Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Autonomous navigation of a Tello Ryze drone based on AprilTags

Tibo Poncelet

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

PROMOTOR :
Prof. dr. Nick MICHIELS

PROMOTOR :
Prof. dr. Joni KÄMÄRÄINEN

COPROMOTOR :
Dhr. Lauri SUOMELA

Gezamenlijke opleiding UHasselt en KU Leuven

UHASSELT KU LEUVEN

Universiteit Hasselt | Campus Diepenbeek | Faculteit Industriële Ingenieurswetenschappen | Agoralaan Gebouw H - Gebouw B | BE 3590 Diepenbeek

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE 3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE 3500 Hasselt

UHASSELT KU LEUVEN

2022
2023

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Autonomous navigation of a Tello Ryze drone based on AprilTags

Tibo Poncelet

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

PROMOTOR :
Prof. dr. Nick MICHIELS

PROMOTOR :
Prof. dr. Joni KÄMÄRÄINEN

COPROMOTOR :
Dhr. Lauri SUOMELA

UHASSELT KU LEUVEN

Autonomous navigation of a Tello Ryze drone based on AprilTags

Tibo Poncelet

June 12, 2023

Preface

During this last semester, I have learned a lot and am incredibly grateful for having the opportunity to study this topic.

I would like to thank my supervisors: prof. dr. Joni Kämäräinen, prof. Lauri Suomela and prof. dr. Nick Michiels for their guidance this semester.

I would also like to thank my family and friends for supporting me.

Contents

List of Tables

List of Figures

Concepts

Algorithm	A sequence of steps or instructions to solve a problem or reach a certain goal.
Lift	The component of a force caused by the flow of a fluid or gas which is perpendicular to the direction of that flow.
Odometery	The use of motion sensors in order to estimate the position and orientation of a robot over a period of time.
Sensor	A device that detects physical quantities and converts them to an electrical signal.
Thrust	A reaction force caused by expelling mass in a certain direction.

Abstract

Autonoom navigerende drones komen steeds vaker voor in toepassingen zoals infrastructuurinspectie, landbouw en pakketbezorging. Het doel van deze masterproef is om software te ontwikkelen die sensordata van een Tello Ryze drone kan lezen en gelijktijdig de drone kan besturen. De software maakt daarnaast ook autonome navigatie op basis van AprilTags mogelijk. De drone moet autonoom een pad kunnen navigeren door middel van AprilTags die verspreid hangen over het pad. Het programma is ontwikkeld in Python en C++ met behulp van het Robotic Operating System. Dit systeem maakt gebruik van nodes die code onafhankelijk kunnen uitvoeren terwijl ze nog steeds met elkaar kunnen communiceren via topics. Dit vergemakkelijkt het uitwisselen van navigatiealgoritmes. Autonome navigatie op basis van AprilTags werd getest door de drone een vooraf bepaald pad te laten volgen dat door AprilTags is gemarkeerd. Deze tags laten de drone toe om naar de volgende tag te navigeren. De software werd beoordeeld via een reeks testen in verschillende omgevingen om de betrouwbaarheid en doeltreffendheid van het syteem te garanderen. De testresultaten tonen aan dat de drone moet navigeren in voldoende verlichte omgevingen en over oppervlaktes met voldoende kenmerken. In dit geval is het systeem in staat de drone autonoom te navigeren. De gemiddelde vertraging tussen toetsenbordinput en dronereactie is 80,49 ms. In conclusie is autonome navigatie met de drone op basis van AprilTags met deze software mogelijk.

Abstract in English

Autonomously navigating drones are becoming more common in applications such as infrastructure inspection, agriculture, and package delivery. This thesis aims at developing software to control and receive data from a Tello Ryze drone while also enabling autonomous navigation based on AprilTags. The drone has to be able to autonomously navigate a path based on visual instructions coded on AprilTags spread across the path. The programming was done in both Python and C++ using the Robotic Operating System. This system enables usage of individual nodes which can execute their code independently while still being able to communicate with each other via specific topics. This facilitates the interchangeability of navigational algorithms and input device driver nodes. Autonomous navigation based on AprilTags was tested by instructing the drone to follow a predetermined path marked by AprilTags. Using this, it can localize itself and navigate to the next tag. The software has been validated through a series of tests to ensure its reliability and effectiveness in varying lighting conditions and environments. The test results show that the system requires the drone to navigate in sufficiently illuminated areas and above surfaces with distinct features. When these conditions are met, it reliably enables the user to control the drone trough keyboard input. The average delay between keyboard input and drone reaction is 80,49 ms. In conclusion, autonomous navigation with the drone based on AprilTags is possible with this software.

Chapter 1

Introduction

1.1 Situation

The computer vision group of Tampere University (TAU) is a research group that focuses on the development of new algorithms and models for processing and analyzing 2D and 3D images and videos. These algorithms and models have applications in robotics and intelligent software and systems. The research group currently investigates applying visual navigation algorithms to drones.

