POLITECNICO DI MILANO

School of Industrial and Information Engineering

Master of Science in Electrical Engineering

PROGRAMMING OF AN INDOOR AUTONOMOUS DRONE AND METROLOGICAL CHARACTERIZATION

Supervisor: Prof. Simona Salicone
Co-Supervisor: Prof. Alessandro Ferrero

Candidate:
Fereshte Bagher Oskooei
Matr.904138

Academic Year 2019-2020

Acknowledgement

Throughout my thesis, I have received a lot of support from whom I would love to thank for.

I would first like to thank my supervisors Professor Salicone and Professor Ferrero for always guiding me with their valuable insights and encouraging me to deliver the best possible results.

I would also like to thank my family and best friends who were always there to lend a listening ear and support me when I needed it.

Contents

List of Abbreviations

UAV	*Unmanned Aerial Vehicle*
UGV	*Unmanned Ground Vehicle*
GPS	*Global Positioning System*
IMU	*Inertial Measurement Unit*
EKF	*Extended Kalman Filter*
ViCon	*Variable Independent Control*
RRT	*Rapidly Exploring Random Tree*
APF	*Artificial Potential Field*
ArUco	*Augmented Reality University of Cordoba*
SDK	*Software Development Kit*

List of Figures

List of Tables

Abstract

In recent years, quadcopter unmanned aerial vehicles (UAVs) have become so widespread and advanced that they are not only used for entertainment but also outdoor autonomous applications such as parcel delivery. However, the same cannot be said for indoor environments due to a lack of global positioning system (GPS) availability. In this Thesis, the goal is to get an accurate understanding of such a quadcopter's metrological characteristics and use them for programming it towards autonomy in indoor environments without the need for further expensive equipment. What are the metrological limits of a quad-copter equipped with a low-cost inertial measurement unit (IMU)? and how can we use it for tasks such as route following in a room?

In order to understand the accuracy and limits of tools such as sensors for altitude and acceleration and technologies like image processing and attitude estimation used in AR.Drone 2.0, the quadcopter in question, tests were run and their data were analyzed. Later, using the results from the previous tests, the drone was programmed to use its image processing technology to follow the prepared path on the floor of a room without pilot interaction. The results indicated that with low-cost IMU and primary technologies embedded in an AR.Drone, it is possible to have a relatively low-cost autonomous route following drone for indoor environments.

Sommario

Negli ultimi anni, i veicoli aerei senza pilota (UAV) del tipo quadricotteri sono diventati così diffusi ed avanzati che non vengono utilizzati solo per l'intrattenimento ma anche per applicazioni autonome all'aperto, come ad esempio la consegna di pacchi. Tuttavia, lo stesso non si può dire per gli ambienti interni a causa della mancanza di disponibilità del sistema di posizionamento globale (GPS). In questa Tesi, l'obiettivo è quello di ottenere una comprensione accurata delle caratteristiche metrologiche di un quadricottero ed utilizzarle per programmarlo in modo che possa volare in autonomia in ambienti interni, senza la necessità di ulteriori costose apparecchiature. Quali sono i limiti metrologici di un quadricottero dotato di un'unità di misura inerziale (IMU) a basso costo? e come possiamo usarlo per il nostro obiettivo, come ad esempio seguire il percorso in una stanza?

Diversi test sono stati eseguiti ed i risultati ottenuti sono stati analizzati, al fine di comprendere l'accuratezza ed i limiti degli strumenti montati sul drone AR.Drone 2.0 (il quad-elicottero in questione) e delle tecnologie in esso implementate: come i sensori di altitudine, i sensori di accelerazione, l'elaborazione delle immagini e la stima dell'assetto. Successivamente, utilizzando i risultati di tali test, il drone è stato programmato in modo tale da utilizzare la sua tecnologia di elaborazione delle immagini per seguire un determinato percorso preparato sul pavimento di una stanza, senza l'interazione del pilota. I risultati hanno indicato che, con una IMU a basso costo e con le tecnologie primarie incorporate nell' AR.Drone, è possibile far seguire al drone un percorso autonomo in ambienti interni.

Chapter 1

Introduction

During the recent years micro unmanned aerial vehicles (mUAV) or drones have been developed in regards of maneuverability, stability and other capabilities. This jump in the technology has prompted a series of researches with the subject of drones and the challenges surrounding them. Most of these studies have a common general goal and that is to have the drone to do a particular task with less interaction from the pilot or at least with higher level commands.

Autonomy of robots in general has reached spectacular heights in recent years. Therefore, it is not out of expectation that flying robots also reach similar levels. With the advancement of electronics, computer science and mechanics that exist in today's engineering, it does not seem too long until our world looks like a page out of a science fiction book from 1990s.

1.1 A brief history on drones

When two brothers and a professor made the first quad-copter "Gyroplane No 1" in 1907, they probably did not think how far this mechanical bird would evolve in a hundred years. Jacques and Louis Brequet, with the help of Nobel prize winner Prof. Charles Richet, invented the most elementary drone that needed multiple people to balance and could not get very high. However, this embarked the idea of military drones at 1917. The idea of a

bomb that would fly towards the target on its own was very desirable. The first operational military drone was made by Germans in 1943 during world war II. "FritzX" a 2300-pound bomb which was used to sink ships while being controlled remotely [9].

Going back to quad-copters, on April 14, 1924 French engineer Étienne Oehmichen flew his quad-copter, which was the developed model of his second multi-copter design built in 1920, a distance of 360 meters setting a world record which was the Fédération Aéronautique Internationale (FAI) distance record for helicopters. In the same year he flew a 1 kilometer circle in 7 minutes and 40 seconds [12].

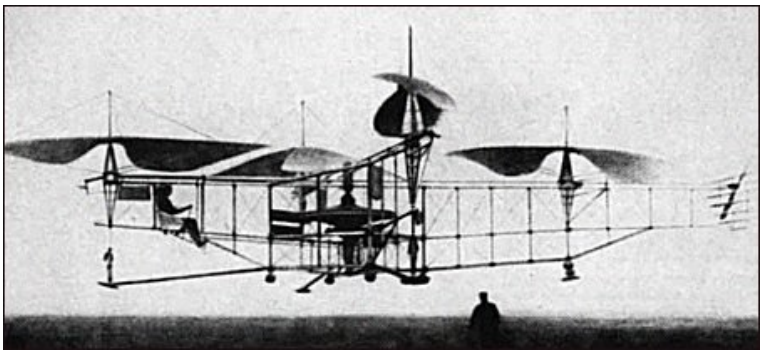

Figure 1.1: The Oehmichen drone

After Oehmichen, Dr. George de Bothezat and Ivan Jerome developed the aircraft in figure 1.2 for the US army, with six bladed rotors at the end of an X-shaped structure. This quad-rotor first flew in October 1922; followed by nearly 100 tests with the maximum altitude of 5 meters. Despite demonstrating feasibility, it was under-powered, unresponsive, mechanically complex and susceptible to reliability problems [2].

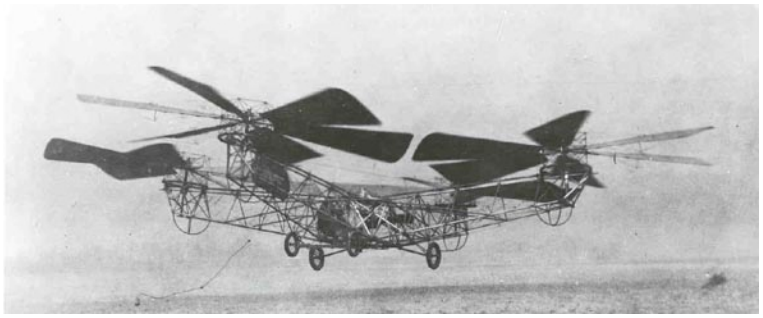

Figure 1.2: The de Bothezat aircraft

Convertawings Model "A" Quad-copter built in 1956 was designed with a broader range of applications in sight. However, due to lack of commercial or military orders, its project was terminated. The design featured two engines driving four rotors with wings added for additional lift in forward flight. No tail rotor was needed and control was obtained by varying the thrust between rotors. This prototype flew successfully many times during the mid 1950's and it was the first quad-copter to fly forward successfully [1].

Figure 1.3: Convertawings Model "A" Quad-copter

Following the development in the transistor industry in 1960s, the radio controlled components were able to be made in smaller sizes. Popularity of RC (radio controlled) planes surged specially in USA. They mostly came in kits for both indoors and outdoors applications. They were the early example of the consumer drones which would spread through the market half a century later.

In 2006, after considering the potential of non-military drones, the Federal Aviation Administration (FAA) in USA, issued the first drone permits for

commercial uses. Primarily there were not many requests for it; soon however, the numbers sped up.

The first ready to fly drone which was controlled by Wi-Fi through the pilot's smart phone, was introduced by the French company Parrot in 2010. The AR.Drone was considered a success very soon by customers and critics alike. Its successor AR.Drone 2.0, which is the subject of this thesis, was the improved and easier to fly version.

Three years later, Jeff Bezo, the founder of Amazon, showcased a drone delivery system. Although this was not the first time the idea was mentioned, it was the first time that the technology was brought into the public's audience. Following this, many corporations and postal organizations invested time and resources on drone based delivery which will be discussed further in this chapter.

In 2016, the company DJI's drone called phantom 4 was introduced to the market. This drone combines computer vision and machine learning to avoid obstacles and successfully track people, animals and objects autonomously [9].

Nowadays, drones from corporations like DJI or small startup companies like skydio have obstacle avoiding abilities as a given; so much that it is almost taken for granted at this point. They can track a single person as they are riding their bike in a mountain trail and dodge canopies and branches at the same time. Additionally, with the abilities of a drone like skydio 2, multiple drones can take off and operate together on an inspection without the human control in the loop. With all of these developments in capabilities of drones and the steps taken towards their automation, the range of their applications grows bigger every day [16] [20].

1.2 Application of drones in today's world

In manufacturing environments, tasks such as material handling and part feeding have been done by UGV (Unmanned Ground Vehicle) for years now. However, UAV (Unmanned Aerial Vehicle) is a fairly newer part of these environments. Equipped with an imaging device, drones can have a 360-degree inspection of an object in a three dimensional space. Also with sensors, they are easily able to sense any specific chemical in the environment and

report back immediately through wireless connections. Additionally, they only occupy the empty upper-air space which was left idle by UGV [21].

These aspects make drones a very suitable agent for inspection tasks, especially in harsh environments and situations such as nuclear power plants or search and rescue missions, where the drone helps the emergency crew plan the best and most secure mode to operate. For instance, a flying gas-sensor that could scout an environment and report back on the existence or lack thereof some target gases, without intervention of humans is very desirable [22]. Gases are only an example of sensors; any kind of measurement units can be planted on UAVs for similar missions.

However, these are not the only cases where drones are used indoors. Hospitals, green houses and production companies also employ UAVs for surveillance and logistics purposes. Additionally, autonomous mapping mechanisms have been implemented in fields like architecture and building managing for drones because of their low cost, high spatial resolution and versatility. Unknown structure interiors like abandoned mines are sometimes too risky for usual ways of mapping either due to harsh environment, space constraints or security reasons; therefore, UAVs are a much better choice [10].

In a study a quad-copter was used for mapping and monitoring of multiple floor buildings using an on-board inertial measurement unit (IMU) and camera plus a scanning laser range sensor retro-fitted with mirrors for beam redirection to the floor and ceiling serving as a primary source of information for position and yaw estimation. In this project two separate extended Kalman filters (EKF) were used for data fusion to increase the data frequency to 100 HZ. Additionally they used Vicon (Variable Independent Control) motion capture systems to get the accurate location and pose of the quad-copter and compare them with the estimated values [32].

Mentioned above was a part of UAV applications in closed environments, with denied access to GPS. However, outdoors, where this restriction does not exist, drones have extended uses including but not limited to: traffic monitoring, crowd monitoring [29], urban planning, civil security applications and drone delivery systems.

Some studies [6] describe a path planning algorithm for a quad-copter based delivery system. This proposed algorithm was a combination of sampling based Bidirectional RRT algorithm and improved Artificial potential field

(APF) algorithm, so that the former is made more goal oriented. This algorithm was dependant on the information from the environment by the sensors to improve the pre-planned path and to decrease the execution time.

More about delivery systems, after Amazon showcased a drone based one, their project called Prime Air now has development centers in USA, UK, Austria, France and Israel. They are working on making a system that is efficient, stable and most of all safe for everyone [3].

Other than Amazon, Corporations such as Walmart, Google, UPS and many other postal organizations are working on a similar means of delivery. In the meantime, some are already using such a system. Project Wing is now developing an unmanned traffic management platform that will allow drones navigate around other unmanned or manned aircraft, humans and any other obstacles. Wing delivery drone is operating in Helsinki, Finland and Canberra, Australia.

On October 1st 2019, UPS Flight Forward Inc. (a subsidiary of UPS) announced it has received the U.S. government's first full Part 135 Standard certification to operate a drone airline. The company will initially expand its drone delivery service further to support hospital campuses around the country. UPS drones will also provide solutions for customers beyond those in the healthcare industry. UPS Flight Forward plans in the future to transport a variety of items for customers in many industries, and regularly fly drones beyond the operators' visual line of sight.

However, drone delivery for medical purposes is not a new concept in Africa. Rwanda is a country in central Africa which is sometimes called "Pays des mille collines" in French ("Land of a thousand hills"). Because of this, some rural areas are separated by long stretches of roads despite being close to one another. In May 2016, the first commercial drone delivery system started in Rwanda by delivering blood and blood samples from and to multiple medical care centers. Zipline, the US Robotics and Drones company, provides and manages this system. Each drone delivery flight takes less than thirty minutes for a trip that would take a motorcycle, hours; but costs the same amount. This project has saved many lives up to now since blood loss after birth and maternal mortality used to be very common in this country. After Rwanda, Zipline has made a similar contract with the government of Ghana in April 2019 to fly thirty drones out of four distribution centers to 2000

health centers. This projects will do 600 flights a day and serve 12 million people [8].

Apart from practical applications for drones, they have been the subject of more entertaining and free-topic researches. These studies although seem more about the entertainment at first, research the challenges surrounding the relatively new vehicles that are quad-copters. The studies that are done in these areas result in developing more stable quad-copters with more versatility, maneuverability and flight safety.

Take ball juggling for instance [25]. In an study, a quad-copter was used along with studying the flight pattern of balls and their interaction with rackets. In result, they were able to develop a quad-copter that would receive a thrown ball and hit it back towards the target with an attached racket. This would enable a human to play with a robot or two quad-copters playing with each other. In a similar study a method was developed in which a fleet of drones would be attached around a net and would all cooperate to catch and throw a ball using the net [27].

In another study a quad-copter was used to hold, balance and fly an upside-down pendulum of half a meter length. This pendulum was allowed to move up to 50 degrees on top of the quad-copter but after that the drone would intervene and move so that it would not fall. This research developed a control system that would take the drones further from their usual near hovering positions to more challenging and dynamic flights [14].

There also has been copious studies with AR.Drone as their test subject. Most of the autonomous navigation and mapping projects done using this particular drone have involved some external cameras or sensory system as a feedback loop for stabilizing and calibrating the attitude estimations done by the drone [28] [5].

Few particular studies however use the vision tools included in the drone for this purpose. In one work a Kalman filter was used to fuse the pose data already estimated by the drone and the data the drone gets from a particular marker seen in the camera to get a better estimate of the drone's pose and location. Using this autonomous control system they have been able to develop autonomous flight plans which include but are not limited to hovering in place and coming back to the designated position after external disturbances [31]. In another similar research an ArUco marker was used for

the drone to find the target spot for landing. The algorithm is so that the data from front camera is fused with IMU sensor data in a Kalman filter to navigate the drone towards the marker when it is in sight [30].

