the protector, it is easier to use this part to mount the corner piece to the drone. The propeller guard mounts on the drone as normal.

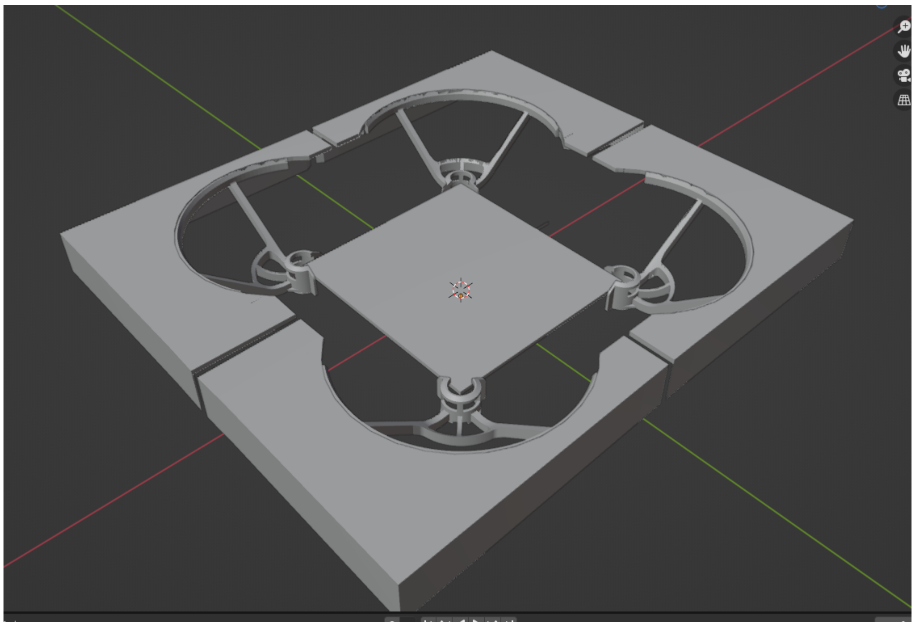
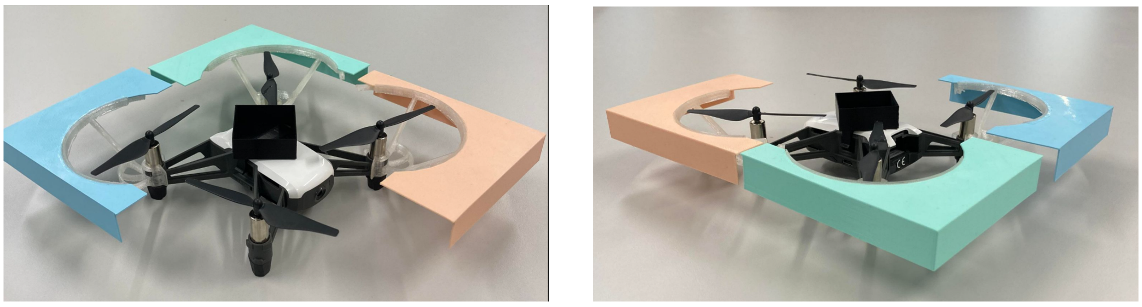


Figure 37: CAD model of potential book design

Design in figure 37 is an improved design from the previous design in figure 34 to hide the drone. This design also gave more security for both propellers and the audience.

The propeller guard and corner plate are glued together. This is done to 3D print the parts using less excess materials.



(a)

(b)

Figure 38: The first printed book design

The design in figure 38 was also not able to create good lift, and its speculated that it is that the parts around creates a big drag around the drone, plus that it is heavy in the on the outside. Also again not satisfied on how well the drone is hidden. A new design is tried, taking inspiration of a "Security cage" available for the Tello drone.

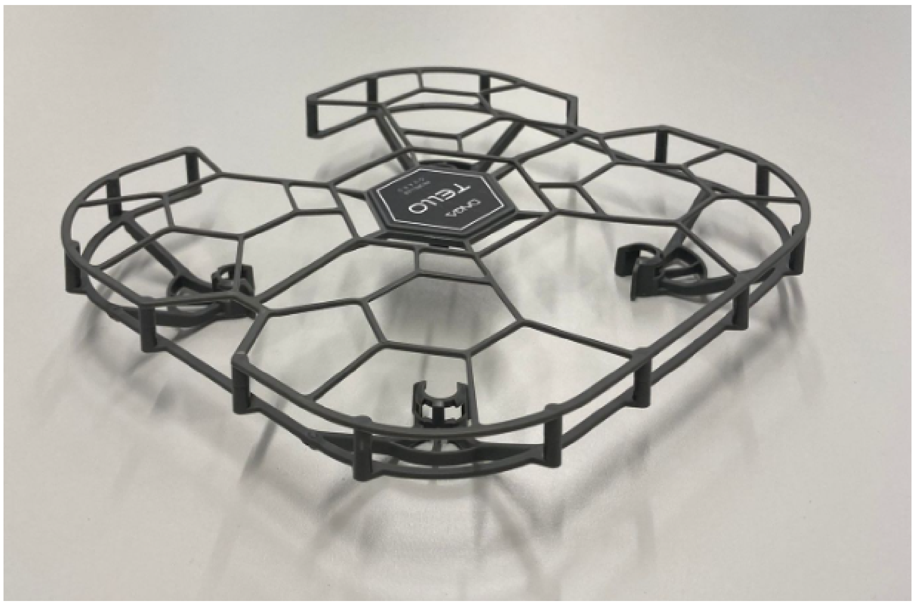


Figure 39: A Cage completely covering the drone.

Made in soft plastic, the cage is folded around the drone and mounted on the same mounts as the propeller guards.

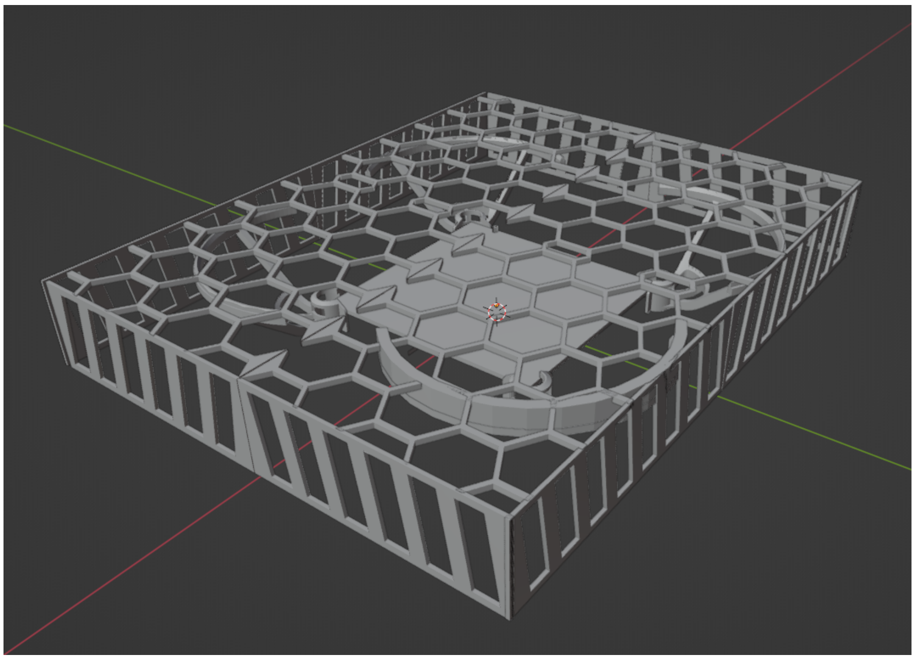


Figure 40: CAD model of potential book design, for better airflow

Using what failed, a new design was created. With thinner walls and holes in them, both the drag and weight is reduced. This design completely covers the drone. The holes solve two issues; it is reducing material, making it lighter and support for better airflow though the model. The idea is that the sides would be covered with thin paper, letting air though for airflow. But making it difficult for audience to see the drone inside. Top of the design the drone is not possible to cover, as it would make it difficult for the drone to fly. Figure 40 is the digital model before getting printed. Same propeller guards are used, but its mounted to the a ring that clears the propeller.

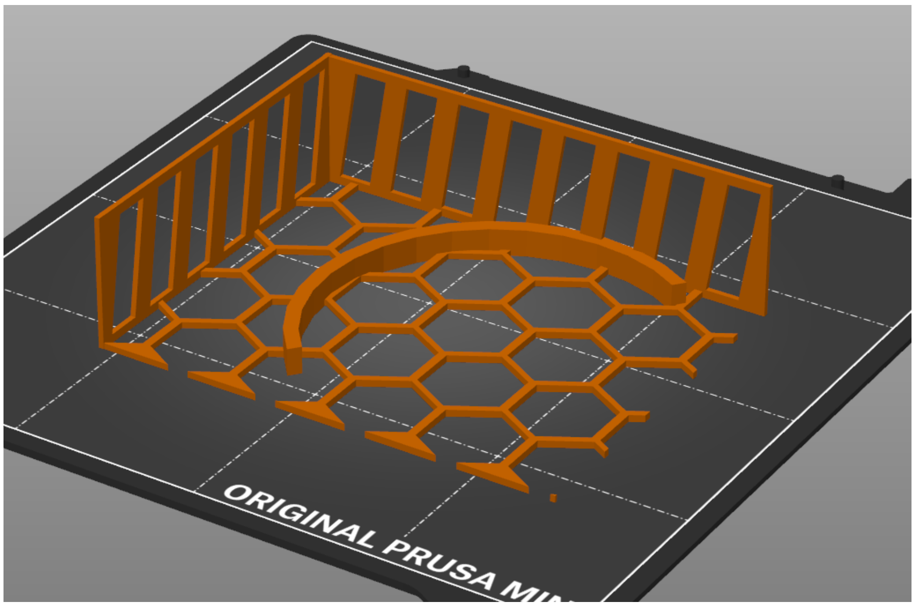


Figure 41: Corner of the book design in slicer.

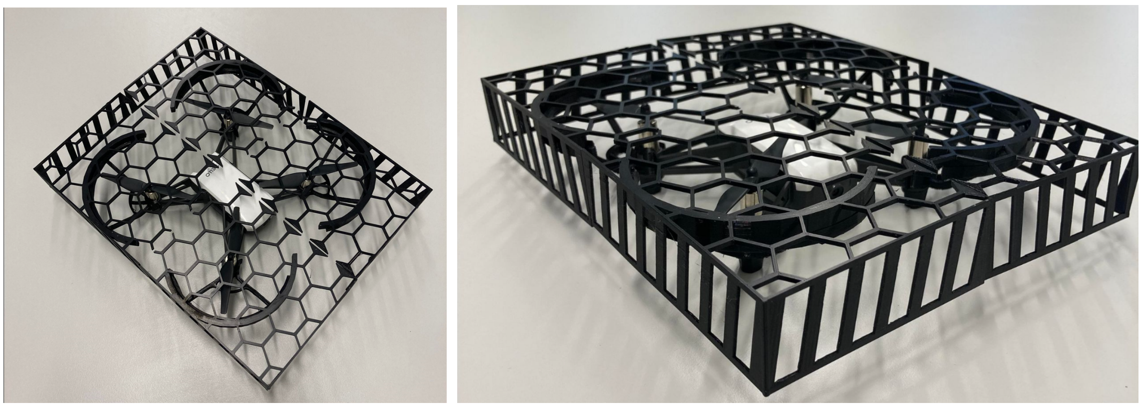


Figure 42: Complete printed book design for the Tello drone

Preferably, the frame would be colored white to be less distinguishable from paper.

4 Results

In results, it is examining the finished result of the thesis, comparing the indoor GPS, the code for the drone, and the finished design for the book.

4.1 Measuring the Precision of the Marvelmind GPS

For the first original setup of the Marvelmind in the classroom, the anchors were placed as close as possible in a rectangle. Placing anchor 1 & 2 and 10 & 11 to be exactly as possible opposite of each other. Also placing 1 & 10 and 2 & 11 for vertical for each other. All anchors were placed 2.5 meters up on the wall. This gave as much clear line of sight to the mobile beacon in center of the room. The Marvelmind software is able to measure all the distances between the anchors on its own. By comparing the distances determined by the software against the physical measurements done using a measuring tape. We can conduct a comparison on reliability and precision. Using the values from figure 8 adding the values of distances to mobile node number 7. Beacon 7 is placed in the center of the room.