Unmanned aerial vehicles (UAVs) commonly referred to as drones, have revolutionized various industries. However, in order to fully use the potential of these devices, autonomous navigation must be applied. Extensive research has been performed on this topic for various applications. In photography they enable aerial angles that were previously difficult or even impossible to achieve without a drone. In the same way, infrastructure can be inspected. Package delivery is not dependent on traffic on the road nor the state of the road. In agriculture it allows farmers to monitor crops and livestock and spray fertilizer from the air. It is even possible for a drone to recognize flowers and pollinate them [8]. Drones could also be used in search and rescue applications, where multiple drones could be searching for targets simultaneously [9]. Museum tour guide, as well as interplanetary sample collection are other examples of different applications for autonomous navigation [10].

1.2 Problem statement

In order to perform autonomous navigation, the system needs to be able to know its environment and localize itself in this environment. This problem is also known as the "simultaneous localization and mapping" (SLAM) problem. This problem is usually solved with a "light detection and ranging" (LIDAR) component which detects walls and obstacles with laser pulses and a combination of global positioning system (GPS) signals with an odometery tracker such as an inertial measurement unit (IMU) in order to determine the position and orientation of the vehicle. The main challenges of autonomous indoor navigation of a UAV is that there is no GPS signal available and high quality LIDAR components can be expensive. In addition, the weight of the components are also a limitation as the UAV has to be able to carry its own weight. This would require rotors capable of generating more lift which increases the overall price.

The drone selected by the research group is the Tello drone, developed by the company Ryze Tech. The main benefit of this drone is that it has a good price to quality ratio and costs around € 100. This relatively low price makes it more accessible for educational purposes.

However, the primary limitation to this drone is that it uses visuals to orient itself. It relies on its vision positioning system which consists of a camera and a 3D infrared module [11].This system is heavily dependent on lighting conditions and performs poorly when flying in areas with illumination conditions lower than 10 lux or exceeding 100 000 lux [11].

The goal of this research is to determine whether visual based navigation algorithms could be applied on the low cost Tello Ryze drones to enable autonomous navigation. The primary limitation with this drone is that it uses visuals to orient itself as opposed to other devices, like a LiDAR laser scanner or a sonar.

1.3 Goals

The main goal of this research is to develop an easy to understand system that enables a Tello Ryze drone to navigate autonomously based on the AprilTags it detects in the images it transmits. During an autonomous flight there should be no human interaction necessary between the liftoff and landing of the drone. In order to achieve this, the drone needs to be able to send images and apply received instructions within a reasonable timeframe. In addition, the algorithm should also be fast enough to process images to instructions. The time necessary for sending an image, processing the image to generate an instruction, as well as receiving the image should be small enough to avoid collision while the drone is navigating. The efficiency should be measured in different light levels as this impacts the quality and observable details in the images. They should also be executed in different areas as the Vision Positioning System of the Tello Ryze drone performs poorly while flying over surfaces wich are: monochrome, highly reflective, transparent and/or moving [11]. The secondary goal is to design the software in such a way so that it can be used to test multiple navigational algorithms, not only AprilTags.

1.4 Method

The method consists of four main tasks. Setting up the developing environment is the first task. The Tello Ryze drone will be used for this research. The Robotic Operating System (ROS) is an open source collection of tools, libraries, drivers and algorithms that are useful for interacting with robotics like drones. For this reason, ROS will be used for developing the software which will interact with the drone. ROS Noetic Ninjemys is the chosen version for development because it has Long Time Support (LTS). ROS is very sensitive to the operating system and encourages the usage of a specific version of Linux. For ROS 1 , Ubuntu 20.04 is recommended. The development of the software will be done in C++ and Python, which are both supported by ROS.

The second task is drone control. The goal is to send instructions to the drone which are generated by a ROS/C++/Python program while receiving images from it.

The third task is implementing the algorithm for generating instructions based on AprilTags detected in images. The images from the drone in the previous step will be processed by an algorithm which will produce corresponding instructions required for successful navigation. Al-

gorithms that are capable of detecting AprilTags will be used to apply autonomous navigation to the Tello Ryze drone [12] [7].

The fourth and final task is testing the autonomous navigation system and measuring its reliability. In order to determine if the system is suitable for practical applications, test flights need to be carried out. For this research, autonomous will be defined as no human interaction with the drone between liftoff and landing the drone. The parameters for determining the efficiency of the navigation will be based on the time the system needs from taking the picture to responding to that image with an instruction.

1.5 Preview

The following chapter, the literature study, will discuss about relevant technology. The different types of drones and their benefits over another will be discussed. Next, the different type of sensors usually applied will be discussed. After that, the most popular methods for solving the SLAM problem will be discussed as well as path and motion planning algorithms. And the last part of the literature study will discuss other algorithms including the AprilTag method.