1.3 Objective of this thesis

As mentioned previously, currently there are drones that can autonomously follow or track humans or objects in outdoors and although in GPS-denied environments such as indoors, this autonomy faces more challenges, there are still means to realize them. Motion capture systems, scanning lasers and many other tracking systems such as these can be utilized to develop autonomous unmanned aerial vehicles. These systems and drones however can be quite costly and realizing them could be most complicated.

The main goal of this thesis was to develop a navigation system for an autonomous low-cost quad-copter. This drone would have to rely on its own internal low-cost inertial measurement unit to estimate the position and follow a pre-planned path. The drone would have no prior knowledge of the indoor environment and with no extra sensory data from expensive motion tracking systems, it would have to know when to turn or go straight.

To explain more in detail, the quad-copter in question would be given a prepared program before taking off and then it would not receive further demands from the pilot until the very end that it had gone through the mission completely. In particular this program would urge the drone to go forward a decided length along the X axis and compensate for any deviation to right or left (Y axis). Secondly it would need to decide which way to turn and how to continue its path according to the prepared environment. Therefore, it would continue a complete route in an indoor environment with no prior knowledge of the room all without human intervention.

To realize this goal, first one needs to get an understanding of the drone in question. In this thesis AR.Drone 2.0 has been the quad-copter in which the research is based on. Therefore, in order to apply an autonomous navigation algorithm on it, its metrological characteristics needed to be studied. It needed to be determined that how accurate the drone's estimated velocity and general attitude estimations were. How far the drone's sight is in different altitudes and questions as such. The answers to these were crucial for

reaching the main objective therefore they needed to be studied at first.

1.4 Thesis Outline

In the following the main chapters in this thesis are briefly described bellow.

- Chapter 2: In this chapter the flying principles of quad-copters in general is discussed briefly. Furthermore, the Parrot AR.Drone 2.0 in particular is expressed in detail. Its hardware and software features, internal control algorithms and image processing capabilities. Additionally a complete archive of the drone's output data and their definitions is provided.
- Chapter 3: This chapter is primarily dedicated to describing algorithms and processes for defining the metrological characteristics of the drone. This thesis mainly concentrates on the estimated linear horizontal velocities provided by the drone.
- Chapter 4 : In this chapter the algorithms from previous chapter are applied in numerous tests and their results are provided and analysed.
- Chapter 5: This chapter describes an algorithm for autonomous target approach flight also another algorithm for autonomous pre-planned route tracking with minimal human interaction in the beginning and end of the path using the results from previous chapter.
- Chapter 6: This chapter includes the result and feasibility proof of applied algorithms from previous chapter.
- Chapter 7: In this chapter the conclusion of the whole research is provided along with suggested future work ideas.

Chapter 2

Parrot AR.Drone 2.0 power edition

AR.drone 2.0 was introduced in 2012 in Las Vegas by the French company Parrot. In comparison to AR.drone 1.0, the design was almost the same but the performance was hugely improved. The former version required some strict training before a safe flight but its successor is much easier to fly, and any person with zero experience of flying drones can securely operate it. In AR.Drone 2.0 a 3-axis Magnetometer was added to the already existing Inertial Measurement Unit of AR.Drone 1.0. Also for measuring the altitude in any height, a pressure sensor was added. But the main difference was the quality of the two cameras which were both improved in AR.Drone 2.0 [18].

Additionally, the Power edition, which was the drone used in this thesis, provided two high-density lithium polymer batteries (1.11 Volt 1500 mAh) in the package, increasing the flight time from 12 minutes (In Ar.Drone 2.0 with a 1000 mAh battery) to 36 minutes. It also comes with two hauls for protection of the drone. One is used when the drone is flown indoors and the hauls is needed so that the propellers and motors are not damaged in case of crashes. The outdoor haul only covers the central part of the drone, protecting the circuits and cameras.

The official way for controlling this drone is through the AR.Freeflight app on any smart phone or tablet. AR.Drone creates its own Wi-Fi network

and the control device needs to connect to it. Through out the years that the AR.Drone has reached the market, apart from controlling it through the official app, many developers have developed other applications to control the drone using the open-source software development kit (SDK) provided by Parrot [4].

The pilot can take videos and pictures while flying but it might exhaust the battery faster therefore a USB port is provided for storing the video or images and to be collected after landing.

Other than the video stream that the drone delivers, either from the front camera, the ground camera or both in a picture in picture video, a stream of data is also sent frequently with time lapses of around 5 milliseconds. This data stream which will be mentioned as "NavData", is used to get an understanding of the drone's state [7].

In table 2.1, the specification of the components and tools used in this drone are reported.

Technical aspects		
Horizontal Camera	720p 30fps HD	
	Wide-angle lens:	92° diagonal
	Basic encoding profile:	H264
	Photo format:	JPEG
	Connection:	Wi-Fi
Vertical Camera	QVGA 60 FPS	
	Wide-angle lens:	64° diagonal
Processor	ARM Cortex A8 1 GHz 32-bit processor with DSP video 800 MHz TMS320DMC64x	
OS	Linux 2.6.32	
RAM	DDR2 1 GB at 200 MHz	
USB	High-speed USB 2.0 for extensions	
Gyroscope	3 axles, accuracy of $2,000°$/second	
Accelerometer	3 axles, accuracy of +/- 50 mg	
Magnetometer	3 axles, accuracy of 6°	
Pressure sensor	Accuracy of +/- 10 Pa	
Motors	4 "inrunner" type brush-free motors:	14.5 watts and 28,500 rev/min
	Micro ball bearing:	Yes
	Nylatron Gears:	Yes
	Bronze self-lubricating ball bearings:	Yes

Table 2.1: Technical aspects

2.1 Control algorithms implemented in the drone

Quad-rotors such as AR.Drone are manufactured mostly as an entertaining product. If they were not safe and easy to fly, the number of customers would not be so high. Commercial drones need to be stable even when contingencies happen. Contingencies such as loss of communication or crashing to objects can easily be disastrous if there is not an internal control algorithm. A simplified explanation of the safety operational plan is expressed in figure 2.1.

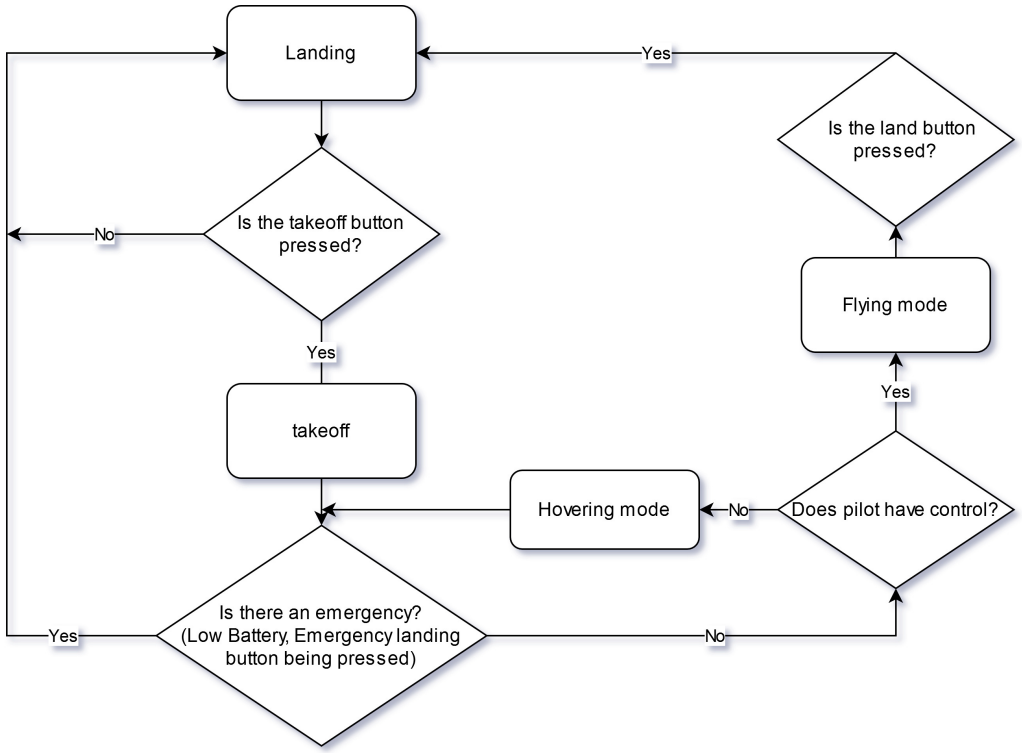

Figure 2.1: Simplified Flowchart of safety control algorithm in the AR.Drone 2.0

As shown in figure 2.1, when the drone is disconnected from the pilot it goes to hovering mode; it stops at the altitude that it already is and hovers in the air. This action requires accurate velocity estimation, so that the drone can make sure its velocity in all three axis is almost zero.

Additionally, there are other aspects of safety such as altitude control. In this drone, the pilot can preset a maximum and minimum altitude limit so

that, while flying, even if the pilot wants to fly the drone outside those limits, the drone would stay in the preset altitude range. For this kind of action, the drone needs to observe its exact altitude at all times. Therefore, multiple sensors have been implemented on the drone and their data are fused in a complex control and navigation algorithm to ensure the sufficient accuracy in attitude and velocity estimation.

2.1.1 Quad-copter flight basics

Quad-copters are in the rotary wings category of unmanned aerial vehicles. They have four arms in the shape of an X with a rotor attached to the tip of each arm. In the center of the cross the control and sensory units are placed [29].

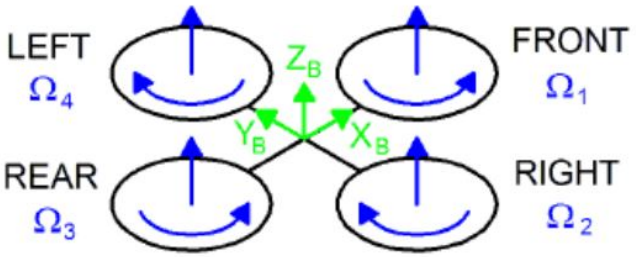

Figure 2.2: Simple quad-copter design

As seen in figure 2.2, in quad-copters the opposite rotors spin in opposite directions to produce the opposite direction forces to keep the drone afloat. For maneuvering the quad-copter all the work lies in the speed of each rotor. If all the rotors are spinning in the same speed, they create the same amount of force. Therefore the quad-rotor either hovers in the place or flies up or down depending on the forces created and the weight of the drone. However, it does that by keeping the rotors at the same relative altitude to each other or in other words in near-hovering position. In figure 2.3 each movement and its respective rotor speed changes are provided.

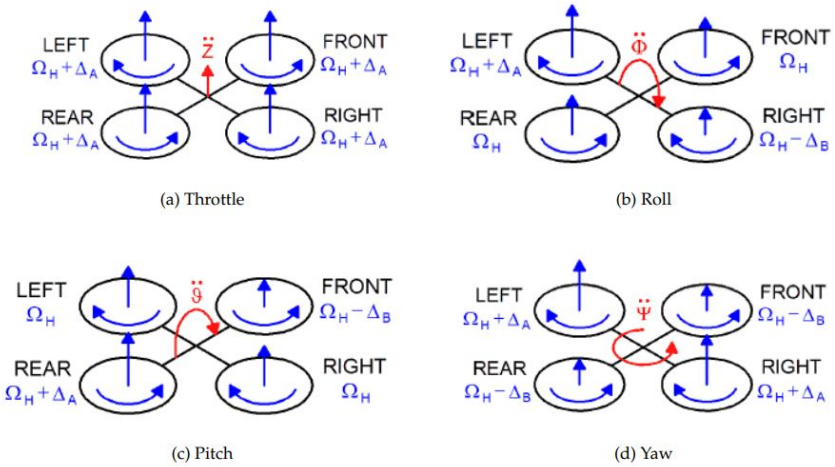

Figure 2.3: Quad-copter movement principles

In flying objects or air-crafts in general three angles are most used in literature; roll, pitch and yaw are what makes describing the pose of the object in respect to the inertial frame possible [4].

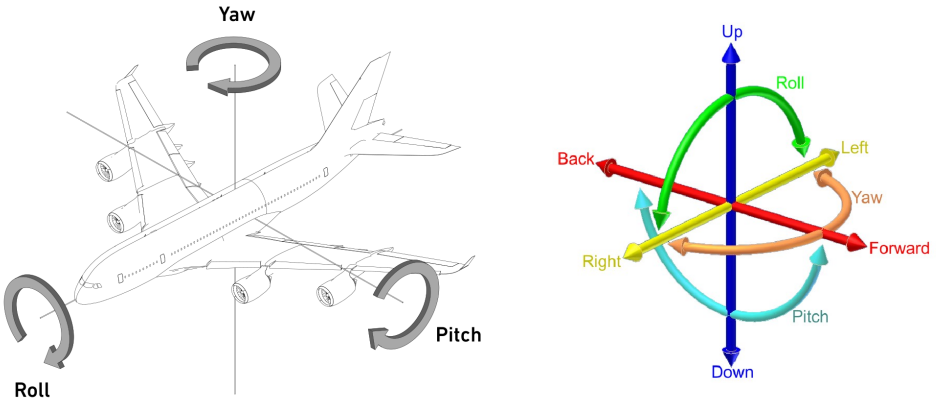

Figure 2.4: Roll, Yaw and Pitch

2.1.2 AR.Drone flight basics

For the AR.Drone the indicated body frame axis is different from 2.2 and it is indicated in figure 2.5. Therefore if this drone wanted to go forward as it is meant for it, the speed of the two back rotors would need to be more than

the pair in the front which would cause the drone to have a non-zero pitch angle. Respectively for a velocity along the y_b axis the drone's two rotors on the right or left would have to have less speed that the ones on the other side of the x_b axis.

The movement commands for the AR.Drone are encoded under a specific protocol. In this protocol, the command signals are arranged as elements of the control signal vector:

$$u = [u_{\dot{z}}\ u_{\dot{\psi}}\ u_{\phi}\ u_{\theta}]$$

Where:

$u_{\dot{z}}$ represents a linear velocity command, which causes displacements along the z_w axis or in other words takes the drone up or down.

$u_{\dot{\psi}}$ represents an angular velocity command, which causes rotations around the z_w axis and gives the drone an angle of yaw.

u_{ϕ} represents an inclination command related to the x_w axis, which indirectly represents a command of linear velocity related to the y_b axis. For this reason it also can be called u_{v_y};

u_{θ} represents an inclination command related to the y_w axis, which indirectly causes linear velocity related to the x_b axis. For this reason it also can be called u_{v_x}.

Each of these variables can take a value between -1 to 1 [31] [23] [15] [11].

Figure 2.5: Coordination frames with respect to the AR.Drone

This is also the reason that for flight commands, the drone does not take a metric unit of velocity like m/s, but a fraction of its maximum velocity. Therefore a typical command would be like: $Forward(m)$. In this command m is a number between 0 to 1. But it also can be a number from -1 to 0 which in this case the drone would not go forward but backwards with the fraction of speed indicated. This is also true for the turning action of the drone around its axis z_b. If the pilot wants the drone to turn clockwise around itself, they would have to send a command such as $Clockwise(0.4)$. In this case the drone turns in clockwise direction with 0.4 of its maximum angular speed. If instead of 0.4, the pilot uses -0.4 this would be interpreted as a command such as $Anticlockwise(0.4)$

2.1.3 Inertial Measurement Unit

Inertial Measurement Units, also commonly known as IMU, have been used for years now for attitude estimation in moving objects. They usually consist of an accelerometer and a gyroscope for sensing the angular velocity of the moving object. In this drone there is also a 3-axis magnetometer implemented. The data from these sensors are needed for attitude (pose) estimation.