	Beacon 1	Beacon 2	Beacon 7	Beacon 10	Beacon 11
Beacon 1	-	11.40m	7.41m	8.60m	-
Beacon 2	11.40m	-	7.23m	-	8.43m
Beacon 7	7.41m	7.23m	-	7.26m	7.24m
Beacon 10	8.60m	-	7.26m	-	11.30m
Beacon 11	-	8.43m	7.24m	11.30m	-

Table 4: Ground Truth values, Values measured with measuring tape:

	Beacon 1	Beacon 2	Beacon 7	Beacon 10	Beacon 11
Beacon 1	-	11.461m	7.395m	8.565m	14.302m
Beacon 2	11.461m	-	7.184m	14.093m	8.445m
Beacon 7	7.395m	7.184m	-	7.242m	7.220m
Beacon 10	8.565m	14.093m	7.242m	-	11.278m
Beacon 11	14.302m	8.445m	7.220m	11.278m	-

Table 5: Marvelmind Values, Values measured with software by Marvelmind:

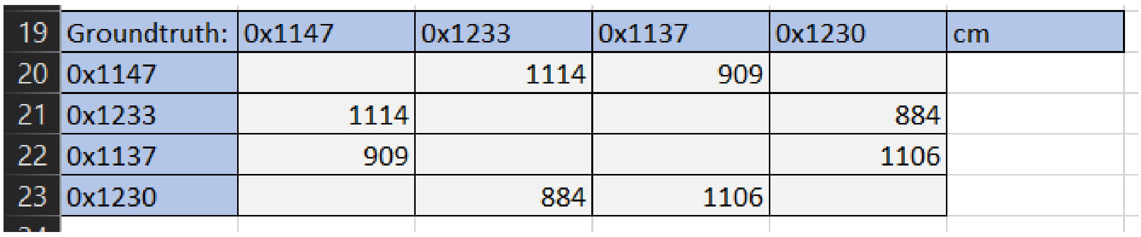
This gives an overview of how reliable the Marvelmind is at measuring distances between the anchors. For the distances between the beacons to number 7, its important to know that the mobile beacon is placed on a table 0.73 meters high, so the direct path has an incline up to 2.5 meters. (Total 1.77 meters incline)

With every anchor mounted at 2.5 meters high, missing between 2 and 6 centimeter. But the z-axis measurement of the mobile was unreliable. The values were incorrect and would miss with $\pm$ 20cm. In figure 8 the z value of node 7 is 0.998, but its placed on a table, 73cm tall.

4.2 Measuring the precision of the POZYX Creator One kit

When testing the POZYX setup, the anchors where placed on the same plane. With the setup of POZYX the measured values were as such:

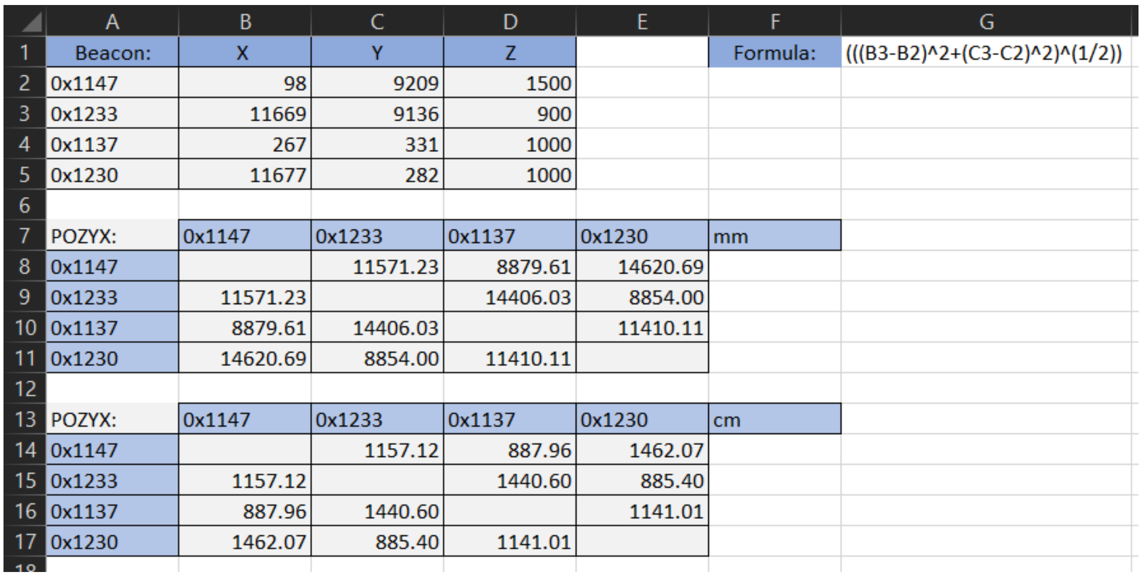
Ground Truth Values, Values measured with measuring tape:



19	Groundtruth:	0x1147	0x1233	0x1137	0x1230	cm
20	0x1147		1114	909		
21	0x1233	1114			884	
22	0x1137	909			1106	
23	0x1230		884	1106		

Figure 43: Found distances using tape measure. Measured in centimeters

Pozyx Values, Values measured with software by Pozyx:



	A	B	C	D	E	F	G
1	Beacon:	X	Y	Z		Formula:	(((B3-B2)^2+(C3-C2)^2)^(1/2))
2	0x1147	98	9209	1500			
3	0x1233	11669	9136	900			
4	0x1137	267	331	1000			
5	0x1230	11677	282	1000			
6							
7	POZYX:	0x1147	0x1233	0x1137	0x1230	mm	
8	0x1147		11571.23	8879.61	14620.69		
9	0x1233	11571.23		14406.03	8854.00		
10	0x1137	8879.61	14406.03		11410.11		
11	0x1230	14620.69	8854.00	11410.11			
12							
13	POZYX:	0x1147	0x1233	0x1137	0x1230	cm	
14	0x1147		1157.12	887.96	1462.07		
15	0x1233	1157.12		1440.60	885.40		
16	0x1137	887.96	1440.60		1141.01		
17	0x1230	1462.07	885.40	1141.01			

Figure 44: Found distances by Pozyx.

Measuring the mobile node distance from the walls. The Mobile beacon is placed 593.6 centimeter and 470.8 centimeter from the walls. (corresponding to x and y)

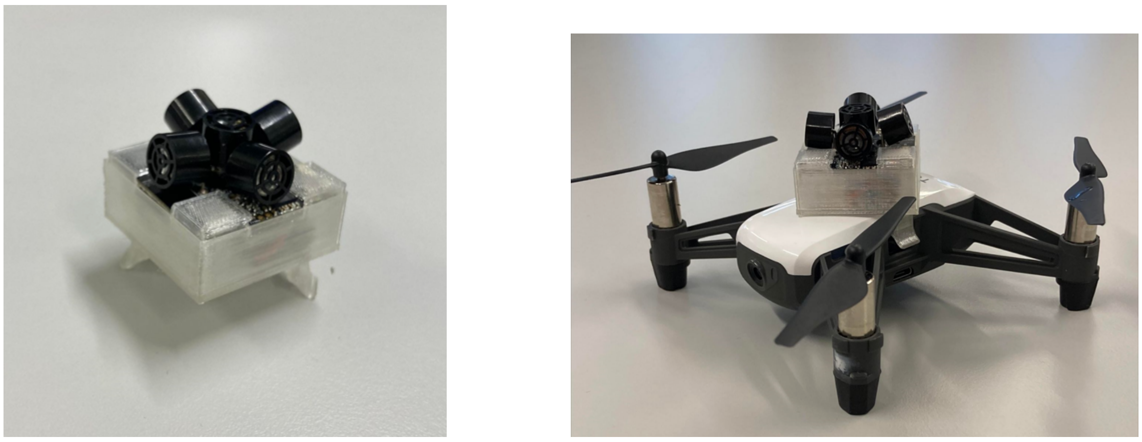
Reading of the mobile beacon 0x682d in figure 11, Its coordinates are X = 6193, Y= 5247 and Z = 1512. However, when the node was laying in the center on a table in the classroom, 73 centimeters tall. Comparing the results, the error is off with ± 25 - 55 cm, and the Z-coordinate was off with 80cm.

4.3 Combining the Tello with Marvelmind

4.3.1 Casing for Marvelmind Mini TX

A three piece part was made to hold the mobile node in place on the drone:

Figure 16 shows the compartment for the mobile node. The bottom is slid inside the middle part from the top. This lets it bend around the top of the drone, without also bending the square compartment. The top part slides on the inside of the middle compartment. The middle and top compartment's walls are 1mm thick. The middle compartment is 30mm wide, leaving enough space for the node (without its casing) and the top cover. This mount is also created with 3D printing. This was printed with no issues, but the bending part (bottom piece) is printed laying sideways, this is to print without support, but mainly to get greater bending strength as well.



(a) (b)

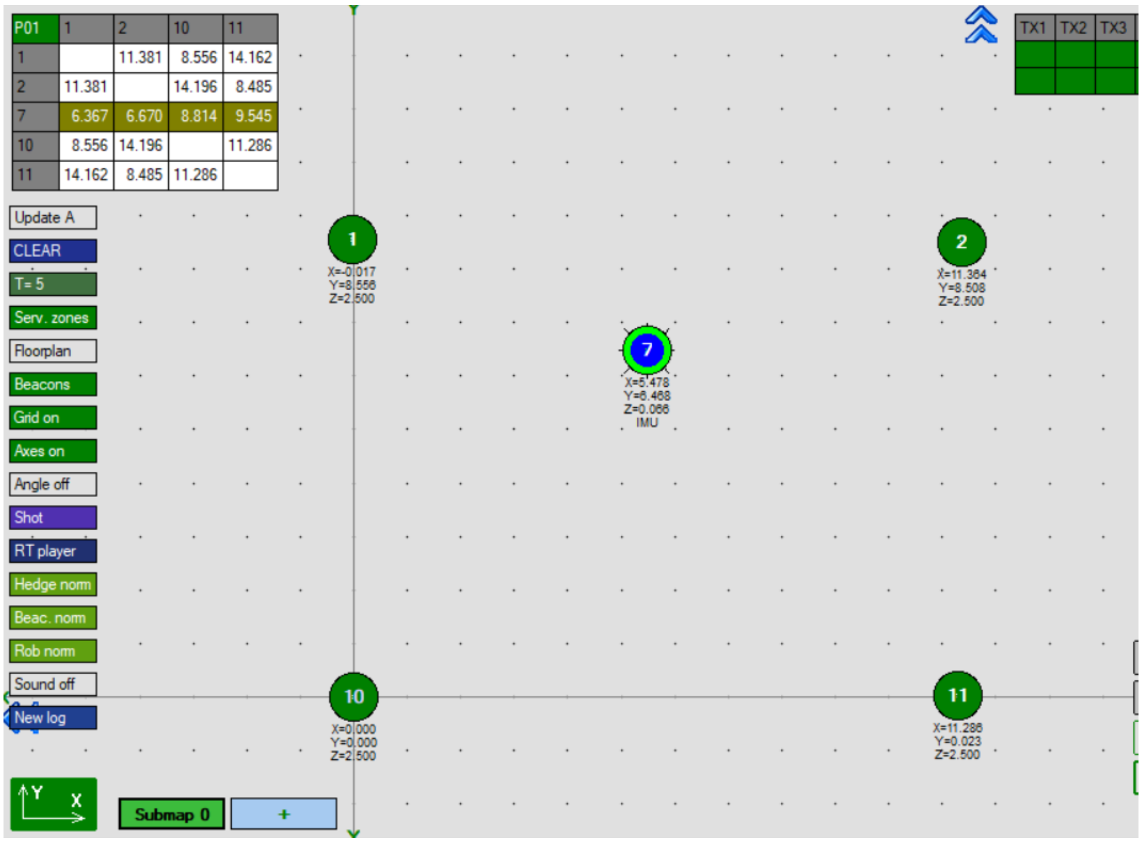
Figure 45: Picture of the Marvelmind Mini TX mounted on the Tello drone

Removing the original casing from the node makes it lighter and smaller, but also more exposed for damage.