Using low-cost inertial sensors implies dealing with bias and misalignment angles, and scale factors are not negligible and differ from one AR.Drone

sensor board to another.

Regarding a 3-axis accelerometer if we note Y_m its measured value and Y_v the true value,

$$Y_m = \alpha R Y_v + \beta$$

Where R represents a first order misalignment matrix between the frame of the camera to the frame of the sensor board, α stands for scale factors and β for accelerometer bias.

$$R = \begin{bmatrix} 1 & \psi & -\theta \\ -\psi & 1 & \phi \\ \theta & -\phi & 1 \end{bmatrix}$$

$$\beta = [\beta_1, \beta_2, \beta_3]^T$$

$$\alpha = diag(\alpha_1, \alpha_2, \alpha_3)$$

However, accelerometers do not measure the acceleration directly. They measure external specific forces which need to be processed to provide the 3-axis metric acceleration. As indicated in figure 2.6 the data of the gyroscope and accelerometer is processed in order to estimate the orientation and position of a moving object.

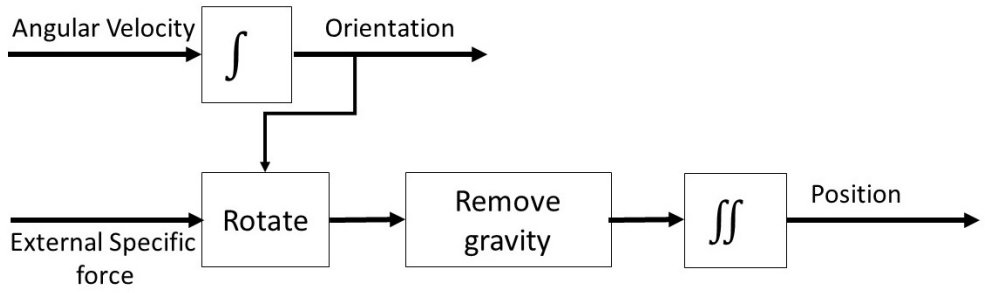

Figure 2.6: IMU data for attitude estimation

However, the attitude estimated only by inertial measurement units are not very accurate due to the noisiness of the sensor data. The bias of the AR.Drone IMU is not dealt with by its control algorithm. Therefore, the

data is fused with information from the camera or the GPS for more accuracy and robustness. The velocity estimates from drone vision is more discussed in the following section.

2.1.4 AR.Drone 2.0 Vision

The vertical camera under the drone uses two complementary algorithms to estimate the horizontal velocity of the drone. They can be altered back and forth to each other in different circumstances. The first one uses The Lucas and Kanade method following by Horn-Schunck method to calculate the optical flow in the bottom-facing camera feed.

In digital images, the Lucas–Kanade method is a broadly used differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade. In this method it is assumed that the flow is essentially same for the neighbourhood of the pixel under consideration. Thus they solve the basic optical flow equations for all the pixels in that neighborhood.

By regarding the information from several nearby pixels, the Lucas–Kanade method is less sensitive to image noise than point-wise methods. On the other hand, since it is a purely local method, it cannot provide flow information in the interior of uniform regions of the image.

The Lucas–Kanade method uses the assumption that the displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the pixel p under consideration.

Therefore, the optical flow equation can be assumed to be true for all pixels within a window centered at pixel p. where, the local image flow (velocity) vector (V_x, V_y). If we note the brightness of the image in point (x, y) at time t $E(x, y, t)$ and assume that the brightness of a point stays constant if the pattern moves therefore:

$$\frac{dE}{dt} = 0$$

Also by the chain rule of differentiation:

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t}$$

Now if we note $\frac{dy}{dt}$ as V_y and $\frac{dx}{dt}$ as V_x we will have:

$$E_x(q_n)V_x + E_y(q_n)V_y = -E_t(q_n)$$

where:

$E_x(q_n)$, $E_y(q_n)$ and $E_t(q_n)$ are derivatives of the image matrix in nth pixel in regards to x, y, and time respectively.

If the equations for all the pixels 1 to n are written, they can be written in matrix form $Av = b$.

$$\begin{bmatrix} E_x(q_1) & E_y(q_1) \\ E_x(q_2) & E_y(q_1) \\ \vdots & \vdots \\ E_x(q_n) & E_y(q_n) \end{bmatrix} \cdot \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -E_t(q_1) \\ -E_t(q_2) \\ \vdots \\ -E_t(q_n) \end{bmatrix}$$

Since this system is over-defined, the Lucas-Kanade method applies the least square principle and solves the 2×2 system and provides [17] [24]:

$$v = (A^T A)^{-1} A^T b$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} E_x^2(q_i) & \sum_{i=1}^{n} E_x(q_i)E_y(q_i) \\ \sum_{i=1}^{n} E_x(q_i)E_y(q_i) & \sum_{i=1}^{n} E_y^2(q_i) \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{n} E_x(q_i)E_t(q_i) \\ -\sum_{i=1}^{n} E_y(q_i)E_t(q_i) \end{bmatrix}$$

The second algorithm is called "corner tracking" and estimates the displacement of several points of interest (trackers). The motion of the camera is deduced from these computed displacements through an iteratively weighted least-squares minimization procedure. The algorithm uses a first processing of data with a FAST type corner detector of Trajkovic and Hedley [33]. After a second analysis a fixed number of trackers are placed over the corner positions. Additionally, some erroneous tracker points are eliminated. Afterwards within a frame depending on the expected tracker's screen speed, their update positions are searched in the image [26].

Since the Lucas-Kanade equation coefficients provide a cheap but useful estimation of the overall picture contrast, it is a good method for when the

contrast in the picture is low. Additionally, due to its constant computational cost, it is used as the default vision algorithm.

The drone can switch to the second algorithm when the speed and scene are suitable. The scene would have to be fine for the corner detector and the image speed would need to be low. This is the reason that in higher speeds only the first algorithm is applied. Also when number of trackers is lower than needed, the vision algorithm goes back to default.

The drone stability and estimated velocity accuracy is dependant on this feature. Since the attitude estimated only based on the IMU sensor data is very noisy and erroneous, the drone depends on its vision to have a better estimate of its velocity. Otherwise when it is needed to be hovering in place, which is done by estimating its horizontal velocity and compensating for any significant changes in the position, it would have a much bigger drifting range. The velocity estimate based on the drone's vision is more accurate when the ground is sufficiently textured and the environment is well-lit.

This incident was also observed in the tests done for this thesis that the drone's estimated velocity lies heavily on its vision. So far that on a floor such as figure 2.7 the drone would send the message that it has no vision in its flying state data array and it would drift away instead of actually hovering in a relatively constant position. Therefore for tests on that location some colorful markers were added to the floor to increase the texture seen by the drone camera.

Figure 2.7: A sample of non-preferable floor for AR.Drone

2.1.5 Attitude Estimation

At first, when the battery is attached, the IMU sensors are calibrated based on the stationary data. This is done automatically, unless the Magnetometer needs calibration. In that case, a notification is received by the pilot to calibrate it. In the next step the sensor data enter a complimentary (Kalman) filter which estimates the attitude and de-biases the gyroscope.

The estimation done by the data from vision is also fused and used to de-bias the accelerometer, which then will feed to the primary filter to update the estimated attitude. This is a rather simplified explanation of the control sequence done in the drone. The complete details of control principle of an AR. Drone is shown in the block diagram in figure 2.8 [26].

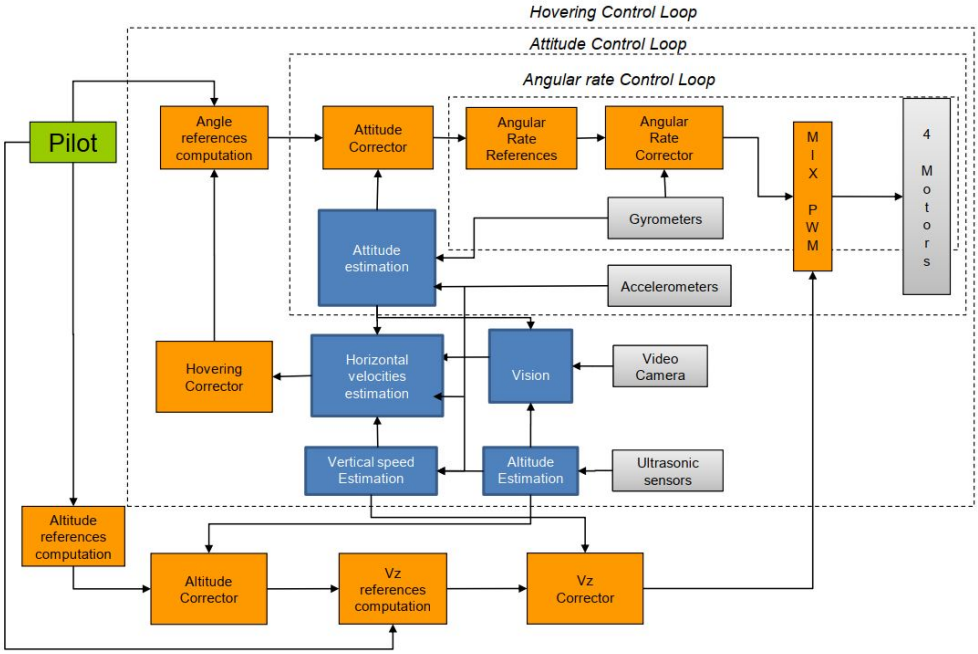

Figure 2.8: The control principle and sensor data fusion in AR.Drone

This procedure gives a fairly accurate estimation of horizontal velocity, Euler angles and the altitude. These information are updated and sent over by the drone to the pilot in a data stream called "NavData". In the next section this data stream will be analyzed more closely.

2.2 NavData Analysis

The data stream that the AR.Drone sends over to the connected device with a frequency of 200 HZ, is called "NavData"; because it includes the navigation information among other things. If the pilot chooses, they can have the summarized format or the "demo" with less frequency. However, by changing the configuration it is possible to receive all the data.

Understanding the demo is quite simple, the complete format of the data however is another matter. Unfortunately there is not any complete source that would explain each and every part of this stream. Although in the official developer guide provided by parrot [4] and also files from people who

have studied and worked with the drone before [19] [13] , there are some comments and notes on what they have figured out about NavData. In the following tables, the available comments have been gathered from and about each part of NavData.

Demo	
controlState	control state (CTRL_TRANS_LOOPING, CTRL_TRANS_LANDING, etc)
flyState	Flying state (landed, flying, hovering, etc.)
batteryPercentage	State of charge of the battery expressed in percentage.
rotation	A vector of roll, pitch and yaw
frontBackDegrees	Theta or Pitch in millidegrees
leftRightDegrees	Phi or Roll in millidegrees
clockwiseDegrees	Psi or Yaw in millidegrees
altitude/altitudeMeters	Altitude in meters
velocity	A vector of VX, VY and VZ
xVelocity	Estimated velocity in X axis in mm/s
yVelocity	Estimated velocity in Y axis in mm/s
zVelocity	Estimated velocity in Z axis in mm/s
frameIndex	Streamed frame index
detection	Type of the tag searched in detection

Table 2.2: Demo

Drone state		
	0	1
flying	Drone is landed	Drone is flying
Video Enabled	Video is disabled	Video is enabled
Vision Enabled	Vision is disabled	Vision is enabled
Control Algorithm	Euler angles control	Angular speed control
Altitude Control Algorithm	Altitude control is disabled	Altitude control is enabled
Sart Button State /User feedback	Start button is not pressed	Start button is pressed
Control CommandAck	None	Received
Camera Ready	Camera is not ready.	Camera is ready.
Travelling Enabled	Traveling is disabled.	Traveling is enabled.
USB Ready	USB is not ready to use.	USB is ready to use.
Navdata Demo	All of NavData is provided.	Only NavData demo is provided.
Navdata Bootstrap	Options are sent.	There are no options of NavData.
Motor Problem	Motor is alright.	Motor has a problem.
Communication Lost	Communication is alright.	Communication has a problem.
Software Fault	Software is alright.	Software is faulty.
Low Battery	Battery SoC is alright.	Battery SoC is low.
User Emergency Landing	User emergency landing is disabled.	User emergency landing is enabled.
Timer Elapsed	Timer has not elpsed.	Timer has elapsed.
Magnometer Needs Calibration	No calibration is needed.	Calibration is needed.
Angles Out Of Range	Angles are in the range.	Angles are out of the range.
Too Much Wind	Wind speed is alright.	It is too windy.
Ultrasonic Sensor Deaf	Ultrasonic sensor is alright.	Ultrasonic sensor is deaf.
cutout Detected	Cutout system is not detected.	Cutout system is detected.
PIC Version Number Ok	Version number is faulty.	Version number is alright.
ATCodec ThreadOn	ATCodec thread is disabled.	ATCodec thread is enabled.
NavData ThreadOn	NavData thread is disabled.	NavData thread is enabled.
Video ThreadOn	Video thread is disabled.	Video thread is enabled.
Acquisition ThreadOn	Acquisition thread is disabled.	Acquisition thread is enabled.
Control Watchdog Delay	Watchdog is well scheduled.	Watchdog has a delay more than 5 ms.
ADC Watchdog Delay	Uart2 is alright.	It has a delay more than 5 ms.
COM Watchdog Problem	Communication watchdog is alright.	Communication watchdog is faulty.
Emergency Landing	There is no emergency landing at the moment.	The drone is in emergency landing.

Table 2.3: Drone state

rawMeasures	
accelerometers	filtered accelerometer [x,y,z]
gyroscopes/gyrometers	filtered gyrometers [x,y,z]
gyroscopes110/gyrometers110	gyrometers [x,y] 110 deg/s
Battery MilliVolt	Battery voltage in milliVolts
us	
usDebutEcho	
usFinEcho	
usAssociationEcho	
usDistanceEcho	
usCourbeTemps	
usCourbeValeur	
usCourbeRef	
echo	
flagEchoIni	
nbEcho	
sumEcho	
altTemp/altTempRaw	Altitude in millimeters

Table 2.4: RawMeasures

physMeasures	
temperature of accelerometer	temperature in Celsius
temperature of gyroscope	
accelerometers	acceleration in mg
gyroscopes	angular velocity in deg/s
alim3V3	3.3volt alim
vrefEpson	ref volt Epson gyro
vrefIDG	ref volt IDG gyro

Table 2.5: PhysMeasures

References	
theta	Theta reference embedded in mdeg
thetaI	Theta reference int in mdeg
phi	Phi reference embedded in mdeg
phiI	Phi reference int in mdeg
psi	Psi reference embedded in mdeg
pitch	Pitch reference embedded in mdeg
roll	Roll reference embedded in mdeg
yaw	Yaw reference embedded in mdeg/s
vx	X-velocity reference in mm/s
vy	Y-velocity reference in mm/s
thetaMod	Theta modele in radian
phiMod	Phi modele in radian
kVX	
kVY	
kMode	
ui	

Table 2.6: Refrences

PWM	
motors	PWM
satMotors	PWM
gazFeedForward	PWM
gazAltitude	mm/s
altitudeIntegral	mm/s
vzRef	PWM
uPitch	PWM
uRoll	PWM
uYaw	PWM
yawUI	PWM
uPitchPlanif	PWM
uRollPlanif	PWM
uYawPlanif	PWM
uGazPlanif	PWM
motorCurrents	mA
altitudeProp	PWM
altitudeDer	PWM

Table 2.7: PWM

Altitude	
vision	Altitude by vision in mm
velocity	Z-velocity in mm/s
ref	Altitude reference in mm
raw	Raw sensor data in mm
observer	(acceleration(m/sˆ2), altitude(m), x, state)
estimated	(vb, state)

Table 2.8: Altitude

Vision	
state	
misc	
phi(trim,refProp)	rad
theta(trim,refProp)	rad
newRawPicture	
capture(theta, ohi, psi, altitude,time)	time in TSECDEC format
bodyV(x,y,z)	mm/s
delta(phi,theta,psi)	
gold(defined,reset,x,y)	

Table 2.9: Vision

VisionDetect	
nbDetected	number of found tags
type	type of detected tags
xc	X position of the found tag inside the picture (0-1000) 0=left
yc	Y position of the found tag inside the picture (0-1000) 0=top
width	optical width of the tag
height	optical height of the tag
dist	distance of the tag (0-1000) or cm needs to be checked
orientationAngle	orientation of the tag for oriented tags in degrees
rotation	
translation	
cameraSource	The camera that the tag is detected on

Table 2.10: VisionDetect

VideoStream	
quant	quantizer reference used to encode frame [1:31]
frame(size,number)	frame size (bytes) and frame index
atcmd sequence	atmcd ref sequence number
atcmd meangap	mean time between two consecutive atcmd_ref (ms)
atcmd vargap	(SU)
atcmd quality	estimator of atcmd link quality
bitrate out	measured out throughput from the video tcp socket
bitrate desired	last frame size generated by the video encoder
data	misc temporary data
tcpQueueLevel	queue usage
fifoQueueLevel	queue usage

Table 2.11: VideoStream

Magneto	
mx	
my	
mz	
raw(x,y,z)	magneto in the body frame(mG)
rectified(x,y,z)	
offset(x,y,z)	
heading	heading unwrapped heading gyro unwrapped heading fusion unwrapped
ok	
state	SU
radius	mG
error(mean, variance)	

Table 2.12: Magneto

Wind Speed	
speed	m/s
angle	estimated wind direction in North-East frame [deg] e.g. if wind_angle is pi/4, wind is from South-West to North-East
compensation(theta,phi)	rad
stateX	SU
debug	

Table 2.13: Wind speed

KalmanPressure		
offsetPressure	[Pa]	
estimated	altitude	mm
	velocity	m/s
	angle(pwm,pressure)	(m/sˆ2,Pa)
	us(offset, prediction)	(m,mm)
	covariance(alt,pwm,velocity)	(m,pwm,m/s)
	groundeffect	(SU)
	sum	mm
	reject	SU
	umultisinus	
	gazaltitude	
	flag multisinus	
	flag multisinus start	

Table 2.14: KalmanPressure

HDvideoStream	
hd video state	
storageFifo	
usbkey size	USB key in kbytes - 0 if no key present
usbkey freespace	USB key free space in kbytes - 0 if no key present
usbkey remaining time	time in seconds
frameNumber	'frame_number' PaVE field of the frame starting to be encoded for the HD stream

Table 2.15: HDvideoStream

Scattered data without categories	
sequenceNumber	incremented for each sent packet
time	time elapsed since the battery has been connected in milliseconds.
gyrosOffsets	(x,y,z) gyroscope offset in deg/s
trims	
angularRates	(r)
eulerAngles	(Theta,Phi) in mdeg
rcReferences	(pitch, roll, yaw, gaz, ag) rc embedded
visionRaw(tx,ty,tz)	
visionOf(dx,dy)	
visionPerf	
trackersSend(locked,point)	
watchdog	Watchdog control
adcDataFrame (version, dataframe)	
games	(double tap counter, finish line counter)
pressureRaw(up,ut,tempreture[0_1C],pressure(Pa))	

Table 2.16: Scattered data without categories

2.3 AR.Drone's Tag Detection Feature

The AR.Drone 2.0 also has a tag detection feature, that is, by image processing and using the two cameras, the drone is able to detect the specific tags shown in figure 2.9. This capability makes multi-player games possible in which tags are stuck to the sides of drone hauls so the friends and enemy drones will be detected by each other.

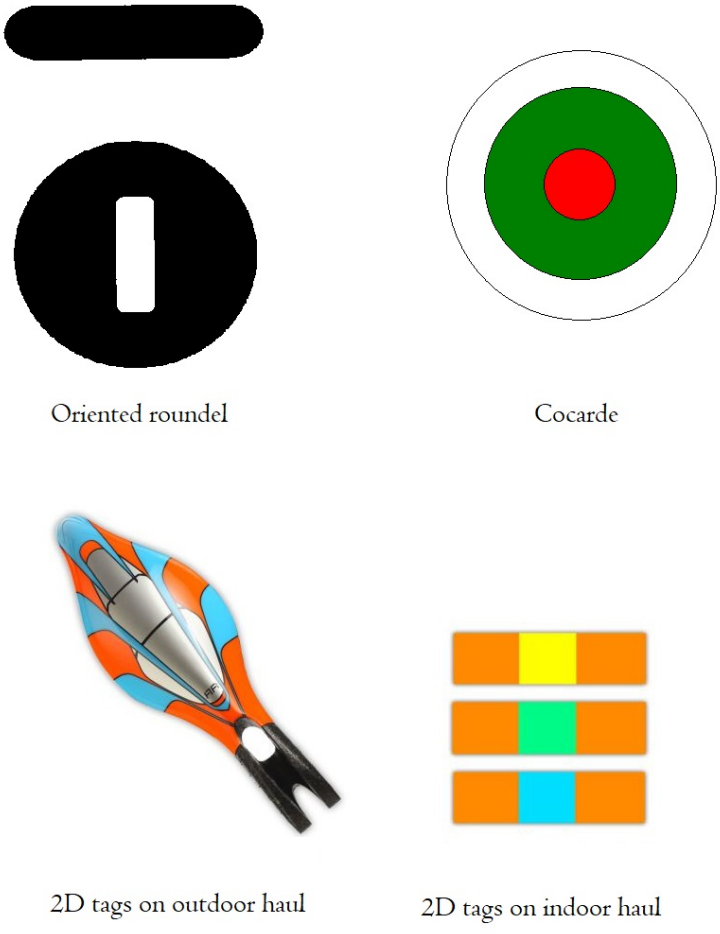

Figure 2.9: Detectable tags by AR.Drone 2.0

However, games are not their only application. In this thesis, this particular capability is used for planning and preparing the path of an autonomous drone. When using the "Oriented roundel", the drone detects the tag in every picture taken by the cameras and if seen, sends the responding notification in the NavData. When the drone is above the tag the vector nbDetectd of the NavData will be [1,0,0,0] and with the orientation angle vector we will be able to understand the position of the tag in respect to the drone. In figure 2.10, the front camera is marked as a red dot and the orientation angles are shown respectively.

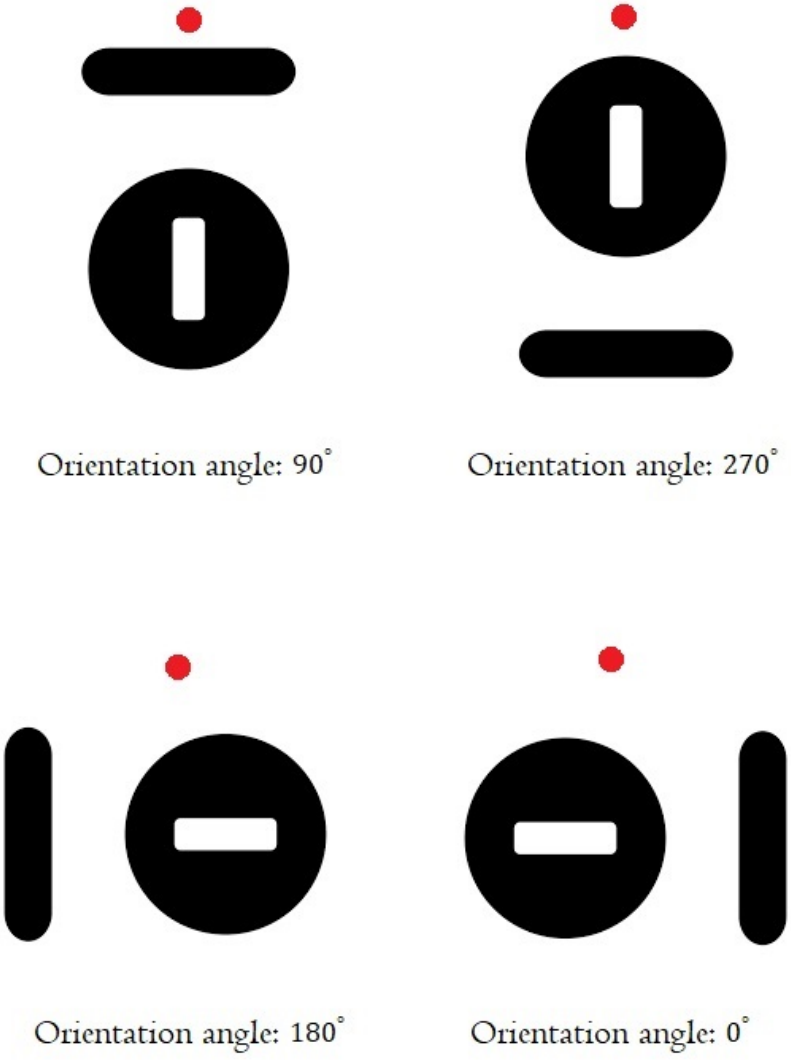

Figure 2.10: The Orientation Angle with respect to the drone

Chapter 3

Methodology For Defining the Metrological Characteristics

The goal of this project was to make a low-cost autonomous drone that would follow a pre-made path in GPS deprived environments. For this purpose the technologies and tools embedded in the drone are used. They include but are not limited to horizontal velocity estimation and tag detection.

First part of research was focused on the metrological characteristics of the measurement units in the drone. For instance to find the error of the estimated horizontal velocities developed in the drone. Also to distinguish the area of sight the drone has in different altitudes. As expected, the area on the floor that the drone can see, increases in size as the drone goes higher. Additionally, with the area of sight, its ability to detect tags located on the floor also increases. One section of this part of research was to determine how far the drone can be from the tag in the $x - y$ (horizontal) plane and still be able to detect it correctly in a specific altitude. Take 2 meters for instance, how much to the left or right and forward or backward of the tag, the drone can be, to still detect it completely. These algorithms and their test results will be discussed in respectively current and next chapter.

Second part of the research was focused on autonomous flight of the drone. One section was to develop an algorithm for the drone so that it would go straight in X-direction for n meters (pre-defined) and then stop and land. It

would need to fix its own deviation in Y-direction during and at the end of the flight path. Also in the final stage, the objective was to prepare a path in a room using the oriented roundel tags on the floor and have the drone autonomously follow the path to the the desired destination. The algorithms and their test results are respectively provided in chapters 5 and 6.

3.1 Interfacing With the Drone

For this work the Ar-drone module of Node.js by Felix Geisendörfer [13] is used. This module allows the user to program the drone in high-level commands and receive the NavData along the way. In this work a code is written beforehand mostly by the choice in the software notepad++ and then operated in Node.js command prompt window. During the flight the program is not changed unless in case of emergencies, when the running code is stopped and emergency landing code is applied. When during the flight the running code is terminated the drone is lost without any commands therefore it initiates its own Hovering control procedure until a new command is delivered.

If in the code we include specific lines we can observe the data we want in the command prompt window with the speed they are received from the drone. This can also be useful to know which loops or commands is being run by printing specific distinguished comment messages.

3.2 Estimated Altitude

Due to the pressure sensors and the sonic-altimeter that is included in the drone, the altitude estimated by it is quite accurate. An algorithm was developed for it regardless. In the simple algorithm seen in figure 3.1, the drone would take off and after stabilizing itself, it would allow the NavData stream to be received by the station. In the next step the drone's altitude would be regarded from the demo pack of NavData. If it was less than H_{target}, the designated height that we want the drone to be in, it would fly upwards with the command speed of 0.1; which as discussed in chapter 2 can be interpreted as 10% of the maximum speed.

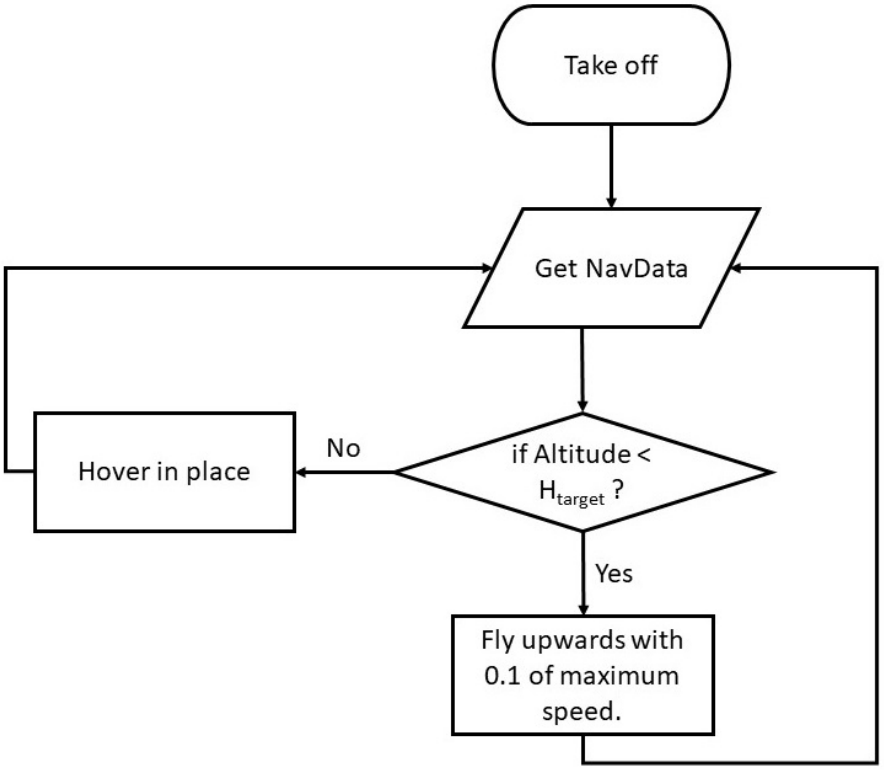

Figure 3.1: Altitude control algorithm

When the estimated altitude reaches the desired value, the hovering command is sent to it repeatedly until the pilot terminates the code all together and runs the landing command. This cycle would run with each new package of NavData that arrives with frequency of 200 HZ.

While the drone is hovering it is possible to measure its real altitude to compare with the estimated value.

3.3 Tag Detection area of sight

For the final step of this work it was required to know how far can the drone get from the tag in a constant altitude $\hat{H}$ to still be able to detect it.

Since this distance was difficult to measure while flying, the algorithm was

written to be run while the drone has not taken off. In this code the drone would still continuously send the NavData package to the station where only the tag detection section was reviewed and recorded.

In this process an oriented roundel tag would be stuck on the floor and the drone would be held manually directly upon it in the approximate altitude of $\hat{H}$ for instance 1 meter. The program would be already running so the drone would confirm that it detects the drone at that position. In the next step the drone would be moved slowly to right and the tag detection data would be checked simultaneously. The instant that the drone cannot detect the tag anymore it would be lowered to the floor and the distance from the tip of the front camera in the starting point to the tip of the front camera in the end point would be measured and recorded.

If repeated for the four direction of left, right, forwards and backwards for different levels of $\hat{H}$, a sense of the drone's area of sight would be achieved.

3.4 Estimated Horizontal Velocity

As mentioned in chapter 2, the estimated horizontal velocity is broadcast by the drone for X and Y directions in the NavData package along with other information including the time stamp of the drone starting from the boot up (The moment the battery was connected).

In this algorithm, the velocities are taken in and integrated discretely over time to estimate the horizontal displacements. In the figure 3.2, this process is described.