4.3.2 Controlling Tello Drone

The final result for controlling the drone are multiple scripts that proves the different ways to approach the issue. For both of the codes to work, it is critical that the Tello drone is facing towards *north* relative to the location system. The first approach made was a program that reads its own position and then rotates the drone towards the entered coordinate. The next step is to fly the total length between the distance. When arrived at the (expected) final position, the drone reads of its altitude from the ground beneath it and corrects this according to the inputted Z variable. Finally the drone reads and checks that it is within a 20 cm error window in x, y and z. If not, it reads its own position and try again.

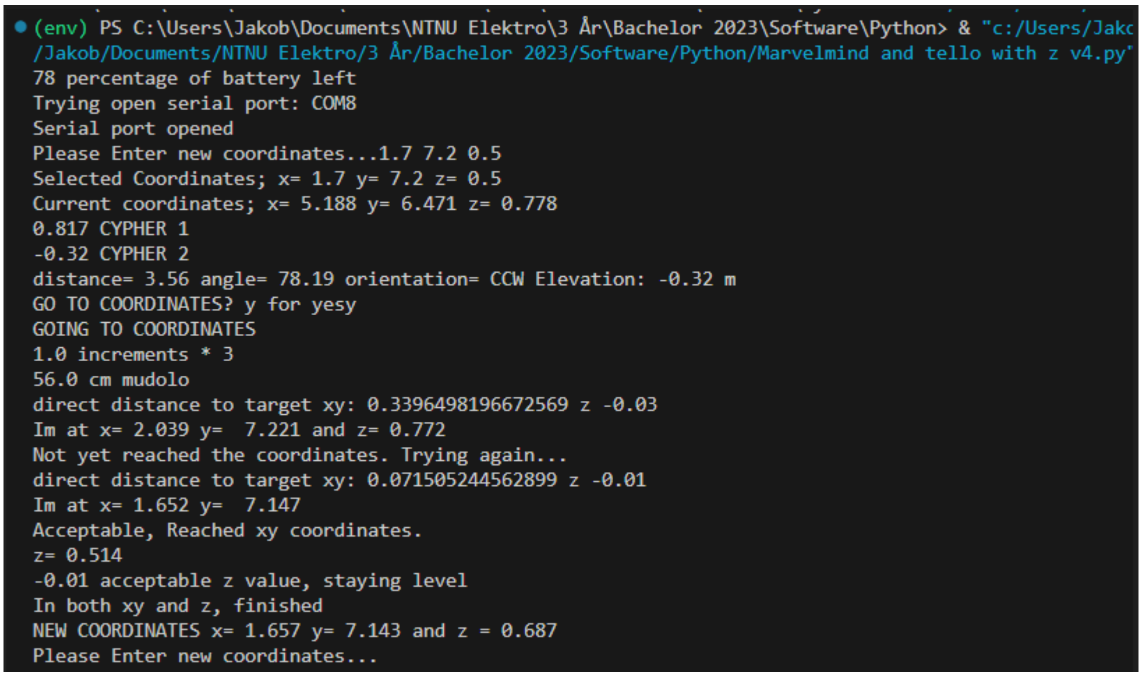
Before starting the code, the checked coordinate of the drone is on the map as figure 46



P01	1	2	10	11
1		11.381	8.556	14.162
2	11.381		14.196	8.485
7	6.367	6.670	8.814	9.545
10	8.556	14.196		11.286
11	14.162	8.485	11.286	

Figure 46: Drone on the map, before flight.

When starting, the drone takes of, asks for the new position to go to. The inputted value here is X= 1.7, Y= 7.2 and Z= 0.5. A lot of other information is printed in the terminal, but the important to notice is in figure 47.



```
● (env) PS C:\Users\Jakob\Documents\NTNU Elektro\3 År\Bachelor 2023\Software\Python> & "c:/Users/Jakc
/Jakob/Documents/NTNU Elektro/3 År/Bachelor 2023/Software/Python/Marvelmind and tello with z v4.py'
  78 percentage of battery left
  Trying open serial port: COM8
  Serial port opened
  Please Enter new coordinates...1.7 7.2 0.5
  Selected Coordinates; x= 1.7 y= 7.2 z= 0.5
  Current coordinates; x= 5.188 y= 6.471 z= 0.778
  0.817 CYPHER 1
  -0.32 CYPHER 2
  distance= 3.56 angle= 78.19 orientation= CCW Elevation: -0.32 m
  GO TO COORDINATES? y for yesy
  GOING TO COORDINATES
  1.0 increments * 3
  56.0 cm mudolo
  direct distance to target xy: 0.3396498196672569 z -0.03
  Im at x= 2.039 y=  7.221 and z= 0.772
  Not yet reached the coordinates. Trying again...
  direct distance to target xy: 0.071505244562899 z -0.01
  Im at x= 1.652 y=  7.147
  Acceptable, Reached xy coordinates.
  z= 0.514
  -0.01 acceptable z value, staying level
  In both xy and z, finished
  NEW COORDINATES x= 1.657 y= 7.143 and z = 0.687
  Please Enter new coordinates...
```

Figure 47: Information out of terminal.

After receiving wanted coordinates, the drone reads of its own coordinates. Printed as X= 5.188, Y= 6.471 and Z= 0.788. It then calculates just the distance and angle between the X & Y coordinates. The values calculated are = 3.56 meters distance and 78.19 degrees counter-clockwise, lastly the Z difference is -0.32m. The script asks for confirmation to go to the coordinates. The drone rotates the angle in the orientation the desired coordinate is. It has been defined in the code that it shall not go further than 3 meters in one go. Therefore the drone goes the first 3 meters, then the 56 cm afterwards. The terminal does not print that it changes elevation, however, the drone do go 32 cm down. Now finished with the instructions, the drone rotates back facing *north*. It then reads its new coordinates and calculates the distance between the desired input (it is printed in wrong order in the terminal). The calculated distance is 0.33 meters in xy-plane, and is not in the acceptable error zone (± 20 cm), The error in the Z-plane is -0.03. Trying again, the drone uses the new calculated distance and the angle and tries to reach the position a second time. When done, the new position is X= 1.652, Y= 7.147 and Z= 0.514. Both XY and Z are withing the 20 centimeter error window and therefore the drone in done, asking for new coordinate to go to.

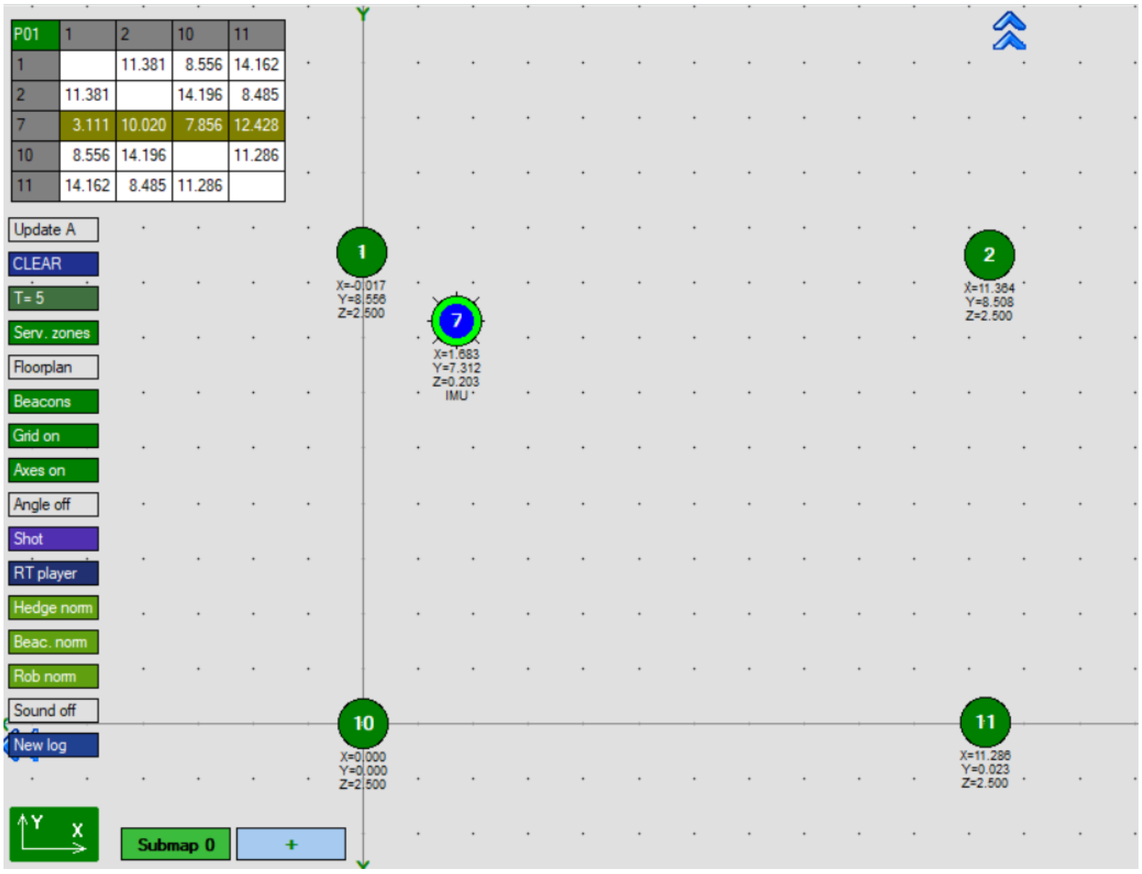

P01	1	2	10	11
1		11.381	8.556	14.162
2	11.381		14.196	8.485
7	3.111	10.020	7.856	12.428
10	8.556	14.196		11.286
11	14.162	8.485	11.286	

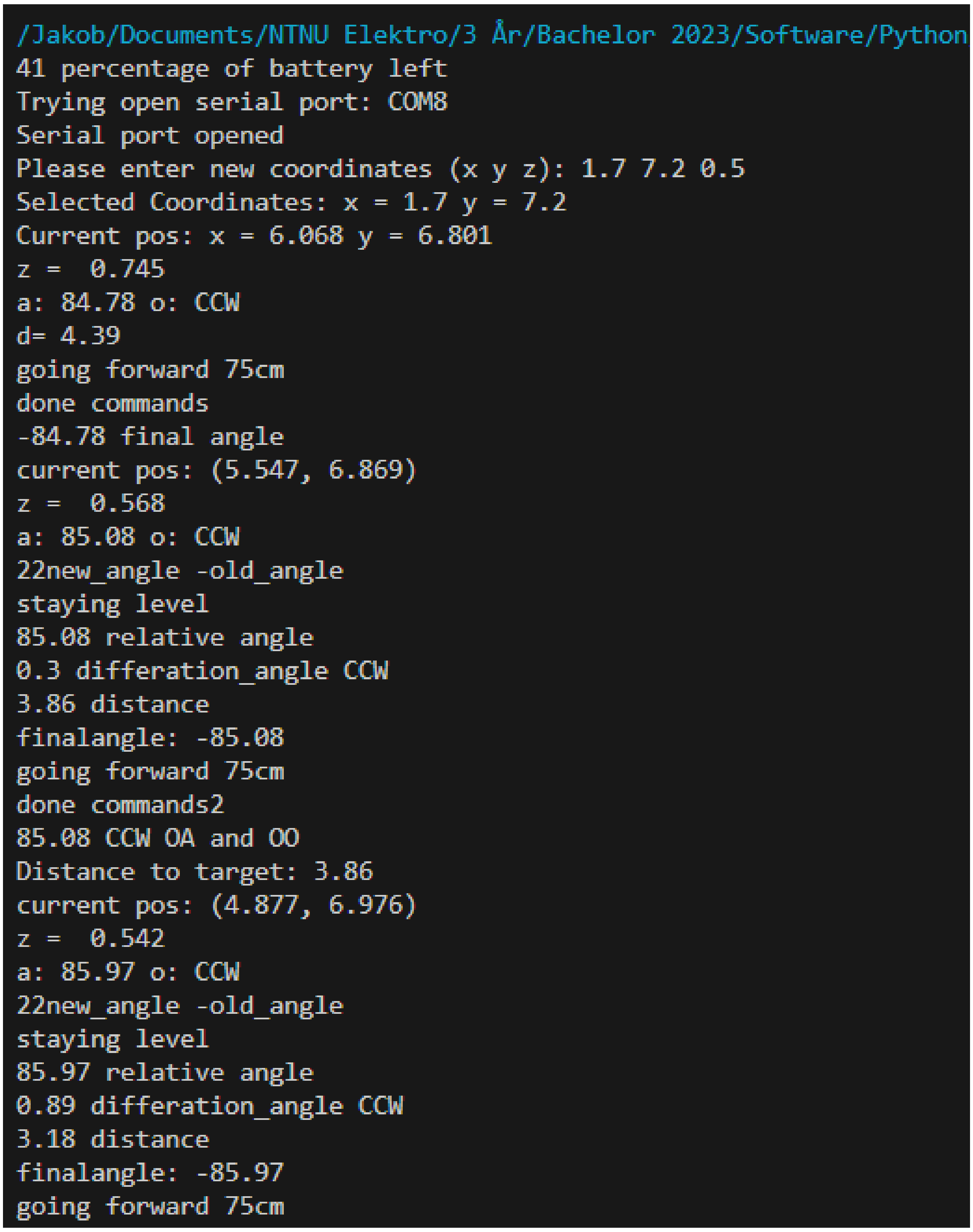
Figure 48: Drone positioned on the map, after flight.