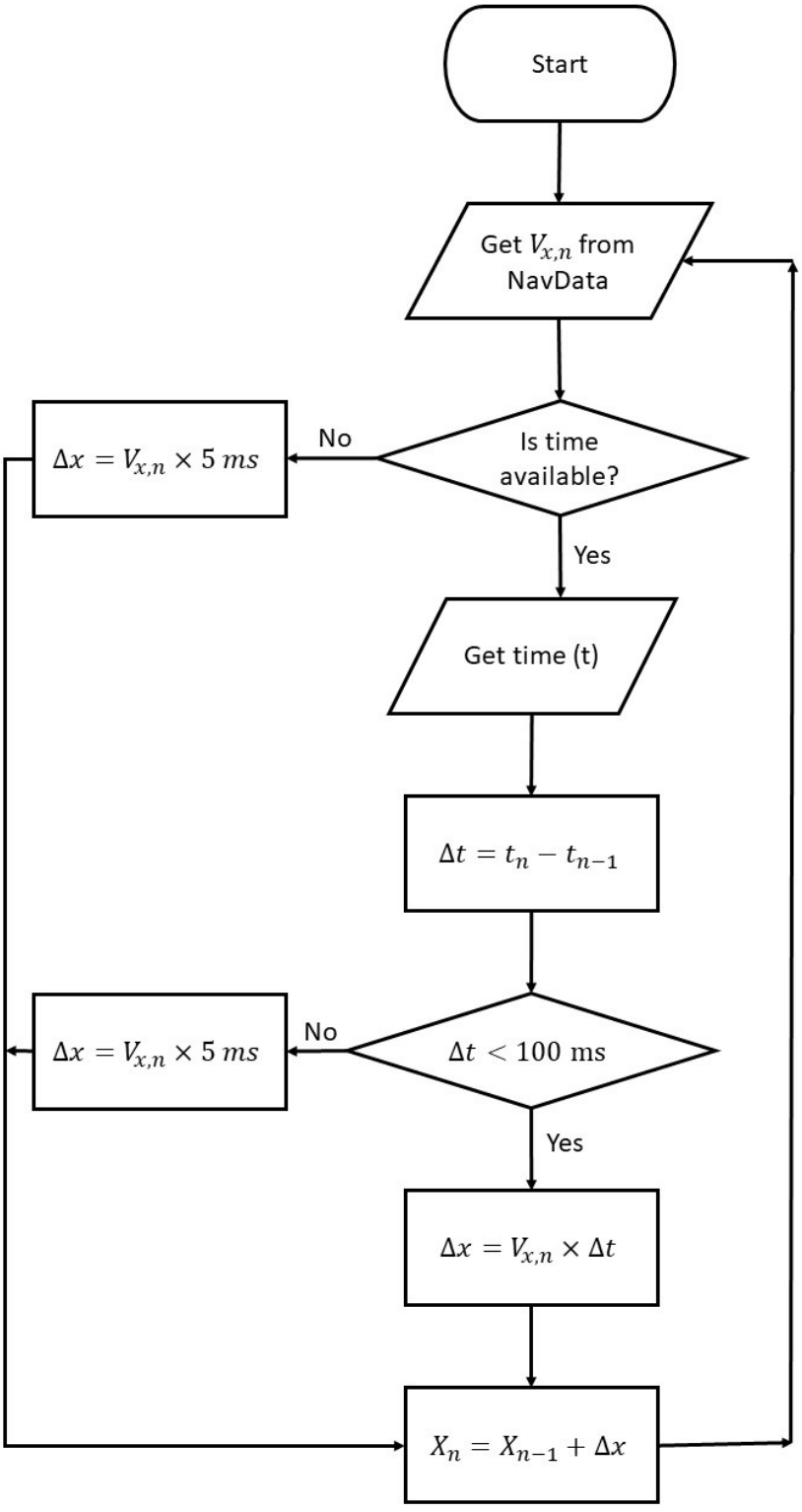

Figure 3.2: X-Displacement computation algorithm

Where:

$V_{x,n}$ is the horizontal velocity in direction of the x-axis. n is the indicator of the step of the computation. As seen in each cycle that the NavData is received, only the newest velocity is used in the calculations.

Δt is the time-step of this cycle of calculation and it is gained by subtracting the current time provided by the drone and the timestamp from the previous NavData package that was processed. This is the reason that during the flight and particularly the beginning of the integration, it is possible for the program or the drone to miss some packages of NavData or only the timestamp particularly. Therefore some precautions are taken to prevent huge errors in the calculations due to this. In case in a step only the velocity is available or the time-step Δt is more than $100ms$, the average time-step which is $5ms$, is used.

Δx is the X-displacement done in this step of calculation process. It is calculated by multiplying the current velocity and the related time-step. In transporting the algorithm to the program it is important to take the units of each variable into consideration. The NavData time package received by the AR-Drone Module used in this research is in milliseconds and the velocities have the unit of millimeters per second. Therefore the time-steps are converted into seconds so that the resulting displacements are in millimeters.

X_n is the current displacement in x-direction which is set to 0 in the beginning of each program. In each cycle of computation, X-displacement from the step before (X_{n-1}) is added to Δx to calculate the current displacement and therefore complete the process of discretely integrating the horizontal X-velocity. The same algorithm is used for the Y-direction displacement.

To find the accuracy of estimated displacements, multiple tests would be run. In them, the drone would be put on a specific point to mark the starting position to . After placing it, a mission would be uploaded to the drone. This mission would include taking off, flying forward for 2 seconds for instance and then landing. In the meantime, the drone would take the velocities and compute the horizontal displacements and report them to the station computer.

After landing, the displacements computed for X and Y directions were recorded. Furthermore, to compute the error, the position of the drone's

landing was marked and its distance from the starting point was also measured in X and Y directions. In the same way that was done for recording the distance for the tag-detection tests, the point of the front camera would be considered as the reference point before taking off and after landing.

More than 130 test were done as mentioned above where the landing distance was from 1 to approximately 6 meters. They were done in different locations with different levels of command speed. In figure 3.3 , it is visible that the computed X-displacement and the measured X-displacement mostly have a polynomial relationship.

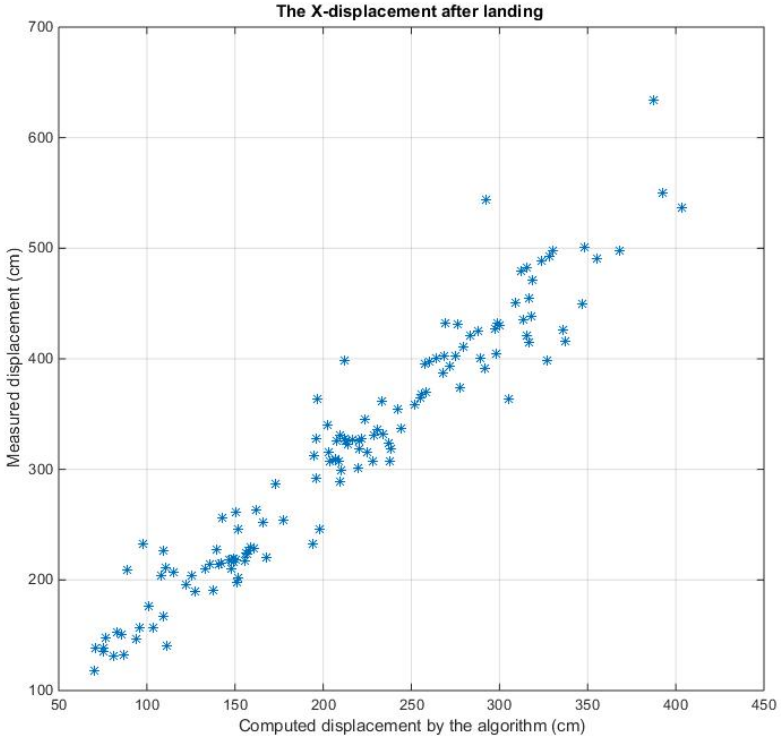

Figure 3.3: X-Displacement after landing: Comparison between measured and computed values

As such an error would be originated from the estimated velocity by the drone, different kind of tests were required to find it without an external actual velocity measurement device.

$$V_{est} = V_{real} + V_e$$

Where:

V_{est} is the velocity estimated by the drone.

V_{real} is the real horizontal velocity of the drone.

V_e is the velocity error.

$$X_{est} = \int V_{est}\ dt = \int V_{real}\ dt + \int V_e\ dt = X_{real} + X_e$$

$$X_e = \int V_e\ dt = X_{est} - X_{real}$$

$$V_e = dX_e/dt$$

Where:

X_{est} is the estimated displacement.

X_{real} is the real horizontal displacement of the drone.

X_e is the error displacement.

This would be possible to apply if the error displacement was available at all times during the flight to be discretely deviated over time. However, throughout the flying period, the real displacement (X_{real}) is not available for recording but only after landing. Therefore another procedure would need to be followed.

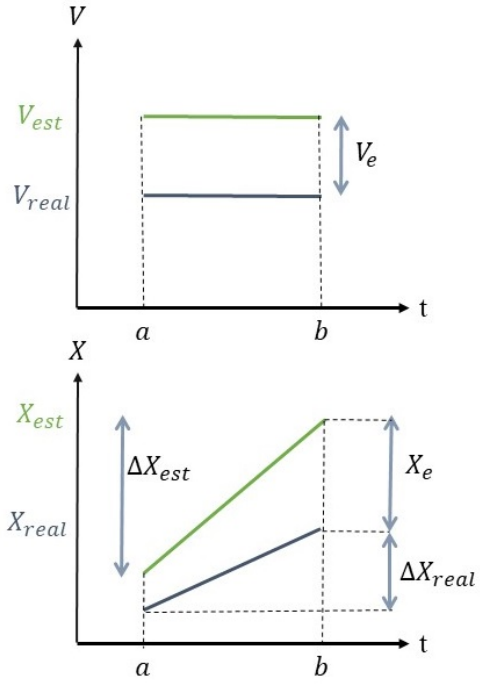

Figure 3.4: Constant V_{est}, V_{real} and their respective displacements

Assuming the situation in figure 3.4, during the time period from a to b the X velocity stays constant, then the displacement has a constant slope. And if the X-displacement is generally in the same trend, by the triangle law we can assume (see figure 3.4):

$$\Delta X_{real} = \frac{\Delta X_{est}}{X_{est}} \; X_{real}$$

$$V_{real} = \frac{\Delta X_{real}}{b - a}$$

$$V_e = V_{est} - V_{real}$$

Running tests that would create situations in which the estimated velocity stays constant yields the error with which the drone estimates the horizontal velocities.

Chapter 4

Test Results for Metrological Characteristics

4.1 Estimated Altitude Accuracy test results

Tests were run according to the algorithm mentioned in chapter 3 for different H_{target} [1]. The results are available in table 4.1 where H_{target} is the target altitude, Estimated Altitude is the altitude estimated by the drone sensors and control loops and Measured Altitude is the altitude measured by hand. However due to difficulty of measuring the altitude of a flying drone the real measurements are more susceptible to human error.

H_{target}	100	125	150	175	200
Estimated Altitude [cm]	105	126	156	181	204
Measured Altitude [cm]	110	131	165	184	210

Table 4.1: The results related to tests about the Estimated altitude accuracy

Seen both in table 4.1 and figure 4.1, the estimated altitude by the drone's altimeter sensors are quite accurate and reliable.

[1] `https://github.com/fereshte75/Autonomous-path-following-ar.drone/blob/master/altitude_control.js`

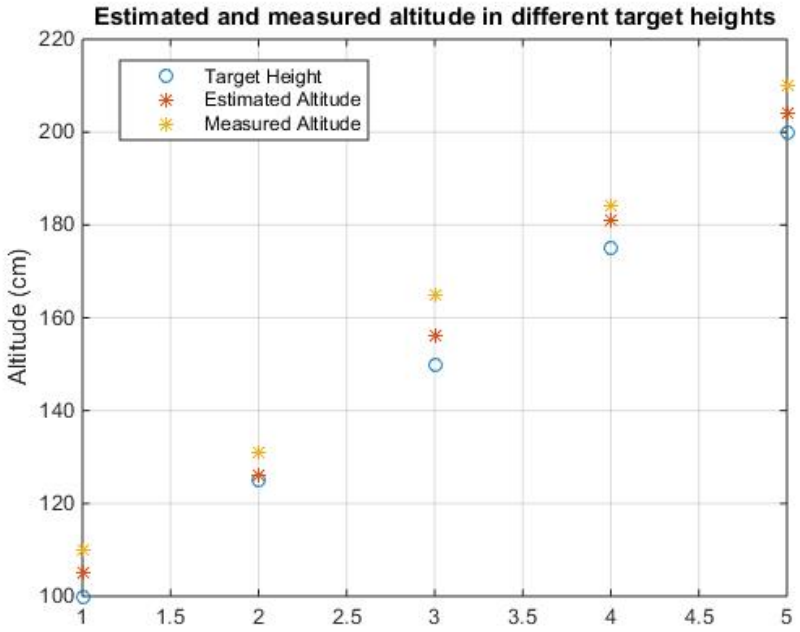

Figure 4.1: Estimated and Measured Altitude for different target heights

4.2 Tag Detection Area of Sight test results

Tests for this purpose and with the respective method were run for the same 5 altitudes of the previous section [2]. Additionally, another test was run for finding the minimum height in which the drone can detect the tag. The least altitude in which it is possible for the drone to detect the tag is around 30 centimeters, but it is not very reliable. On the other hand, at 50 centimeters of altitude directly upon the tag the drone is sure to detect the tag. For higher altitudes, the results are provided in table 4.2.

[2] `https://github.com/fereshte75/Autonomous-path-following-ar.drone/blob/master/flightless_data.js`

Altitude [cm]	75	100	125	150	175	200
Forward direction (X-)[cm]	-15	-20	-26	-33	-44	-51
Backward direction (X+)[cm]	31	42	48	57	77	100
Right direction (Y-)[cm]	-27	-32	-51	-63	-81	-103
Left direction (Y+)[cm]	28	33	48	63	90	102

Table 4.2: The results related to tests about Tag-Detection area of sight

For better visual understanding of the drone's detection limits reported in table 4.2, the data were plotted to give a 3-dimensional sense of the values (see Figures 4.2 - 4.5). In these figures, take figure 4.2 for instance, if we consider the tag to be approximately at point $(0, 0, 0)$ then the drone for instance at altitude of 100 centimeters with its center of mass almost aligned with the center of tag, it can move left or right about only 33 centimeters to still be able to detect the tag. Also it can move forward for 42 centimeters and backwards 20 centimeters for the same detection.

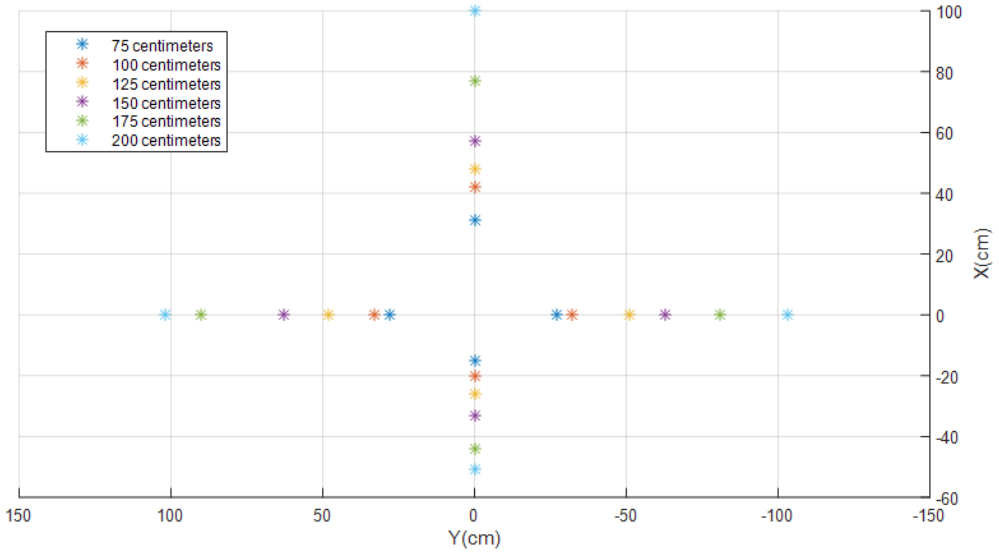

Figure 4.2: Tag detection limits (X-Y) plane

Figure 4.3 shows the $Y - Z$ plane and it is visible that the drone has approx-

imately equal area of sight in its right in respect to its left.

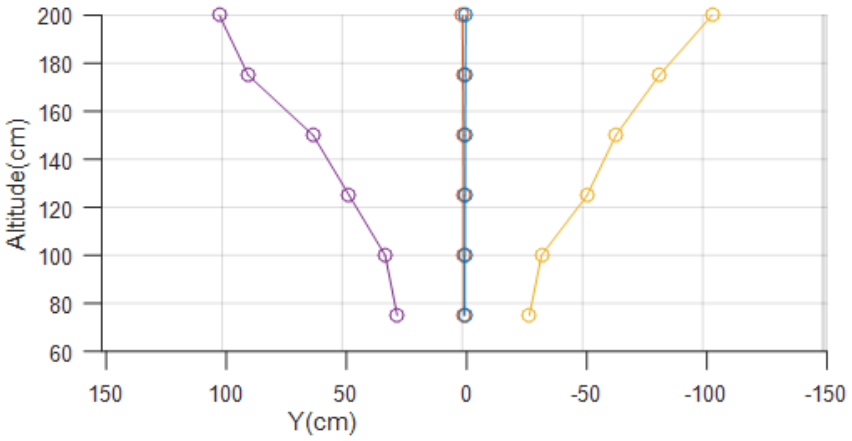

Figure 4.3: Tag detection limits (Y-Z) plane

Figure 4.4 shows the $X - Z$ plane and it is visible that the drone has a wider area of sight in its back than its front.

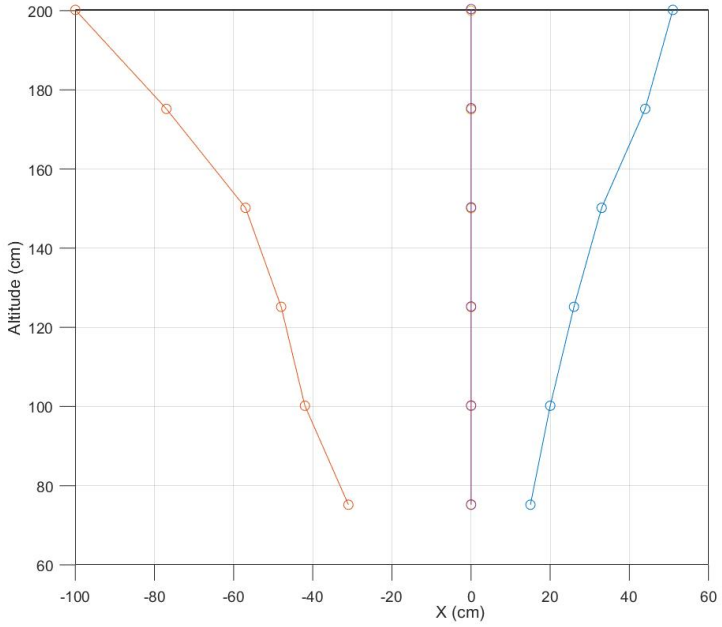

Figure 4.4: Tag detection limits (X-Z) plane

Figure 4.5 shows a 3 dimension view of the data from altitude 75 centimeters up to 2 meters.

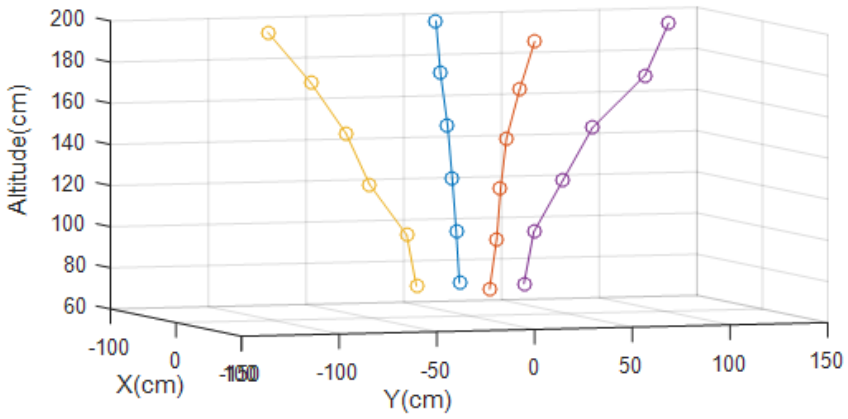

Figure 4.5: 3D plot of Tag-detection limits in different heights

As seen in both table and figures above the detection limit is similar for both sides of the y-axis as expected since the bottom facing camera is implemented in the center of y-axis and if the tag is in either left or right side of the drone, it has the relative same distance from the camera.

This however does not hold for the x-axis since the bottom facing camera is in the rear end of AR.Drone's body, the detection sight area is more limited when the tag is in front of the drone along the x-axis. Unsurprisingly it is reverse when the drone is in front of the tag.

4.3 Estimated Horizontal Velocity Error Test Results

In order to estimate the error for the velocity, a situation such as the one described in previous chapter was created by uploading a mission to the drone that include flying for some time with a relative constant velocity and then landing [3]. In these tests, the velocity and X-displacement would have a trend such as figure 4.6.

[3] `https://github.com/fereshte75/Autonomous-path-following-ar.drone/blob/master/velocity_data.js`

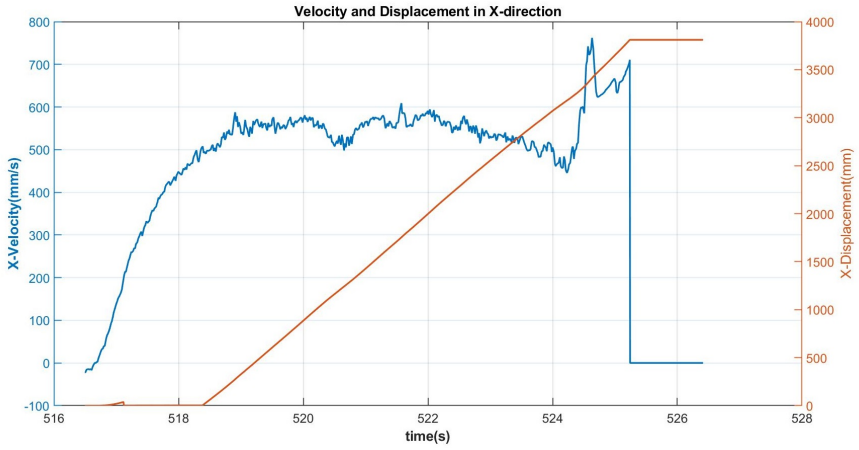

Figure 4.6: X-Velocity and Displacement over approximately 10 seconds

Additionally the drone would set its estimated displacement to zero ($X_{est} = 0$) upon seeing a tag. This was done so that the drone would not start integrating the velocity before stabilizing itself after the takeoff. Therefore a tag would be placed on the floor near the take off position of the drone so that it would be detected after the drone has taken off. This is also visible in figure 4.6 that the X-displacement does not start increasing above 0 before when the time is approximately 518.5 seconds. This matter was also considered in measuring X_{real} which was 464 cm in this case. The real displacements (X_{real}) were measured from the tag as the starting point to the landing point.

In this example $t_a = 518.55\ s$ and $t_b = 523.97\ s$.

$$\Delta X_{est} = X(b) - X(a) = 2976.8\ mm$$

$$\Delta \hat{X}_{real} = \frac{\Delta X_{est}}{X_{est}(end)} \times X_{real}(end) = \frac{2976.8}{3810.8} \times 4640 = 3624\ mm$$

$$V_e = mean(V_{est}) - mean(V_{real})$$

$$V_e = \frac{\Delta X_{est} - \Delta \hat{X}_{real}}{t_b - t_a} = -119.25\ mm/s$$

Where:

ΔX_{est} is the displacement during the time period of [a-b].

$X(b)$ and $X(a)$ are the displacements estimated by the drone at times t_b and t_a.

$\Delta\hat{X}_{real}$ is the real displacement during the time period of [a-b] that we calculate from the data at hand.

$X_{est}(end)$ is the total displacement estimated by the drone at the end of the path.

$X_{real}(end)$ is the total displacement measured manually after the drone's landing.

V_e is the error velocity.

$mean(V_{est})$ is the average velocity estimated by the drone during the time period of [a-b]. The average is taken because in real flights the velocity has little perturbations and cannot stay exactly constant.

$mean(V_{real})$ is the average real velocity during the time period of [a-b].

As seen in the example above for the [a,b] period that had a mean velocity of 547.8 mm/s, the error velocity was about $-119mm/s$.

This test was done for different levels of velocity for x direction and it was seen that V_e is not constant and changes with V_{est}. The results are available in table 4.3.

Mean Velocity Estimated by the drone ($mean(V_{est})$) [mm/s]	Error Velocity (V_e) [mm/s]
191.39	-69.1
260	-62.5
503.11	-123.1
547.8	-119.25
559.08	-114.1
632.38	-120.8
654.51	-168.3
955.25	-219.6

Table 4.3: The V_e for different X-velocities

By using a curve fitting tool on these data the relationship bellow was achieved for drone's estimated velocity and its error.

$$V_e = a \times V_{est} + b$$

$$a = -0.2038 \text{ and } b = -14.97$$

Goodness of fit: R-square: 0.9117 The closer R-square is to 1, the better the fitted curve matches the observations. Therefore, this model seems to fit our data well.

We can apply the correction for this error in the displacement estimation algorithm. Because these results are consistent with the relationship seen in figure 3.3.

$$V_{real} = V_{est} - a \times V_{est} - b = (1 - a) \times V_{est} - b$$

This is also consistent with the data seen in figure 3.3 since:

$$X_{real} = (1 - a)X_{est} - b\Delta t$$

The same procedure was done for the Y-Velocity. The results are available in table 4.4. The curve fitting tool was also applied to this data.

$$a = -0.1497 \text{ and } b = -24.05$$

Goodness of fit: R-square: 0.8142

R-square of 0.81 is also acceptable for our curved fit.

Mean Velocity [mm/s]	Error Velocity [mm/s]	Mean Velocity [mm/s]	Error Velocity [mm/s]
211.40	-30	-456.647	36.3
307.09	-44	-560.159	108.4
391.51	-56.5	724.417	-196.7
606.67	-112.8	738.22	-190.5
985.69	-216.3	-706.85	23
990.55	-240.7	-330.95	52.4
316.727	-39.50	-330.95	52.4
518.53	-64.30	-706.85	23
1019.10	-162.2	-1077	90.2
-690.77	18.70	649.88	-78.9
-884.95	186.20	466.315	-13.8

Table 4.4: The error velocity (V_e) for different Y-velocities

Chapter 5

Autonomous Flight Methodology

5.1 Autonomous Target Locating and Landing

The goal of this section is to develop an algorithm in which the drone would fly a distance forward to a target on x-axis. It would need to fly straight to that spot, stop and land. During the flight the drone should autonomously detect its deviation along y-axis and compensate for them so that it stays on a straightforward path.

Utilizing the results from the previous chapter, it is possible to achieve better estimates with the drone regarding horizontal displacement. For applying the correction factors, it is enough to add the errors to the estimated velocities before the integration process.

$$\dot{V}_x = 1.2038 \times V_{est} + 14.97$$

$$\dot{V}_x = 1.1497 \times V_{est} + 24.05$$

These velocities are integrated to calculate the horizontal displacements. After applying the correction factors so that the drone would have a more

accurate estimate of its position, a control algorithm would be needed so that the drone would not miss its target due to high speed. Based on figure 5.1, the drone's forward speed must be controlled with the simple equation bellow.

$$Speed = \alpha(X_{est} - X_{control}) + Speed_{Max}$$

Where:

$$\alpha = \frac{Speed_{min} - Speed_{Max}}{gap}$$

X_{est} is the calculated x-displacement using the estimated velocity.

X_{target} is the distance we need the drone to fly to and land upon.

$X_{control}$ is the point during the flight where the declining speed begins.

$Speed_{Max}$ is the maximum fraction of speed that we are willing to use for our command speed during our flight.

$Speed_{min}$ is the minimum fraction of speed that we want the drone to use right before it reaches the target.

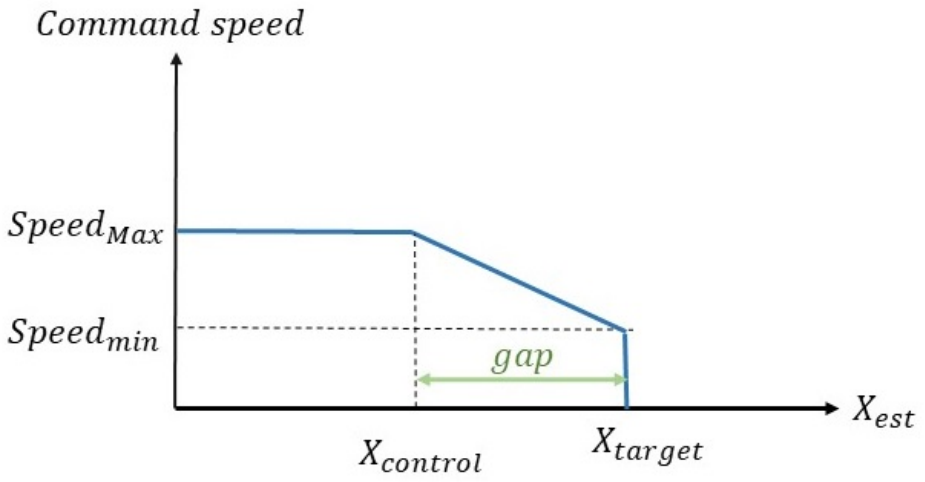

Figure 5.1: Speed control for X-direction target approach

In this series of tests, the drone's mission would be to take off, fly forward, fix deviation in Y-axis, approach the target and then land upon reaching it. The detailed algorithm is available in figure 5.2.