Second approach checks the drones position and finds the angle between itself and final coordinate. Then drives a determined distance. Then checks position again. From this new position. it finds a new angle. And by comparing the new angle to the previous rotated angle it then hones its angle and then approach the new location the determined distance.

In the second approach, the drone tries to continuously point towards the determined finished coordinate. The functions for calculating the angle, orientation and distance is the same in the first code. The main difference is that the drone measures its coordinates midway.

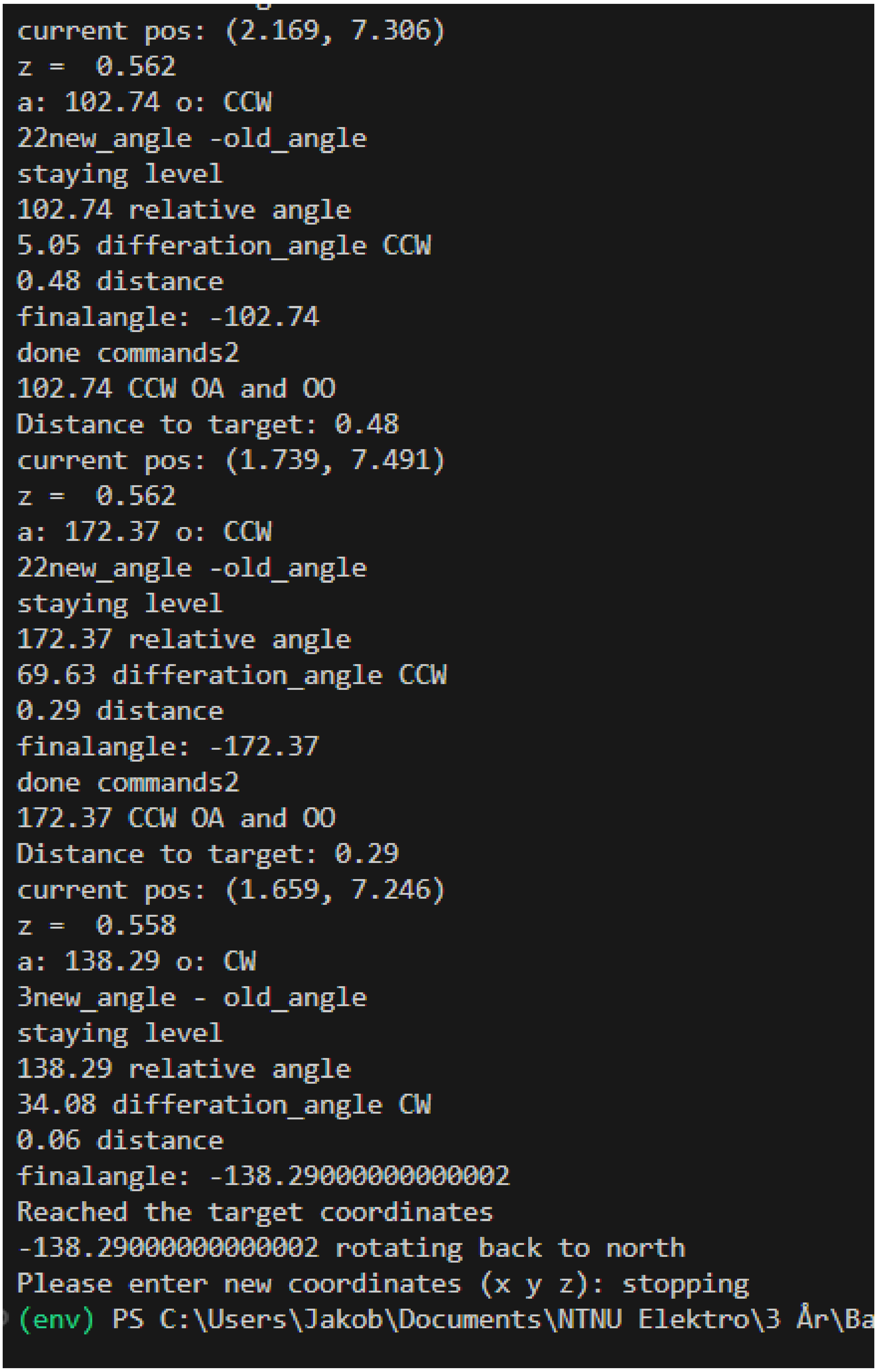
The program begins with connecting to the systems, and then the Tello takes of, while levitating it asks for new coordinates. The same coordinates are given as input, X= 1.7, Y= 7.2 and Z= 0.5.

Again, the angle and distance is calculated. The drone rotates towards the destined position (84.78 degrees CCW) and since the distance is more than defined increments (0.75 meters) it flies the 75 centimeters towards the goal position. Then the drone fixes to its in correct Z- value. From the new position calculates the distance and rotation. However, the new angle (85.06 degrees CCW) calculated is compared with the old angle what was previously executed to find the angle the drone needs to rotate. new_angle - old_angle (85.08-84.78) = 0.3 CCW. Small adjustment is made, and then a new 75 centimeters are moved towards the goal coordinate.



```
/Jakob/Documents/NTNU Elektro/3 År/Bachelor 2023/Software/Python
41 percentage of battery left
Trying open serial port: COM8
Serial port opened
Please enter new coordinates (x y z): 1.7 7.2 0.5
Selected Coordinates: x = 1.7 y = 7.2
Current pos: x = 6.068 y = 6.801
z =  0.745
a: 84.78 o: CCW
d= 4.39
going forward 75cm
done commands
-84.78 final angle
current pos: (5.547, 6.869)
z =  0.568
a: 85.08 o: CCW
22new_angle -old_angle
staying level
85.08 relative angle
0.3 differation_angle CCW
3.86 distance
finalangle: -85.08
going forward 75cm
done commands2
85.08 CCW OA and OO
Distance to target: 3.86
current pos: (4.877, 6.976)
z =  0.542
a: 85.97 o: CCW
22new_angle -old_angle
staying level
85.97 relative angle
0.89 differation_angle CCW
3.18 distance
finalangle: -85.97
going forward 75cm
```

Figure 49: Terminal during flight with program number 2



```
current pos: (2.169, 7.306)
z =  0.562
a: 102.74 o: CCW
22new_angle -old_angle
staying level
102.74 relative angle
5.05 differation_angle CCW
0.48 distance
finalangle: -102.74
done commands2
102.74 CCW OA and OO
Distance to target: 0.48
current pos: (1.739, 7.491)
z =  0.562
a: 172.37 o: CCW
22new_angle -old_angle
staying level
172.37 relative angle
69.63 differation_angle CCW
0.29 distance
finalangle: -172.37
done commands2
172.37 CCW OA and OO
Distance to target: 0.29
current pos: (1.659, 7.246)
z =  0.558
a: 138.29 o: CW
3new_angle - old_angle
staying level
138.29 relative angle
34.08 differation_angle CW
0.06 distance
finalangle: -138.29000000000002
Reached the target coordinates
-138.29000000000002 rotating back to north
Please enter new coordinates (x y z): stopping
(env) PS C:\Users\Jakob\Documents\NTNU Elektro\3 År\Ba
```

Figure 50: Terminal during flight with program number 2

As the Tello approaches the goal coordinates, it continuously checks if Z - value is within acceptable error. If it is, it stays level.

The final position it arrives to in figure 50 is X= 1.659, Y= 7.246 and Z = 0.558. The total distance is 6 centimeters from the coordinate that was given as input. Finally the Tello rotates back, pointing *north*.

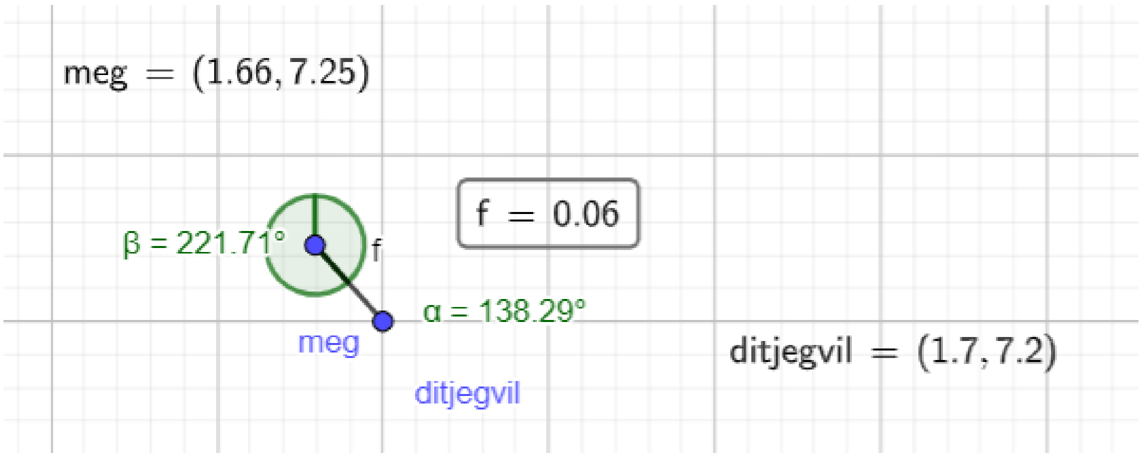


Figure 51: Position difference between the final positions

Both of these programs works well for solving the challenge of controlling a drone to get from one coordinate to another, but also correcting itself on its own.

4.4 Finished design of the flying book

The finished model for the book is hidden inside the book cover. Using this design the book is hidden when you see it straight on and the actor can show the book to the audience without revealing there is a drone inside.

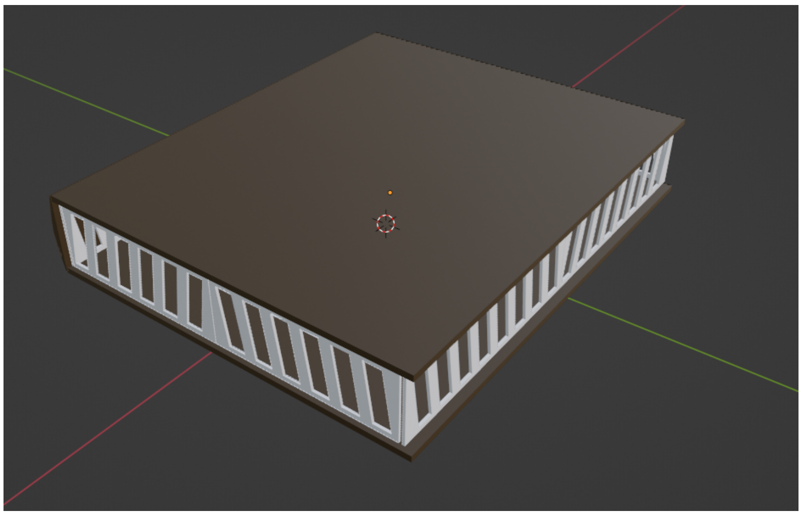


Figure 52: 3D model of book cover

The actor then could open the first page of the book and the *papers* inside can take off freely from the actors hand.

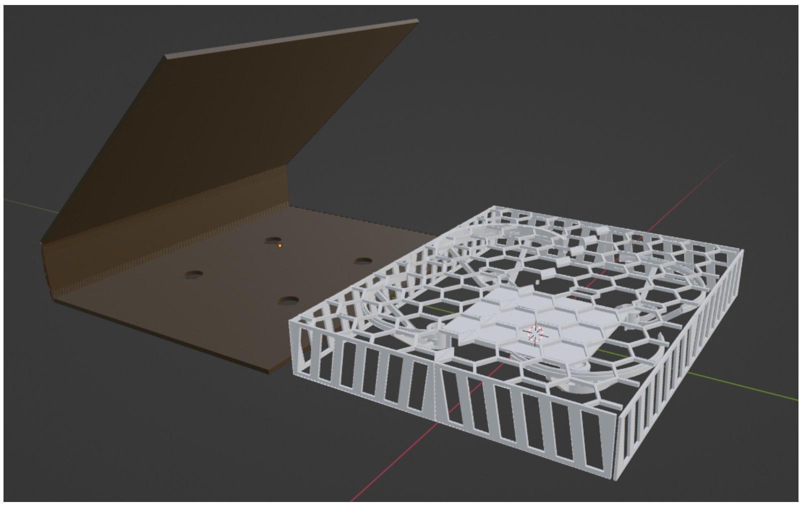


Figure 53: 3D model of book cover separated

The wholes on the backside helps the drone to stay in place without slipping out when the book is closed. The cage around the drone is 4 parts, each a corner of the drone.

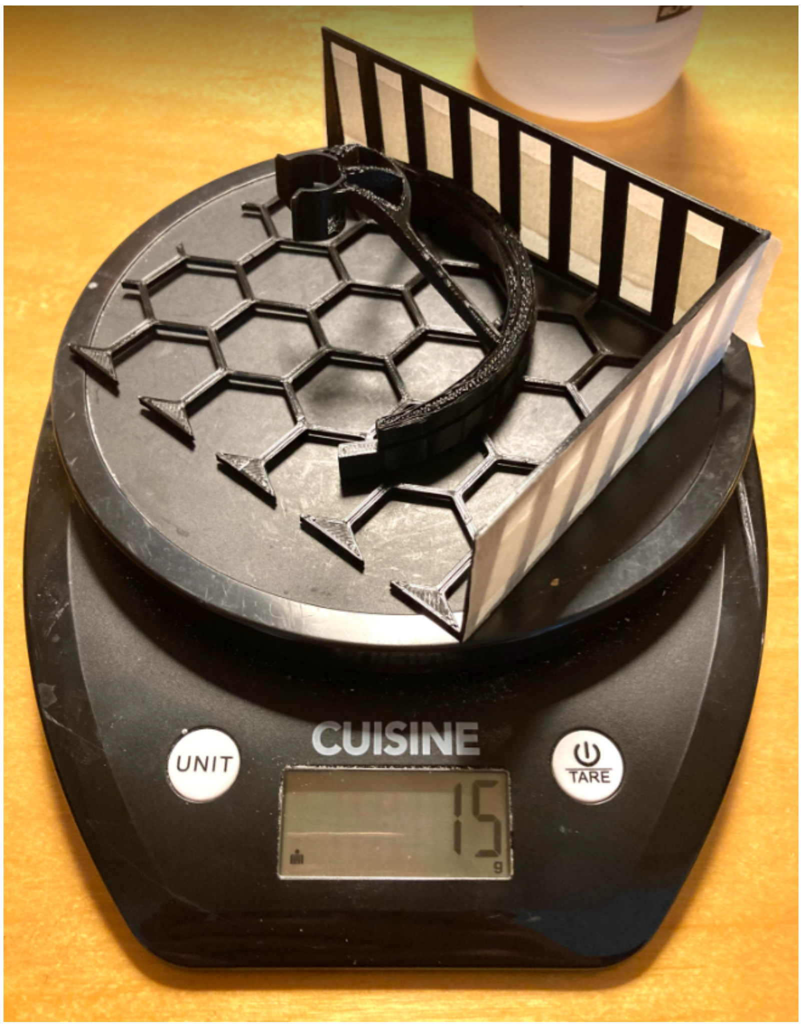


Figure 54: Weighted book disuise

The 3D printed part weights 15 grams, since it carries one on each propeller mount, the drone lifts a total of 60 grams. The printed frame for the drone is 23cm long, 18 cm wide and 3 cm tall.

To hide the drone further, the idea is to cover the holes in the side panels, using masking tape around. This is to add minimal weight but still cover the drone.

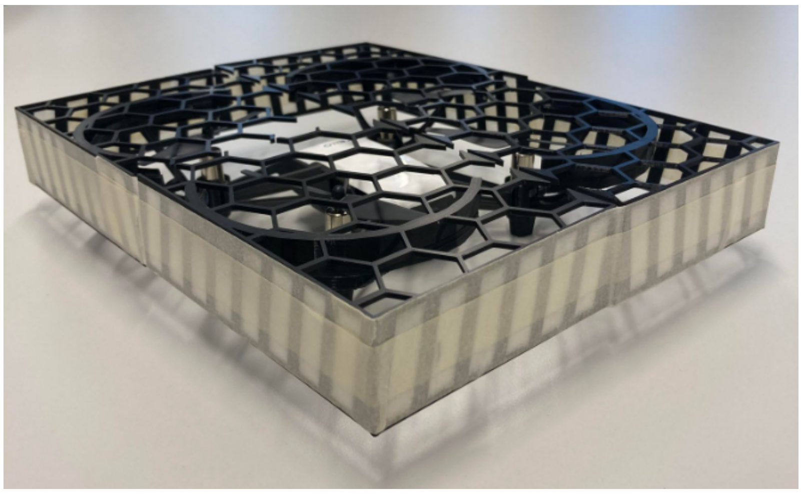


Figure 55: Using tape to cover the frame better

Here the frame around the drone is covered in masking tape. Preferably the frame would be printed in white plastic.

5 Discussion

In this chapter, it is discussed what worked well and what could have been done differently. It is also theorized about possible reasons for errors and possible solutions.

5.1 Indoor Location System Measuring and Errors

When comparing the Marvelmind values to the measured values found with measuring tape. We see that there is a ±2-4 cm offset in direct distance from mobile beacon to anchors. This error is an acceptable error when the distance from anchor to mobile beacon is around 7.2 meters. The longest distance from anchor to anchor is 1100 cm it misses with 6.1 cm.

5.1.1 Discussion on measured results

In an effort to get better reading results of the Z axis, one set of anchors diagonal of each other were placed down so they were 1 meter above the ground. (Look at the z values of the beacons in figure7) This was done to create a greater cross-reference when the anchors were reading the distances to the mobile beacon. However the results were disappointing, Reading the x, y as z coordinates were more off. The reason for this is probably because of objects in the classroom such as chairs and desks. So having all 4 anchors mounted on the wall 2.5 meters up gave best results. To achieve even better results, it could be possible to tweak settings in the Marvelmind software. This was not done, so the settings used were *straight out of the box.*

5.1.2 Errors and issues with Marvelmind

With the Marvelmind there were couple of issues, mostly connection and readings. Reading the position of the mobile beacon would sometimes have spiking from its coordinate, another issue was connection with one of the beacons in the room. When not able to connect to the beacon at all, a curious test to just change the antenna of the modem that is connected to the PC. Changing the stock linear antenna, to a circular antenna stopped the connection errors and also reduces the spikes. Figure56 shows the different antennas and the modem, the left one is the stock antenna and the right is the changed one.

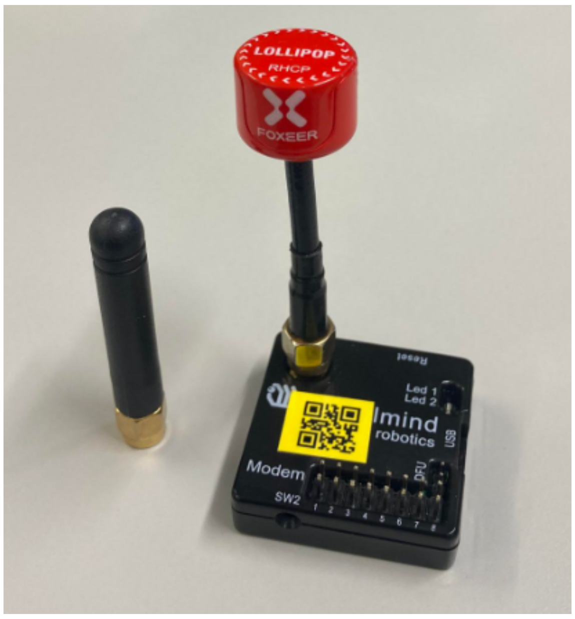
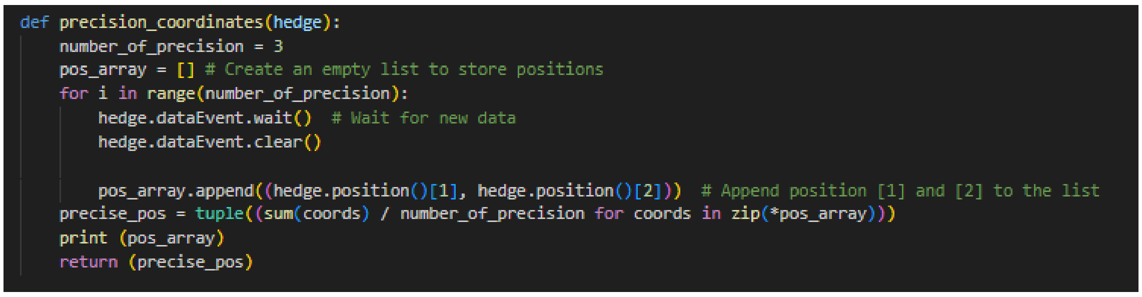


Figure 56: Different antennas on the Marvelmind modem [11]

The probable reason this fixed the connection issues was the way the antenna receives and sends out signals. Another fix for better readings was to have a clear line of sight. It was important not to stand between the mobile beacon and anchors as this resulted in weird data.

A solution before the spikes were solved, was to have a deviation filter. Reading off the location multiple times and then calculate the mean value of these positions. Figure 57 below shows the code that was written to do this. The code works well, however, it was not used.



```
def precision_coordinates(hedge):
    number_of_precision = 3
    pos_array = [] # Create an empty list to store positions
    for i in range(number_of_precision):
        hedge.dataEvent.wait()  # Wait for new data
        hedge.dataEvent.clear()

        pos_array.append((hedge.position()[1], hedge.position()[2]))  # Append position [1] and [2] to the list
    precise_pos = tuple((sum(coords) / number_of_precision for coords in zip(*pos_array)))
    print (pos_array)
    return (precise_pos)
```

Figure 57: Code for reading finding mean position

5.2 Pozyx Creator one

The Pozyx Location system gave disappointing results, a 25 - 55 cm error with X and Y values. The Z value is off with 80 cm. This is not good enough results for a reliable indoor GPS-system for a drone. Looking at the coordinates for the beacon 0x1137 (beacon that is in origo, figure 44). The beacon is not completely in origo, its off 2.6 cm in X and 3.3 cm in Y. Taking the mobile beacon values and move them towards origo with this offset. The new position is X = 566, Y = 437 cm. The ground truth error is then X = 27.6cm and Y = 33 cm. These values are improved, but still not an acceptable error.

However, it was not done more extensive testing knowing that Marvelmind was mainly going to be used for the drone. So testing with cross-referencing the positions of the diagonal anchors high up and at ground level was not done. Maybe this would give a greater Z value.

Further, there could have been more ways to test the Pozyx system. Similar to Marvelmind, no settings of frequencies or software were changed during testing, it could also be this would improve better readings.

5.2.1 Extracting location data from Pozyx

There is also a other issue using Pozyx location system to control the drone. Since the Tello creates a connection to the laptop using a local hot spot, receiving location data of Pozyx though MQTT would not be possible as it could not connect to WiFi.

5.3 Comparing the Local GPS Systems

When comparing the different indoor GPS-Systems, there is a significant difference with using UWB or supersonic sound to locate the mobile beacon. Overall did Marvelmind give better results in performance and precision, but showed worse reliability. For Marvelmind to have the best results, the anchors and the mobile beacon needs to have a clear line of sight, as standing between sensors cause spikes in values. Pozyx in comparison have no issue locating the mobile beacon with objects around, but the trade-off is worse precision.