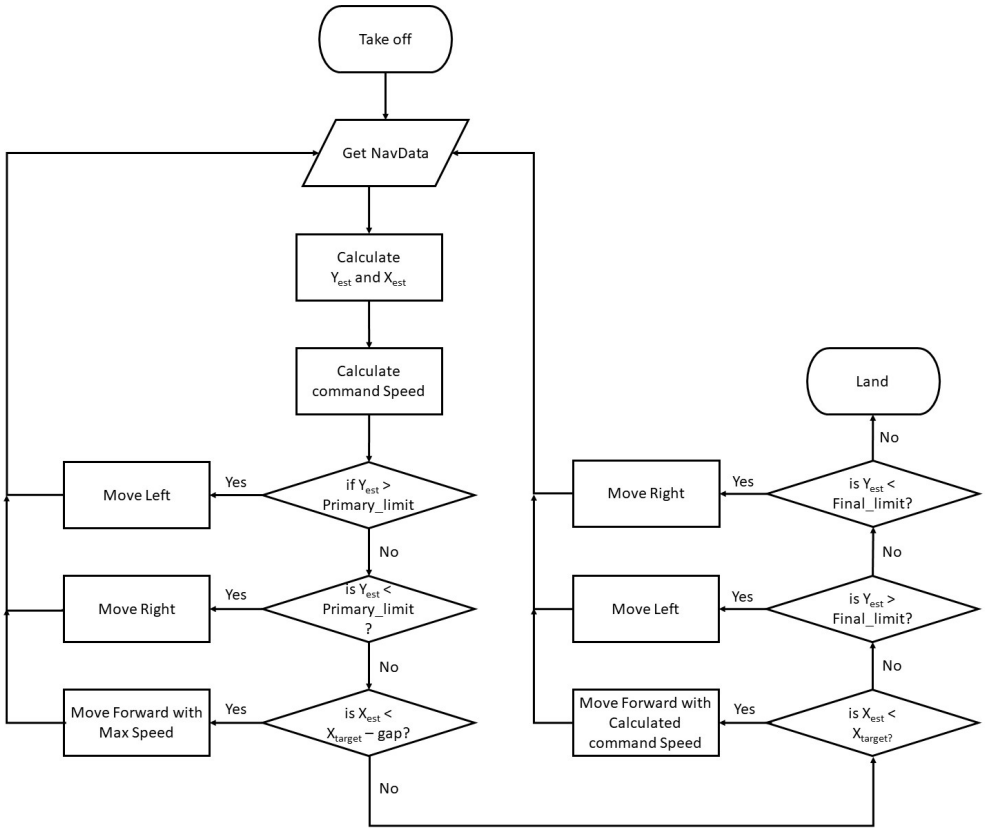

Figure 5.2: The main algorithm for target approach, speed control and two level Y-direction deviation control

As seen in figure 5.2, the flight would start with taking off and starting to receive the NavData stream. The first act is to compute X_{est} and Y_{est} using the corrected velocities based on algorithm from chapter 3. Next, the command speed for the forward flight is calculated based on figure 5.1. In this algorithm during the flight, the drone would fix any deviation bigger than primary limit in Y-direction; however when it had reached the target it would compensate for any residual deviation bigger than the final limit. The primal limit is bigger than final limit so that during the flight the priority

is put upon reaching the target along the x-axis but at the same time big deviations are taken care for safety reasons.

When the drone estimates that it has reached $X_{control}$ it starts to use the decreased calculated command speed for the forward flight.

And finally when the drone estimates that it has reached the target it compensates for y-deviation in order to be inside the final limit range.

For these flights for each target, the operator has five degrees of freedom in choosing the $Speed_{Max}$, $Speed_{min}$, Gap, Primary Y-deviation limit and Final Y-deviation limit.

5.2 Autonomous Pre-planned Route Following

After developing an algorithm that would program the drone to approach a target in the X-direction with Y-deviation control, it os possible to proceed to autonomous path following.

For this purpose, once again another tool of the drone is to be utilized. It is possible to configure the drone to hover over the oriented roundel tag seen in chapter 2. In a normal flight, this feature would let the pilot fly the drone freely until a tag would come into the vision of down-facing camera. Then the internal control algorithm of the drone would take over and align itself on the tag so that the corresponding orientation angle β would be 0. This tool is available by changing the flying mode configuration of drone to 0 for free flight of pilot or to 2 for hovering over tag in case of visibility.

Therefore, a path similar as figure 5.3 would need to be prepared on the floor.

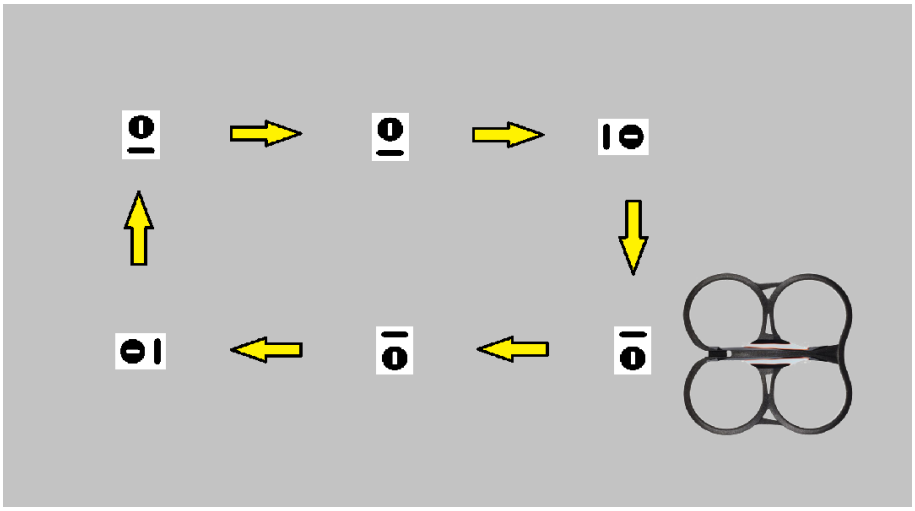

Figure 5.3: A sample path prepared by tags

In this procedure the distances between tags need to stay constant because the complete path following is the repetition of the algorithm seen in figure 5.4; so the target which is the distance between tags, needs to be constant for the whole flight.

It is possible to place the drone anywhere in the path that as long as after taking off and flying forward, it would fly over a tag and detect it, it would be matched with the path and follow it until it gets the landing command from the station. The algorithm flowchart is available in figure 5.4. In which:

Tag flag: A variable utilized for distinguishing between different states of flight for the drone.

Flying mode: Control configuration for enabling or disabling the hovering procedure.

Angle β: The orientation angle provided by the NavData when the tag is visible.

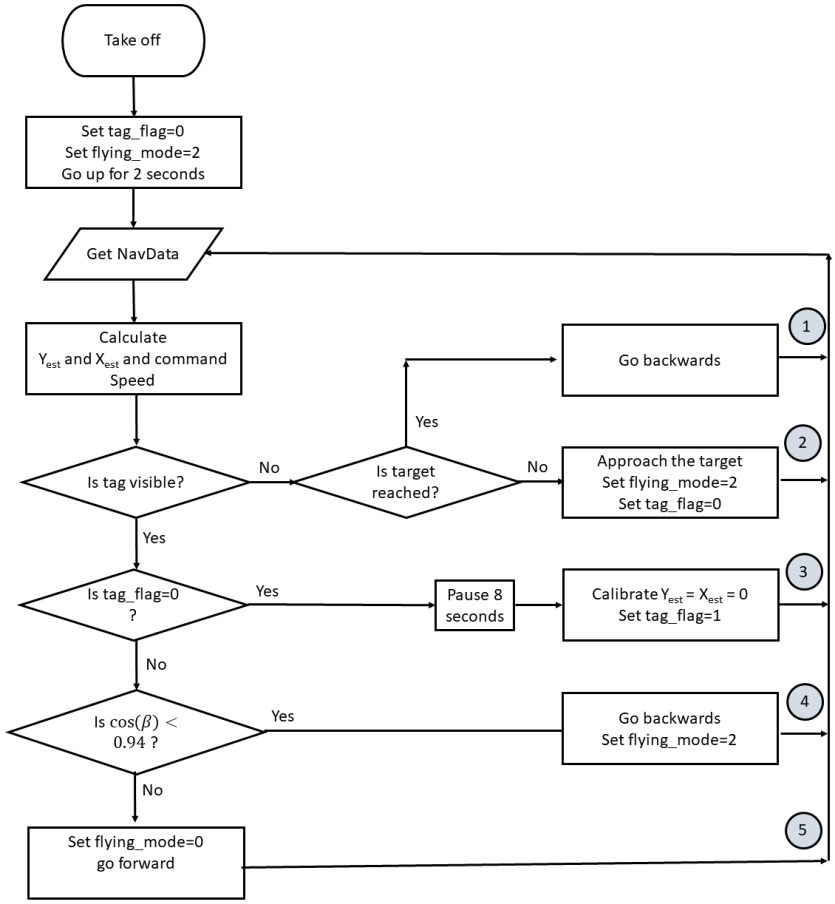

Figure 5.4: Autonomous Path following algorithm

In this algorithm 5 situations are predicted which are shown in the flowchart of figure 5.4.

1. This is a contingency plan for when the drone has reached the target but has not located the tag yet. Therefore it goes backwards for some time in case the tag was missed on the way.

2. In this situation the drone is on the path between two tags approaching the tag on the next target.

3. In this situation the drone has seen the tag and because the tag flag was zero, the program pauses for 8 seconds in which time the drone

aligns itself on the tag in a proper angle and hovers. After 8 seconds the estimated displacements X_{est} and Y_{est} are set to 0. Also the tag flag is set to 1 to embark the alignment on the tag for the next step.

4. This situation also solely exists as a contingency plan for when the drone has noticed the tag and gone through situation 3 but for some reason it has not aligned itself on the tag with the right angle. Thus, we check if the orientation angle was around 0 or 360 degrees before continuing. If the angle is not the one that is desired, the drone is to go backwards and be configured once again to hover and align atop the tag.

5. This is the situation in which the drone is still seeing the tag but since tag flag is one (meaning situation 3 has already happened), it is ready to continue to the next target. Therefore, the drone is configured to have the free flight and go forward. As soon the tag is out of drone's vision boundary, we are back to situation 2 and on the path to the next target.

Chapter 6

Autonomous Flight Test Results

6.1 Autonomous Target Locating and Landing Test Results

In almost all of these tests [1], the minimum command speed was set to 0. Maximum command speed and the control gap was set according to the target to get the best results. In targets closer than 6 meters, the maximum command speed was set to 0.05 with a gap of 1.5 meters, and for further targets, it was set to 0.08 with a gap of 3 meters. In figure 6.1, the landing position for all the target approaches from 2 to 10 meters is shown. For each target at least 12 and at most 20 tests were performed, measured and recorded.

[1] `https://github.com/fereshte75/Autonomous-path-following-ar.drone/blob/master/target_approach.js`

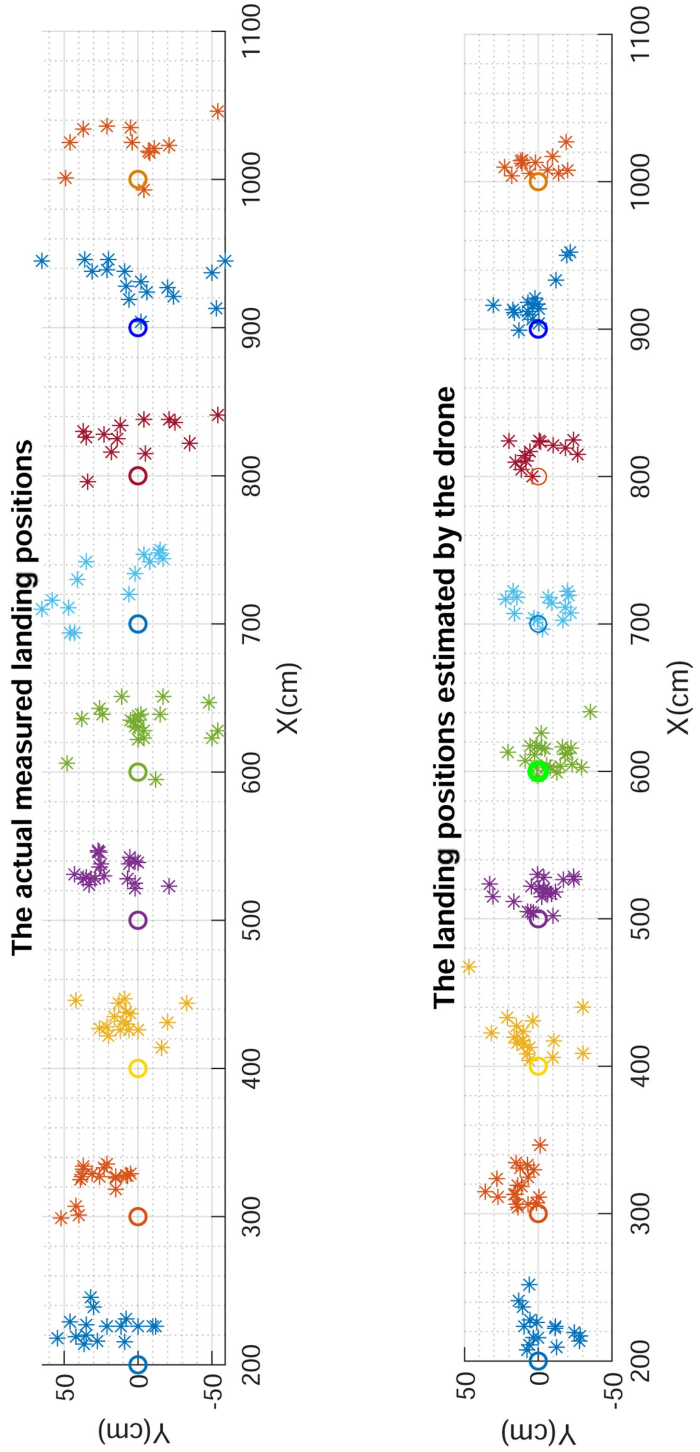

Figure 6.1: The estimated and measured landing positions and their respective targets.

In figure 6.2 to 6.5 the histograms for error of landing regarding the distance from respected targets in both directions of X and Y are provided.

In figure 6.2 specifically, the target of each test is subtracted from its real measured displacement $(X_{real} - Xtarget)$.

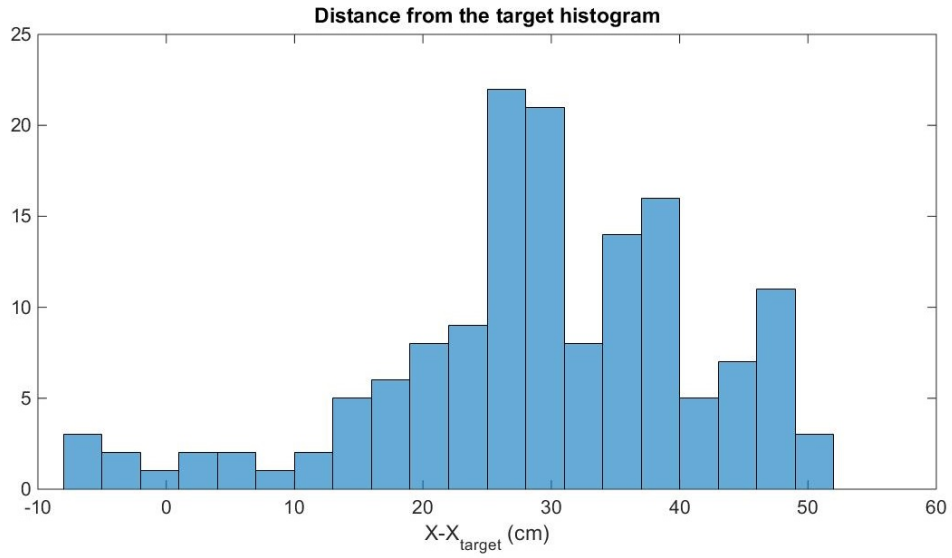

Figure 6.2: The measured distance from respected targets in X-direction Histogram

In figure 6.3 the target of each test is subtracted from its displacement estimated by the drone $(X_{est} - X_{target})$.

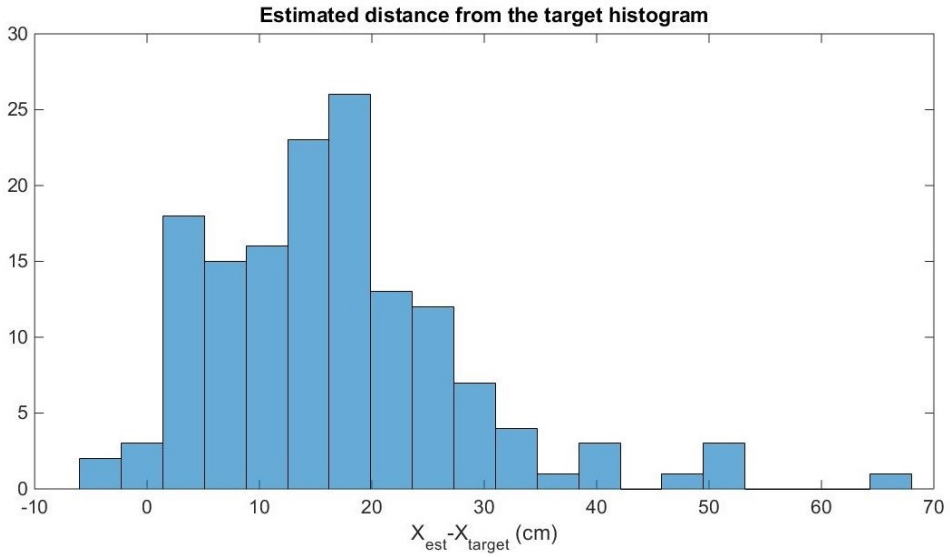

Figure 6.3: The estimated distance from respected targets in X-direction Histogram

In figure 6.4, the measured displacement in Y-axis of the drone is depicted. Similar to figures 6.2 and 6.3, Y_{target} is subtracted from Y_{real}; but in our missions the drone should not have displacement in y-axis therefore, $Y_{target} = 0$. Thus it is simply ignored and the Y-displacements are the data for the histogram.