It is also important to mention human error when testing and measuring both systems. The only tool used for measuring ground truth distances was a measuring tape, slight angle between anchors could result in a couple of centimeter offset.

In final comparison, it is difficult to compare Marvelmind to Pozyx. While Marvelmind using supersonic sound was a lot more precise, there were errors with objects being in the way for the anchors. Pozyx failing to be precise was more consistent with connection. Also, using UWB is supposed to be greater through walls as well, giving a bigger area for usability.

[compare prices?]

5.4 Tello Drone

The commands given to the Tello drone are mostly just to move it around or turn it, except for the reading of its battery percentage or altitude. However, how precise the drone follows instructions. An example of this is instructing the drone to fly 100cm forward. But when executing instructions, it only flies 95 cm in total, this could cause issues. This also applies to rotation in the yaw axis.

The accuracy of the drone is also why having an acceptable window of 20 cm error from the final coordinate is OK.

If 20 centimeter error window would be evaluated as too extensive. A possible solution to compensate for this is to have the drone go forward 30cm, then back 20cm, going a total of 10 cm. The Tello code accepts a command to go 21 cm. So it could be possible to go just 1 cm forward. But again, how precise the Tello drone is, is questionable.

Another issue met with the Tello drone, is the criteria needed for keeping its location in the air. To avoid drifting, it inspects the ground beneath and checks for movement of the texture. If the texture beneath is glossy or monotone, it cannot resolve if there is any movement. When flying above the white tables in the classroom it would have issues staying still and result in the drone starting to drift around. Luckily, the classroom floor has an Excelon Imperial texture which works well. As long there is enough light in the room to see the texture. It is also stated in the Tello Instructions that this can cause issues. (Monochrome surfaces and poor lighting) [18].

A final challenge that also was given by Teateret Vårt was to make the drone more quiet for the play, this was taken out of the challenge for the thesis. But could possibly be solved using toroidal propellers [12]

5.5 Python Code and Solutions

5.5.1 Container

The design for the container was mostly made straight forward starting in Blender software. With a CAD plugin it is not an issue to get everything to fit in scale. Measuring the Mini TX with a caliper and using those sizes to make the parts fit snugly. With the Prusa slicing software it is also possible to check that objects are to scale.

5.5.2 Software and Programming

There are a lot of ways to make the drone go from location A to location B. The most general/ simplest way could be to move the drone along the X-axis first, then Y- axis and lastly the Z - axis.

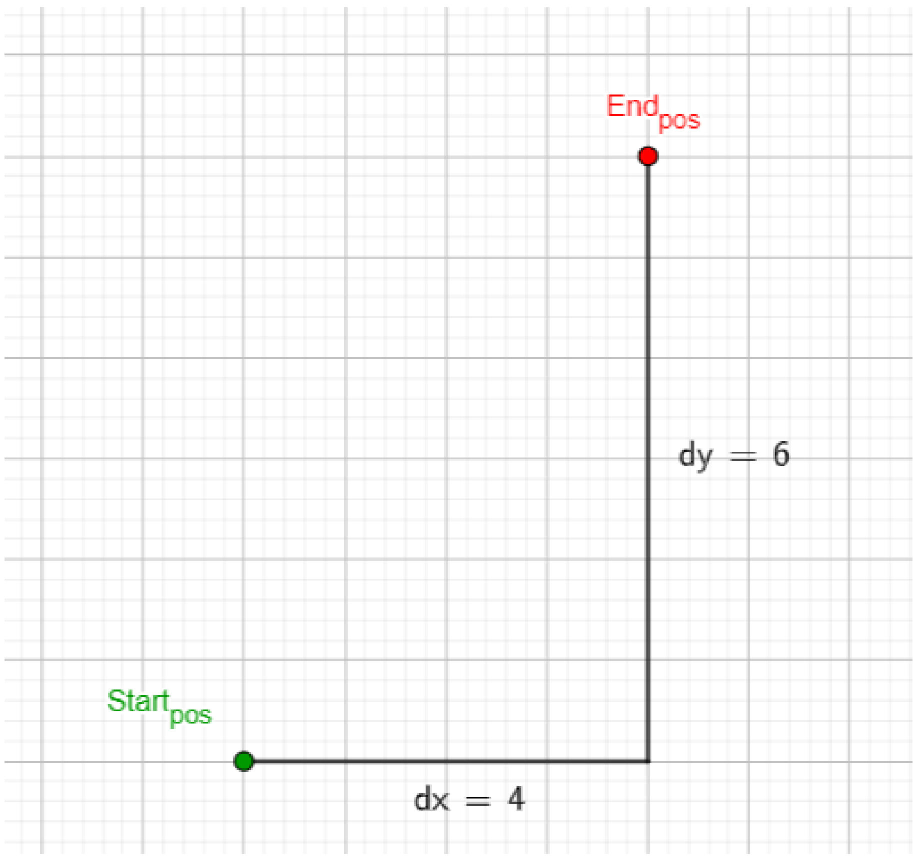


Figure 58: Visualization of simplest code, move one axis at a time

Figure 58 shows how the move from position A to B, one axis at a time (visualization is just with X and Y). This solution would remove orientation completely, as the drone is available to fly right, left, and backward without having to rotate toward the position.

The final result are 2 codes made to control the drone and both work as intended.

5.5.3 First Approach

The first code is working well, it finds the angle and distance between itself and the final coordinate and uses this to travel there. This is the quickest code as it goes straight from a to b blindly and when done it then checks its position. Issues with this approach is that it does not check midway. So it could miss-align and end up in the wrong position.

5.5.4 Second Approach

The second approach is more complicated because it looks at the previous angle it rotated, and then has to calculate by it to rotate to a new angle. Example: If the last angle was 30 degrees CCW, and the new coordinate is 60 degrees CW, it then needs to rotate a total of 90 degrees CW. This approach works just as well, it is slower but more accurate.

5.5.5 Issues and flaws with the code

Both of the codes will have issues if the drone is placed not facing towards *north* relative to the location system. If placed otherwise it will fail. This is a critical weakness in the code. A possible solution to find the angle it is oriented at start would be to read its location, drive forward 20cm, then read again. Comparing these to locations it can figure out the way the vector is pointing and then orient itself from this. This was never tried, but is believed to be a solution.

The Tello library used to control the drone gave limitations. One of the limitations the Tello drone has is the limit of how short and how far it can fly. It is limited to fly a minimum 20 cm and a maximum 500cm. For the first code, this was fixed by having it in a loop. If the distance from a to b was more than 5 meters, it would first fly those 5 meters then the rest. if it were more than 10 it would fly 5 meters twice then the rest. In the code, you can define the increments to shorter distances. For 20 cm it was not created a workaround, it was just accepted that the drone would be inside the area within a 20 cm range. Which is a OK solution because any shorter distances, the drone would maybe have issues to completely reach destined coordinates.

There were also issues with the Z variable for the local GPS. It was expected to be better, luckily there was also an acceptable solution that worked well for this. Using the Tello drone's distance sensor that is calculates its elevation from the ground beneath. It could then know how many meter above ground it was. This was used as variable *better_z* in the code. On stage in the theatre there is no props so the stage ground will be flat. This makes the solution acceptable. It there was a table 1 meter tall on stage, and the drone would be above it and it would read of its distance it would be wrong compared to ground truth.

5.6 Book Design

For the design of the book, there were two main challenges revolving around each other; being flyable and looking like a book. A design that solves both of these issues perfectly is difficult, as it is impossible to create a book replica with no visible feature of the drone and still fly. The approach to these challenges began with mostly testing what worked and what did not. The first design was quite optimistic about how much the Tello drone could lift as extra weight. The flight test using the pizza box (figure 34) showed that it was not that good at lifting extra. So therefore new designs were tested. The final design covers the drone on all sides and on top (not underside). This design hides the drone as much as possible, and also protects the actor and audience if it were to fail during live play.

Flaws with the design however makes the drone unstable and hard to control, it is most likely to be all the material that is mounted around the propeller on the outside. The total weight of the model would be to slimmed down.

So there is still room for optimizing this design even more. The book design still has a lot of potential for optimization. The model that was printed ended up being able to fly, but with poor control, as it would swing back and forth. In the final meeting with Teateret Vårt before the thesis deadline, it was given feedback that they wanted the book to fly with the cover.

A final design acknowledging this request was modeled:

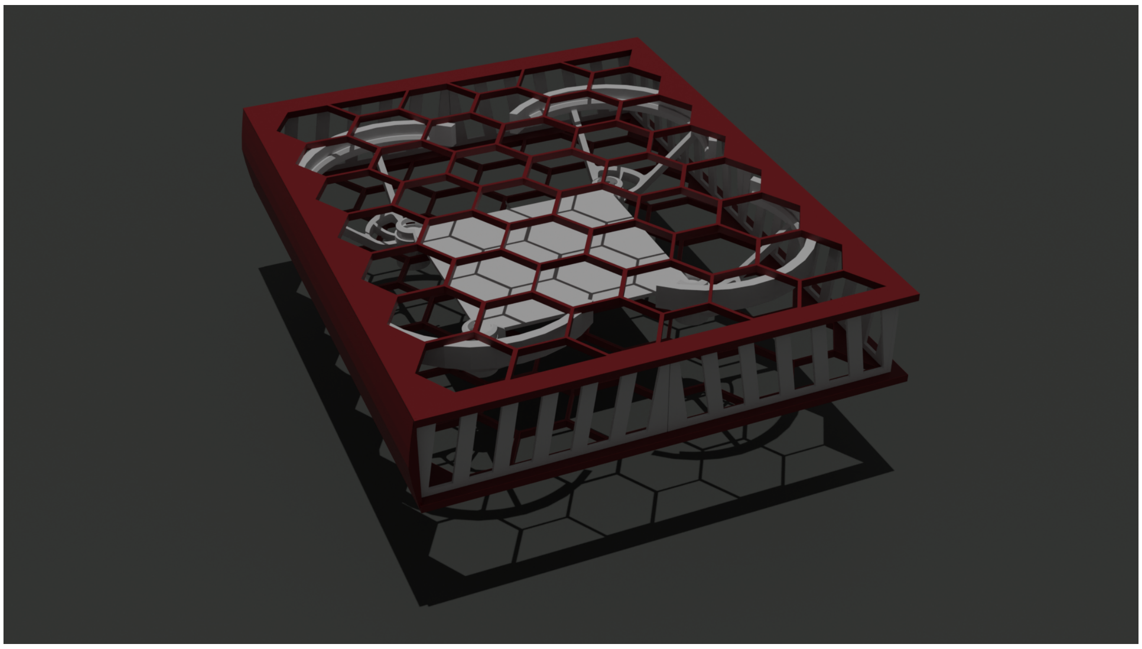


Figure 59: Computer rendering of book design.

Figure 59 is a model made as the final adjustment from Teateret Vårt, however, this design is neither printed nor tested for flight.

With more time or resources, a design could be a book with an implemented drone that was built into it.

6 Conclusion

The objective of this project was to incorporate drones into a live theater performance. To do this, different indoor location systems were tested to track and control the drone on the stage. Multiple scripts to control the drone automatically have been created, making the drone able to go between different locations on stage without human interaction. A final challenge was to create a disguise for the drone, attempting to hide it as a book prop for the audience.

This thesis has also given insight into the different location technologies that are possible and the pros and cons of each of them. Using UWB distributed by Pozyx, the connection for tracking is reliable but not very precise. Another pro with UWB is, in theory, that it is better at longer ranges. Using Marvelmind's ultrasound for the location system, however, gave better results in precision, but would easily fail with obstacles in between sensor and anchor.

Two scripts have been created to control the drone between coordinates. As there are multiple ways to do so, the ones that are created in this thesis make it possible to autonomously fly the drone between coordinates on stage of the theatre. Each of the scripts does have their strengths. As the first one is quick to move but less precise. The second, however, is more precise, but slower.

On the balance of artistic design and technical functionality, the project has made more progress in the technical department. Creating multiple scripts able to successfully control the drone, while disguising the book still needs more development before it can be taken into a live play.

Setting up the indoor GPS and controlling the drone within it, has given a lot of insight into how to fuse the systems together, as both Tello and Marvelmind has their area of usage and limitations. The project has also given a lot of insight into Python programming.

Creating the disguise for the drone has also been a difficult challenge. The drone has strict requirements that it needs to fulfil to be able to fly, such as with extra weight and airflow. Matching these requirements while also creating an authentic costume for the drone was paradoxical. However, a potential solution has been made, but it does still need optimization to be reliable enough for live theatre.

For further development of this project, it would be best to change to a more powerful drone. A drone that struggles less with lifting extra weight would make the prop design around the drone easier. Using a more powerful drone, however, would then be heavier and cause new issues with safety and flight time.

Additionally, not being locked to the limited control commands the Tello Library distributed, would result in more dynamic flight control. A possible solution would be to use a custom drone equipped with a different flight controller. A feasible solution would be to use ArduPilot software, an open software used to control drones autonomously. ArduPilot is compatible with both Marvelmind [2] and Pozyx [3] to control the drone. However, to use ArduPilot for a control system, other hardware (than what the Tello is equipped with) is needed for controlling the drone. Therefore it would be better to build the drone in prospect for its role in the play.

In final conclusion, it is possible to create flying props for theatre, but it does require concrete design, hardware, and software for its assignment. The design of the drone is going to be very dependent on its purpose in the play. The design of the drone and prop is dependent be its flight time, size, and control. Also sound, safety, and movement on stage.

Furthermore, there are other potential uses for indoor GPS-controlled drones, such as inspection and maintenance. If a drone is equipped with cameras and sensors can then be used to inspect places that are hard-to-reach. Saving time and resources. For example, a cathedral with a high roof where scaffolding would be difficult to set up.

Another potential is mapping and navigation, using the drone to create a 3D computer visualization of an area. It could possibly assist in architecture and emergencies. A possibility could be a scenario with a house fire creating dense smoke, using the UWB to locate the drone. Creating a 3D mapping for the firefighters outside. Another possible example is using drones for inventory management, using drones flying in storage facilities or warehouses to scan items available in the inventory.

Bibliography

[1] Albisay. *DJI Tello propeller guard*. 2023. URL: https://www.thingiverse.com/thing:4339771/files (visited on 20th Apr. 2023).

[2] Ardupilot. *Marvelmind for Non-GPS navigation*. 2023. URL: https://ardupilot.org/copter/docs/common-marvelmind.html (visited on 20th June 2023).

[3] Ardupilot. *Pozyx for Non-GPS Navigation*. 2023. URL: https://ardupilot.org/copter/docs/common-marvelmind.html (visited on 20th June 2023).

[4] Adrian Carrio et al. *IEEE: Onboard Detection and Localization of Drones Using Depth Maps*. 2020. DOI: 10.1109/ACCESS.2020.2971938.

[5] N.S. Correal et al. *IEEE: An UWB relative location system*. 2003. DOI: 10.1109/UWBST.2003.1267871.

[6] George Dekoulis. *Drones Applications*. URL: https://books.google.no/books?hl=en&lr=&id=_XaQDwAAQBAJ&oi=fnd&pg=PA3&dq=drones+book&ots=UAoH7o0mp0&sig=xKy1xPMXVkqHumgdPnem28lqbbQ&redir_esc=y#v=onepage&q&f=false (visited on 20th June 2018).

[7] Dragonfly.cv. *What does 6 dimentions of freedom mean?* URL: https://dragonflycv.com/what-does-6-degrees-of-freedom-or-6-dof-mean/# (visited on 20th June 2023).

[8] DroneBlog. *How much can a drone lift*. 2023. URL: https://www.droneblog.com/drone-payload/ (visited on 20th June 2023).

[9] Marvelmind. *Marvelmind Starter Set*. 2023. URL: https://marvelmind.com/product/starter-set-nia-03-smalldrone/ (visited on 15th Apr. 2023).

[10] Marvelmind. *Mini-TX Beacon*. 2023. URL: https://marvelmind.com/product/mini-tx/ (visited on 27th Apr. 2023).

[11] Marvelmind. *Super-Beacon*. 2023. URL: https://marvelmind.com/product/super-beacon/ (visited on 1st May 2023).

[12] MIT. *Toroidal propellers*. 2022. URL: https://www.ll.mit.edu/sites/default/files/other/doc/2023-02/TVO_Technology_Highlight_41_Toroidal_Propeller.pdf (visited on 15th June 2023).

[13] Pozyx. *The Creator One Kit*. 2023. URL: https://www.pozyx.io/creator-one-kit (visited on 12th Apr. 2023).

[14] Prusa. *Prusa Mini*. 2023. URL: https://www.prusa3d.com/product/original-prusa-mini-semi-assembled-3d-printer-4/ (visited on 25th Apr. 2023).

[15] Prusa. *Prusa Slicer Software*. 2023. URL: https://www.prusa3d.com/page/prusaslicer_424/ (visited on 28th Apr. 2023).

[16] Ahmed El-Rabbany. *Introduction to GPS*. 2001. URL: https://books.google.no/books?hl=en&lr=&id=U2JmghrrB8cC&oi=fnd&pg=PR13&dq=Gps+systems&ots=9NEUjQWGHU&sig=C6J79It8NzCLhfQP2n5bsWpZRF8&redir_esc=y#v=onepage&q=Gps20systems&f=false (visited on 1st May 2023).

[17] Ryze. *Tello*. 2023. URL: https://www.ryzerobotics.com/tello (visited on 5th Apr. 2023).

[18] Ryze. *Tello User Manual*. 2018. URL: https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20User%20Manual%20v1.4.pdf (visited on 5th Apr. 2023).

[19] Store Norske Leksikon SNL. *Drone*. 2023. URL: https://snl.no/drone (visited on 5th Apr. 2023).

[20] Store Norske Leksikon SNL. *Sult*. URL: https://snl.no/Sult (visited on 2nd Apr. 2023).

[21] Onur Tekdas and Volkan Isler. 'IEEE, Sensor Placement for Triangulation-Based Localization'. In: *IEEE Transactions on Automation Science and Engineering* 7.3 (2010), pp. 681–685. DOI: 10.1109/TASE.2009.2037135.

[22] Al-Furat Al-Awsat Technical University. *An Overview of Local Positioning System: Technologies, Techniques and Applications*. URL: https://core.ac.uk/download/pdf/287743886.pdf (visited on 20th May 2023).

Appendix

A Mocking of Marvelmind

```
    import random
import time
import ast

import math as m


def generate_coordinates():

    # Generate random coordinates with two decimal places

    x = round(random.uniform(15, 25), 2)

    y = round(random.uniform(15, 25), 2)

    return [x, y]



def print_current_coordinates(): #mocking av marvelmind coordinater

    # Generate and print current coordinates

    coordinates = generate_coordinates()

    print("Current coordinates: x =", coordinates[0], "y =", coordinates[1])
    return coordinates


def create_direction(my_pos, wanted_pos):
    x1, y1 = my_pos
    x2, y2 = wanted_pos

    dx = x2 - x1
    dy = y2 - y1

    distance = m.sqrt(dx**2 + dy**2)
    angle = m.atan2(dy, dx)
    angle_deg = m.degrees(angle)  # radians to degrees
    #print (angle_deg, 'original angle')

    o_angle = angle_deg
    rotation = ''

    angle_deg = (angle_deg + 360) % 360  # angle is between 0 and 360
    #print(angle_deg, 'new angle')
    #print(abs(angle_deg), 'abs of angle')
    #print(abs(angle_deg - 90), 'v-90')

    if 0 < o_angle < 90: #between 0 and 90 deg
        rotation = 'CW1'

    if -90 < o_angle < 0: #between -90 and 0
        rotation = 'CW2'
        #print(abs(angle_deg), 'abs')
        #angle_deg = 360 - angle_deg

    if -180 < o_angle < -90: #between -180 and -90
        rotation = 'CCW1'

    if 90 < o_angle < 180:  #between 90 and 180
        rotation = 'CCW2'

    if angle_deg < 0 : #if degrees is lower than 0  #0 = angle_deg < 0:
        angle_deg = abs(angle_deg-90)
        #print (angle_deg, 'okok')
    if angle_deg > 180 :  # if degrees is above 180  0: angle_deg > 180
    #elif -180 > angle_deg > 180:
        angle_deg -= 90

    elif angle_deg < 180:     #if degrees is more than 0 and less than 180
        angle_deg = abs(angle_deg - 90)
        #print("alå")

    #print(angle_deg , 'v2')

    if rotation == 'CW2':
        angle_deg = 360 - angle_deg

    return distance, angle_deg, rotation

###########MAIN#####################

def main():
    print("Enter the coordinates as 'x, y'. Press Enter without any input to continue.")

    #coordinates = generate_coordinates()
    while True:
        user_input = input()
        numbers = list(map(float, user_input.split()))
        if len(numbers) != 2:
            print ("error, print two numbers")
        else:
            new_pos = numbers
            print('Selected coordinates', 'x=', new_pos[0], 'y=', new_pos[1])
        coordinates = print_current_coordinates()
        new_distance, new_angle, orientation = create_direction(coordinates, new_pos)

        print ('distance=', round((new_distance), 2), 'angle=', round((new_angle), 2),
        ↪  'orientation=', orientation)

        time.sleep(0.1)


if __name__ == "__main__":
    main()

⋮
```

B Coordinate Deviation Filter

```
    def precision_coordinates(hedge):
    number_of_precision = 3 #decides the amount of checks
    pos_array = [] # Create an empty list to store positions
    for i in range(number_of_precision):
        hedge.dataEvent.wait()  # Wait for new data
        hedge.dataEvent.clear()

        pos_array.append((hedge.position()[1], hedge.position()[2]))  # Append position [1] and [2]
        ↪  to the list
    precise_pos = tuple((sum(coords) / number_of_precision for coords in zip(*pos_array)))
    print (pos_array)
    return (precise_pos)

⋮
```