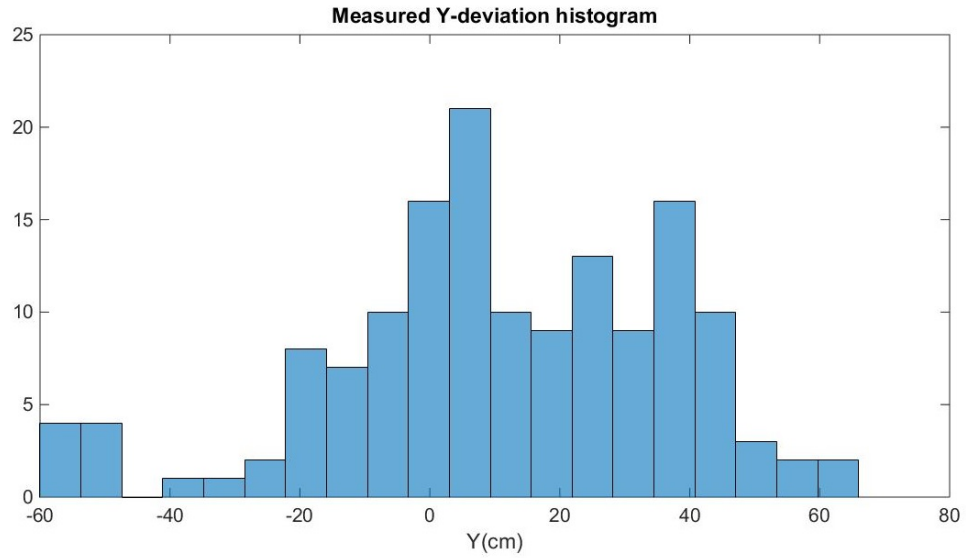

Figure 6.4: The measured deviation from zero in Y-direction Histograms

In figure 6.5, the estimated displacement in Y-axis of the drone is depicted.

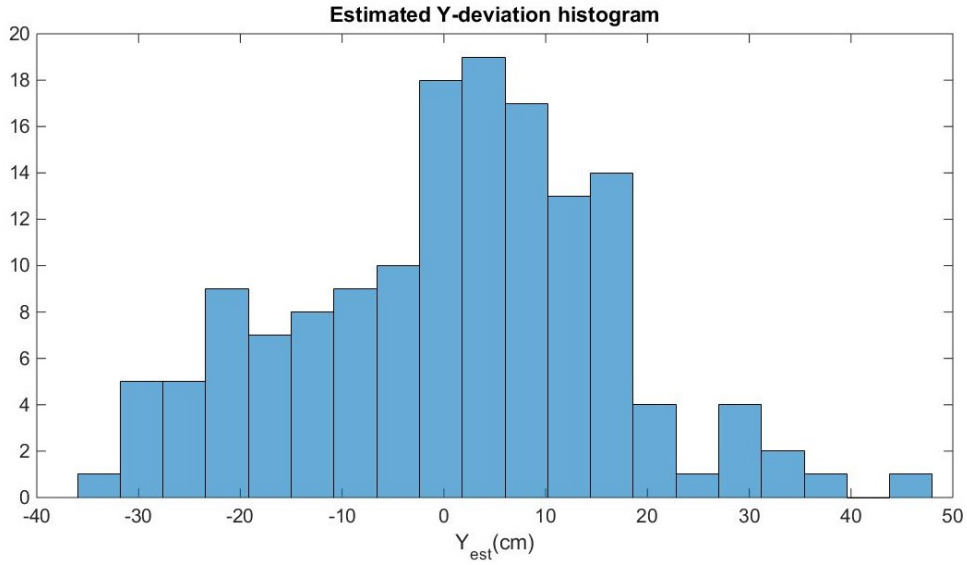

Figure 6.5: The estimated deviation from zero in Y-direction Histograms

As seen in table 6.1, in average the drone lands almost 29 cm after its target in X-direction which is equal to the average of nearly 17 cm estimated by the drone. This distance can be caused by the procedure of final Y-deviation

control after reaching the target and the landing procedure itself that takes the drone further from the target.

Regarding the Y-deviation after landing, it can be understood from table 6.1 that the estimated deviation mean is near zero which is the intended value. However, as the real measurements' mean shows, the drone tends to land to the right in average.

	Mean [cm]	Standard deviation [cm]
Y	11.16	25.89
Y_{est}	0.78	15.47
$X - X_{target}$	28.81	12.53
$X_{est} - X_{target}$	16.87	11.56

Table 6.1: Mean and standard deviation

6.2 Autonomous Pre-planned Route Following Test Results

A path similar to the test path considered in figure 5.3 was prepared.

Figure 6.6: The test path prepared in an auditorium. The floor texture is increased with colored stickers for the sake of vision estimated velocity

In this test the target distance between the tags was chosen 4 meters. In other tests and paths the target was chosen 2 meters, some other 3 meters, and in some up to 5 meters [2]. As expected if the distance between tags is increased the drone is more susceptible to faults; since the drone aligns and stabilizes itself over each tag and they are similar to checkpoints for following the path. However, with closer tags the flight time increases and more preparation is needed for the environment before flight. Therefore the distance between tags is a trade-off between flight time and reliability.

Figure 6.7: The test path prepared with four Oriented Roundel tags

Additionally other paths with corners with angles different than 90° were also prepared as seen in figure 6.8.

[2] `https://github.com/fereshte75/Autonomous-path-following-ar.drone/blob/master/path_following.js`

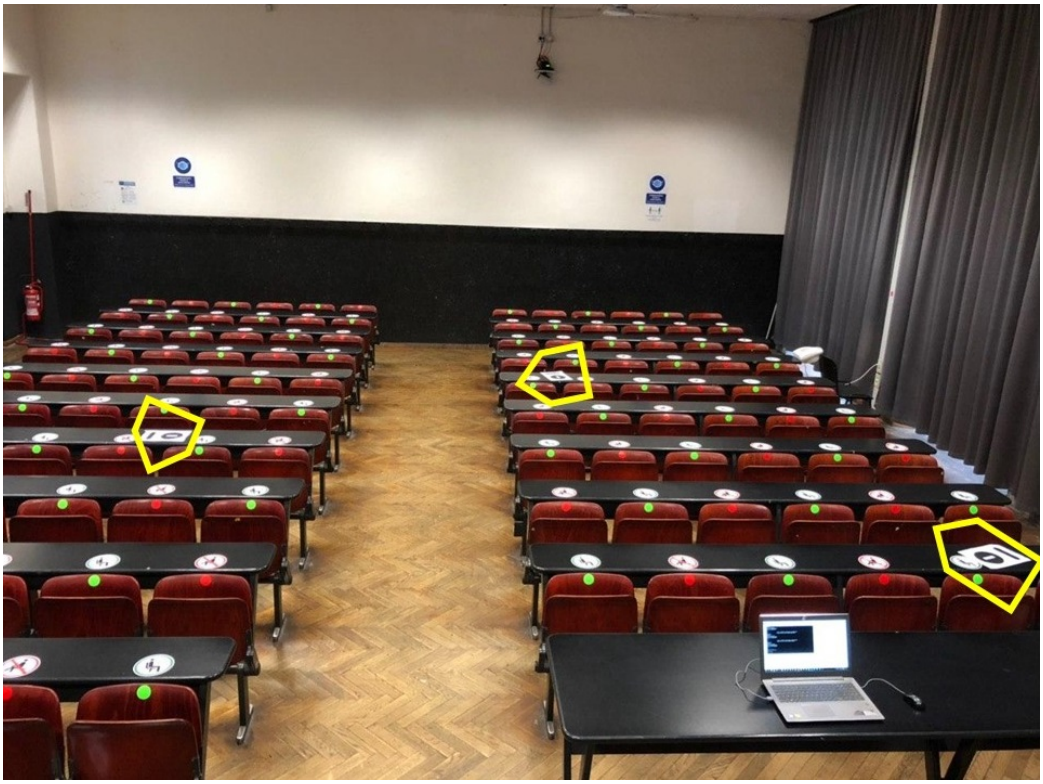

Figure 6.8: The test path prepared with four Oriented Roundel tags in random angles setting

The videos proving the feasibility of this algorithm are available in the link bellow:

`https://www.youtube.com/playlist?list=PLMgMMhA1mz0TW52uztdYwuiSyRUfi9Xbp`

Chapter 7

Conclusion and Future Research Suggestions

The goal of this thesis was to develop an algorithm for an autonomous low-cost quad-copter to follow a pre-planned route in GPS-denied environments. The objective included achieving it without expensive external motion tracking system. Therefore in order to do so, it was imperative to find the metrological characteristics of the drone at hand.

AR.Drone 2.0 with its low-cost inertial measurement system (IMU) was a good choice of subject. For defining the accuracy of said sensors, numerous test were run and it was observed that the double sensors implemented on the drone for estimating the altitude were quite accurate and reliable. However, the horizontal velocities estimated by drone which was the result of fused data from IMU and the bottom facing camera were somewhat erroneous and needed correcting factors to be used in horizontal displacement estimation calculations.

After correcting the estimated velocity values, it was possible to describe algorithms for autonomous flight with acceptable accuracy. In target approach tests done for up to 10 meters the drone landed averagely 28.81 centimetres after and 11.16 centimetres to the right of the intended target. Finally, based on autonomous target approach with deviation control, a new algorithm was developed to have the drone autonomously follow a prepared path using the

oriented roundel tag as indicators of the path to the drone. The final algorithm proved functional in multiple tests over diverse floors, in different rooms and with different angles as turning points in corners.

Further studies on this research are recommended as following.

- As seen in some of the test results the correction factors do not result in complete accuracy of the estimated horizontal displacement. It could provide fruitful to study the estimated velocity's sensitivity in regards to the battery voltage.
- It could be possible to mount proximity sensors on the drone's haul to add another control loop considering the objects around the quadcopter for obstacle avoidance purposes.
- As mentioned in chapter 5, the path following algorithm only takes into account 5 outcomes and in regards to not seeing the tag it can only go backwards. It is possible to predict more situations and add branches to this algorithm or come up with more conclusive contingency plans.
- It is possible to plan routes based on this work's final algorithm in corridors of a building and add a video stream block of code to the program to use the drone as an autonomous surveillance camera. This project would have constant patrol paths and live and recorded video streams at the main station.

Bibliography

[1] Convertawings model 'a' quadrotor at the cradle of aviaton museum.

[2] History of quadcopters and other multirotors, 2018.

[3] Amazon.com: Prime air, 2019.

[4] Parrot S. A. A.r.drone developer guide.

[5] S. Al Habsi, M. Shehada, M. Abdoon, A. Mashood, and H. Noura. Integration of a vicon camera system for indoor flight of a parrot ar drone. In *2015 10th International Symposium on Mechatronics and its Applications (ISMA)*, pages 1–6, 2015.

[6] R. Athira Krishnan, V. R. Jisha, and K. Gokulnath. Path planning of an autonomous quadcopter based delivery system. In *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)*, pages 1–5, 2018.

[7] Jack Brown. Parrot ar. drone 2.0: Reviews, price, features, competitors, specs, 04 2016.

[8] Fintan Corrigan. Drones for deliveries from medicine to post, packages and pizza, 05 2019.

[9] Luke Dormehl. The history of drones in 10 milestones, 09 2018.

[10] Lachlan Dowling, Tomas Poblete, Isaac Hook, Hao Tang, Ying Tan, Will Glenn, and Ranjith Unnithan. Accurate indoor mapping using an autonomous unmanned aerial vehicle (uav), 2018.

[11] P. Falcón, A. Barreiro, and M. D. Cacho. Modeling of parrot ardrone and passivity-based reset control. In *2013 9th Asian Control Conference (ASCC)*, pages 1–6, 2013.

[12] Francesco Sabatino. *Thesis KTH*. Master's degree thesis, KTH, Stockholm, Sweden, June 2015.

[13] Felix Geisendörfer. felixge/node-ar-drone.

[14] M. Hehn and R. D'Andrea. A flying inverted pendulum. In *2011 IEEE International Conference on Robotics and Automation*, pages 763–770, 2011.

[15] A. Hernandez, C. Copot, R. De Keyser, T. Vlas, and I. Nascu. Identification and path following control of an ar.drone quadrotor. In *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 583–588, 2013.

[16] Sean Hollister. Skydio 2 review: a drone that flies itself, 12 2019.

[17] Berthold Horn and Brian Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 08 1981.

[18] J. Hvizdoš and P. Sincák. Control library for ar.drone 2.0. In *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 77–82, 2015.

[19] J.Philipp de Graaff. *PS-Drone-Documentation*.

[20] Jeremiah Karpowicz. Predicting the future of the drone industry in 2020 with mike blades — commercial uav news, 12 2019.

[21] Yohanes Khosiawan and Izabela Nielsen. A system of uav application in indoor environment. *Production & Manufacturing Research*, 4:2–22, 2016.

[22] A. Koval, E. Irigoyen, and T. Koval. Ar.drone as a platform for measurements. In *2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO)*, pages 424–427, 2017.

[23] Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl. AR-drone as a platform for robotic research and education. In *Communications in Computer and Information Science*, pages 172–186. Springer Berlin Heidelberg, 2011.

[24] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). volume 81, 04 1981.

[25] M. Müller, S. Lupashin, and R. D'Andrea. Quadrocopter ball juggling. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5113–5120, 2011.

[26] Pierre-Jean Bristeau, FranÃ§ois Callou, David VissiÃre, Nicolas Petit, editor. *The Navigation and Control Technology Inside the AR.Drone Micro UAV*, 2011.

[27] R. Ritz, M. W. Müller, M. Hehn, and R. D'Andrea. Cooperative quadrocopter ball throwing and catching. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4972–4978, 2012.

[28] V. Rustagi, M. Ludhiyani, A. Sinha, and R. Dasgupta. Model based drift free localization for multirotors. In *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, 2018.

[29] K. V. M. Sai Kumar, M. Sohail, P. Mahesh, and U. R. Nelakuditi. Crowd monitoring and payload delivery drone using quadcopter based uav system. In *2018 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 22–25, 2018.

[30] M. F. Sani and G. Karimian. Automatic navigation and landing of an indoor ar. drone quadrotor using aruco marker and inertial sensors. In *2017 International Conference on Computer and Drone Applications (IConDA)*, pages 102–107, 2017.

[31] L. V. Santana, A. S. Brandão, M. Sarcinelli-Filho, and R. Carelli. A trajectory tracking and 3d positioning controller for the ar.drone quadrotor. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 756–767, 2014.

[32] S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *2011 IEEE International Conference on Robotics and Automation*, pages 20–25, 2011.

[33] Miroslav Trajkovic and Mark Hedley. Fast corner detection. *Image and Vision Computing*, 16:75–87, 02 1998.