C First Approach Code

```
import os
import sys
import math as m

master_folder = os.getcwd()

sys.path.append(os.path.join(master_folder, "Tello-master"))
sys.path.append(os.path.join(master_folder, "marvelmind.py-master\src"))

from marvelmind import MarvelmindHedge
import tello as tl

my_increments = 3                                       #Increments decide how many meters it
↪  should go before checking new pos. Maximun increments is 5 meters

z_height = 0


def get_current_coordinates(hedge):
    hedge.dataEvent.wait()                              #Wait for new data
    hedge.dataEvent.clear()
    #z_height = tl.get_tof()/100                        #Tello height readoff
    return hedge.position()[1], hedge.position()[2], hedge.position()[3]

def create_direction(my_pos, wanted_pos):
    x1, y1 = my_pos
    x2, y2 = wanted_pos

    dx = x2 - x1
    dy = y2 - y1

    distance = m.sqrt(dx**2 + dy**2)
    angle = m.atan2(dy, dx)
    #print(angle, 'angle radiants')
    angle_deg = m.degrees(angle)                         #Radians to degrees
    rotation = ''
    #print (angle_deg, 'original angle')

    o_angle = 90 - angle_deg                             #rotate angle 90 deg ccw
    #print (o_angle, 'angle-90')
    if 180 > o_angle > 0:
        rotation = 'CW'
        #print(o_angle, 'new angle')

    if o_angle > 180:
        o_angle = 360 - o_angle
        rotation = 'CCW'
        #print (o_angle, 'new angle')

    if o_angle < 0:
        o_angle = (abs(o_angle))
        rotation = 'CCW'

    return distance, o_angle, rotation

def find_z(my_pos, wanted_pos):
    z1 = wanted_pos
    z2 = my_pos

    z_diff = round(- (z2 - z1),2) #get two decimals
    return z_diff

def Tello_Commands_Z(elevation):
    if elevation > 0.2: #positive value, upwards
        tl.up(elevation*100)

    elif elevation < -0.2: #negative value, downwards
        tl.down(abs(elevation)*100)

    elif 0.2 > elevation > -0.2:
        print('staying level')


def Tello_Commands(distance, rotation, orientation):
    my_rotation = rotation
    if orientation == 'CCW':                                    #Rotate angle towards point
        tl.anticlockwise(rotation)
    else:
        tl.clockwise(rotation)

    if distance > my_increments:                                #Drone can with this if loop fly longer
    ↪  than 5 meters.
        increments = distance // my_increments                  #how many increments goes into the
        ↪  distance. 15meter in 5 increments = 3 increments
        print(increments, 'increments *', my_increments)                        #increments can be
        ↪  defined to how often it stops. max lenght of increment is 5 meters and shortest is 20cm.
        for i in range(int(increments)):
            tl.forward(my_increments*100)
            #x, y = get_current_coordinates(hedge)

        mudolo = round(distance % my_increments, 2)
        print(round((mudolo*100),2), 'cm mudolo')
        if mudolo > 0.2:                                        #if mudolo values are more than 20 cm
            tl.forward(mudolo*100)
        else:
            print('staying')

    else:
            tl.forward(distance*100)                            #distance = less than 5 meters. go
            ↪  distance in cm


    if orientation == 'CCW':                                    #Rotate back again to '0'
        tl.clockwise(rotation)
    else:
        tl.anticlockwise(rotation)


def main():
    tl.start()
    print(tl.get_battery(), 'percentage of battery left')

    hedge = MarvelmindHedge(tty="COM8", adr=None, debug=False)  #Create MarvelmindHedge object

    if len(sys.argv) > 1:
        hedge.tty = sys.argv[1]

    hedge.start()  # Start receiving data from marvelmind

    tl.takeoff()

    try:
        while True:
            user_input = input("Please Enter new coordinates...")
            numbers = list(map(float, user_input.split()))
            if len(numbers) != 3:
                print('Error')

            else:
                new_pos = numbers
                print('Selected Coordinates; x=', new_pos[0], 'y=', new_pos[1], 'z=', new_pos[2])

            current_pos = get_current_coordinates(hedge)
            print("Current coordinates; x=", current_pos[0], "y=", current_pos[1], 'z=',
            ↪  current_pos[2])

            current_pos_xy = (current_pos[0], current_pos[1])   #cord of hedge,   only x and y.
            current_pos_z = tl.get_tof()/1000                   #read height of tellodrone, only z
            print(current_pos_z, 'CYPHER 1')

            wanted_pos_xy = (new_pos[0], new_pos[1])            #cord of new pos, only x and y
            wanted_pos_z  = (new_pos[2])                        #last value of input coordinates
            z_difference = find_z(current_pos_z, wanted_pos_z)
            print(z_difference, 'CYPHER 2')

            my_Distance, my_Angle, my_Orientation = create_direction(current_pos_xy, wanted_pos_xy)
            ↪   #Create path

            print('distance=', round((my_Distance), 2), 'angle=', round((my_Angle), 2),
            ↪   'orientation=', my_Orientation, 'Elevation:', z_difference,'m')

            question_input = input('GO TO COORDINATES? y for yes')  #Doublecheck of commands, This
            ↪   was learned the hard way.
            if question_input == 'y':
                print('GOING TO COORDINATES')
                Tello_Commands(round((my_Distance), 2), my_Angle, my_Orientation)
                Tello_Commands_Z(z_difference)
                while True:
                    while True:
                        x, y, z = get_current_coordinates(hedge)
                        current_pos = (x, y, z)
                        better_z = tl.get_tof()/1000

                        distance_to_target   = m.sqrt((new_pos[0] - current_pos[0])**2 + (new_pos[1]
                        ↪   - current_pos[1])**2)    #Check distance between wanted coord. and new
                        ↪   coord.
                        distance_to_target_z = find_z(better_z, wanted_pos_z)

                        print ('direct distance to target xy:', distance_to_target, 'z',
                        ↪   distance_to_target_z)

                        if distance_to_target <= 0.2:
                        ↪   #Check if distance within threshold
                            print('Im at x=', x, 'y= ', y,)
                            print('Acceptable, Reached xy coordinates.')
                            break

                        else:
                            print('Im at x=', x, 'y= ', y, 'and z=', z )
                            print('Not yet reached the coordinates. Trying again...')
                            current_pos = get_current_coordinates(hedge)
                            current_pos_xy = (current_pos[0], current_pos[1])

                            my_Distance, my_Angle, my_Orientation = create_direction(current_pos_xy,
                            ↪   wanted_pos_xy)
                            Tello_Commands(round(my_Distance, 2), my_Angle, my_Orientation)

                    while True:
                        if 0.2 > distance_to_target_z > -0.2:
                            print('z='  , better_z,)
                            print (distance_to_target_z,'acceptable z value, staying level')
                            break

                        else:
                            better_z = tl.get_tof()/1000
                            print('wrong z altitude, im at:', better_z,'m')
                            z_difference = find_z(current_pos_z, wanted_pos_z)
                            Tello_Commands_Z(z_difference)
                            #Continue the loop and check again

                    if distance_to_target <= 0.2 and 0.2 > distance_to_target_z > -0.2:
                        print('In both xy and z, finished')
                        break

            else:
                print('not ok, LANDING')
                tl.land()

            x, y, z = get_current_coordinates(hedge)
            print('NEW COORDINATES x=',x, 'y=', y, 'and z =', z)

    except KeyboardInterrupt:
        hedge.stop()  # stop receiving data

if __name__ == "__main__":
    main()
```

⋮

D Second Approach Code

```
    import os
import sys
import math as m

master_folder = os.getcwd()

sys.path.append(os.path.join(master_folder, "Tello-master"))
sys.path.append(os.path.join(master_folder, "marvelmind.py-master\src"))

from marvelmind import MarvelmindHedge
import tello as tl

my_increments = 1  # Increments decide how many meters it should go before checking new pos. Maximum
↪  increments are 5 meters

old_angle = 0
z_height = 0
old_orientation = ''

final_angle = 0


def get_current_coordinates(hedge):
    hedge.dataEvent.wait()  # Wait for new data
    hedge.dataEvent.clear()
    return hedge.position()[1], hedge.position()[2], #x y
    print('im at: x', hedge.position()[1],'y', hedge.position()[2])

def calculate_angle(my_pos, wanted_pos):
    x1, y1 = my_pos
    x2, y2 = wanted_pos

    dx = x2 - x1
    dy = y2 - y1

    angle = m.atan2(dy, dx)
    #print(angle, 'angle radiants')
    angle_deg = round(m.degrees(angle), 2)                              #Radians to degrees

    rotation = ''
    #print (angle_deg, 'original angle')

    o_angle = 90 - angle_deg                                  #rotate angle 90 deg ccw
    #print (o_angle, 'angle-90')
    if 180 > o_angle > 0:
        rotation = 'CW'
        #print(o_angle, 'new angle')

    if o_angle > 180:
        o_angle = 360 - o_angle
        rotation = 'CCW'
        #print (o_angle, 'new angle')

    if o_angle < 0:
        o_angle = (abs(o_angle))
        rotation = 'CCW'

    return round((o_angle),2), rotation


def calculate_distance(my_pos, wanted_pos):
    x1, y1 = my_pos
    x2, y2 = wanted_pos
    dx = x2 - x1
    dy = y2 - y1

    distance = round(m.sqrt(dx**2 + dy**2), 2)
    return distance


def rotate_towards_angle(angle, orientation):
    if angle > 0:
        if orientation == 'CCW':
            tl.anticlockwise(angle)

        else:
            tl.clockwise(angle)

    else:
        print('angle = 0, straigt ahead')

def second_angle(new_angle, old_angle, orientation, last_orientation):
    if orientation == 'CW' and last_orientation =='CW':
        if new_angle < old_angle:
            angle = round(abs(new_angle - old_angle),2)
            print('1new_angle -old_angle')
            orientation = 'CCW'
        else:
            angle = round(abs(new_angle - old_angle),2)
            print('12new_angle -old_angle')
            orientation ='CW'

    elif orientation == 'CCW' and last_orientation =='CW':
        angle = round(abs(new_angle - old_angle),2)
        print('2new_angle - old_angle')
        orientation = 'CW'

    elif orientation == 'CCW' and last_orientation == 'CCW':
        if new_angle < old_angle:
            angle = round(abs(new_angle - old_angle),2)
            print('21new_angle -old_angle')
            orientation = 'CCW'
        else:
            angle = round(abs(new_angle - old_angle),2)
            print('22new_angle -old_angle')
            orientation ='CCW'

    elif orientation == 'CW' and last_orientation =='CCW':
        angle = round(abs(new_angle - old_angle),2)
        print('3new_angle - old_angle')
        orientation = 'CW'

    return angle, orientation

def move_towards_coordinate(distance):
    if distance > 0.75:                  #if further than 50cm, just go 50 cm
        tl.forward(75)
        print('going forward 75cm')
    else:
        tl.forward(distance*100)        #if shorter, go complete distance


def read_z():
    zpos = tl.get_tof()/1000
    return zpos


def find_z(my_pos, wanted_pos):
    z1 = wanted_pos
    z2 = my_pos

    z_diff = round(- (z2 - z1),2) #get two decimals
    return z_diff

def Tello_Commands_Z(elevation):
    if elevation > 0.2: #positive value, upwards
```

```
        tl.up(elevation*100)

    elif elevation < -0.2: #negative value, downwards
        tl.down(abs(elevation)*100)

    elif 0.2 > elevation > -0.2:
        print('staying level')

def calc_totangle(totangle, rotangle, orientation):
    if orientation == 'CCW':
        totangle = totangle - rotangle
    else:
        totangle = totangle + rotangle
    return totangle

def main():
    tl.start()
    print(tl.get_battery(), 'percentage of battery left')

    hedge = MarvelmindHedge(tty="COM8", adr=None, debug=False)  # Create MarvelmindHedge object

    if len(sys.argv) > 1:
        hedge.tty = sys.argv[1]

    hedge.start()  # Start receiving data from Marvelmind

    tl.takeoff()

    final_angle = 0

    try:
        while True:
            user_input = input("Please enter new coordinates (x y z): ")
            numbers = list(map(float, user_input.split()))
            if len(numbers) != 3:
                print('Error: Invalid input format')
                continue

            target_pos = numbers
            target_pos_xy = (target_pos[0], target_pos[1])
            target_x, target_y, target_z = numbers
            print('Selected Coordinates: x =', target_pos[0], 'y =', target_pos[1])

            while True:
                current_pos = get_current_coordinates(hedge)
                print("Current pos: x =", current_pos[0], "y =", current_pos[1],)
                current_pos_z = read_z()
                print('z = ', current_pos_z)

                angle, orientation = calculate_angle(current_pos, target_pos_xy)
                print('a:', angle, 'o:', orientation)

                distance = calculate_distance(current_pos, target_pos_xy)
                print('d=', distance)

                z_diff = find_z(current_pos_z, target_pos[2])

                rotate_towards_angle(angle, orientation)
                move_towards_coordinate(distance)
                Tello_Commands_Z(z_diff)

                print('done commands')
                final_angle = calc_totangle(final_angle, angle, orientation)
                print(final_angle, 'final angle')
                old_angle = angle
                old_orientation = orientation

                while True:
                    current_pos = get_current_coordinates(hedge)
                    print('current pos:', current_pos)
                    current_pos_z = read_z()
                    print('z = ', current_pos_z)
                    angle, orientation = calculate_angle(current_pos, target_pos_xy)
                    distance = calculate_distance(current_pos, target_pos_xy)
                    print('a:', angle, 'o:', orientation)
                    z_diff = find_z(current_pos_z, target_pos[2])

                    rotangle, orientation = second_angle(angle, old_angle, orientation,
                    ↪  old_orientation)
                    rotate_towards_angle(rotangle, orientation)
                    Tello_Commands_Z(z_diff)


                    print(angle, 'relative angle')
                    print(rotangle, 'differation_angle', orientation)
                    print(distance, 'distance')
                    final_angle = calc_totangle(final_angle, rotangle, orientation)
                    print('finalangle:', final_angle)

                    if distance > 0.2:
                        move_towards_coordinate(distance)

                        print('done commands2')
                        old_angle = angle
                        old_orientation = orientation
                        print(old_angle, old_orientation, 'OA and OO')

                    if distance < 0.2:
                        print('Reached the target coordinates')
                        print(final_angle, 'rotating back to north')
                        if final_angle < 0:
                            tl.clockwise(abs(final_angle))
                        else:
                            tl.anticlockwise(abs(final_angle))
                        break
                    print('Distance to target:', distance)

                break

    except KeyboardInterrupt:
        hedge.stop()  # Stop receiving data

if __name__ == "__main__":
    main()
```