Chapter 3 will discuss the method and explain the workings of the software which enables a Tello Ryze drone to navigate autonomously based on AprilTags. Finally, this part also explains the methods used for the measurements as well as the test flights.

Finally, chapter 4 will discuss the results of the measurements and test flights. After that, a general conclusion is also given.

Chapter 2

Literature study

2.1 Introduction

In this chapter, relevant technology and methods for visual based autonomous navigation will be discussed as well as non-visual based methods. Even though this thesis will only apply visual based methods, it is important to understand the benefits of methods that use alternative approaches and why extra sensors could be beneficial. The first section will handle available drone hardware. This includes generic drone types and sensors. It will also discuss the mechanics which allow drones to navigate. The second section contains more software related research, primarily algorithms. For this second section it is important to understand which benefits the different sensors bring while using a certain algorithm.

2.2 Drone

This section will discuss the most common types of drones that are available on the market with their benefits and downsides. Next, commonly used sensors in drones will be discussed together with a brief overview of their workings. The final part of this section will go more in depth about the mechanics and principles behind drone navigation.

2.2.1 Types of drones

Multi-rotor unmanned aerial vehicles (UAVs), often referred to as drones are aircrafts that are controlled by a pilot on the ground instead of having an onboard pilot. There are three main types of drones [4]. The first type are the fixed-wing drones which has the same flight principals as an airplane. An example of the fixed-wing drone is given in figure 2.1.

This drone has a front propeller which provides the thrust necessary to move the drone forward. The movement control of this drone is achieved by movable surfaces, also known as control surfaces. This drone can achieve high speeds, can be operated for a longer time than other drones and has a higher flight range but cannot hover nor fly backwards [4].

The second type of drone is also known as the helicopter drone. This drone has, like a helicopter, one main rotor and one tail rotor. This drone can hover and fly in each direction, unlike the fixed-wing drone, by changing the collective and cyclic pitches of the main rotor. The rotation of

Figure 2.1: Example of a fixed wing drone [1]

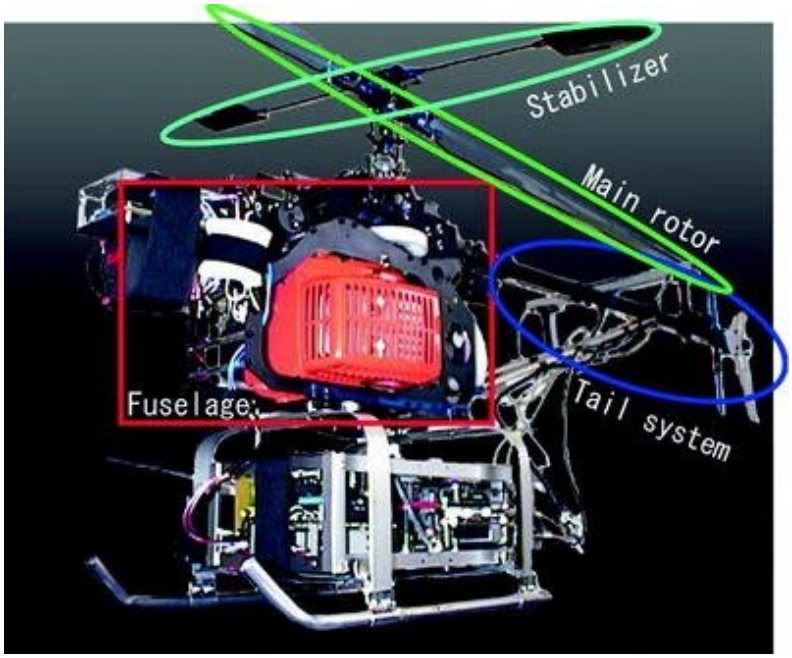

Figure 2.2: Example of a helicopter drone [2]

the main rotor causes a reaction moment on the helicopter drone which needs to be compensated for by the rotor on the tail of the drone [4]. An example of the helicopter drone is given in figure 2.2.

The complexity of the general mechanisms and tail rotor compensation mechanisms results in this kind of drone being more difficult to control than the other drones [4].

The last type of drone is the multi-rotor drones. These drones can lift and rotate independently by adjusting the speed of the different rotors. This mechanism is far easier than the one of the helicopter drone while still being able to hover and fly backwards unlike the fixed-wing drone [4]. An example of a multi-rotor drone is given in 2.3.

The downside of this kind of drone is that the time of flight is more limited than the other drones, usually between five and twenty minutes. This is caused by the fact that multiple rotors are relatively heavy compared to the other parts of the drone and overcoming this extra weight costs more energy [4]. Regardless of these downsides, the multi-rotor drone still has the most potential for visual based applications [4].

Figure 2.3: Tello Ryze drone

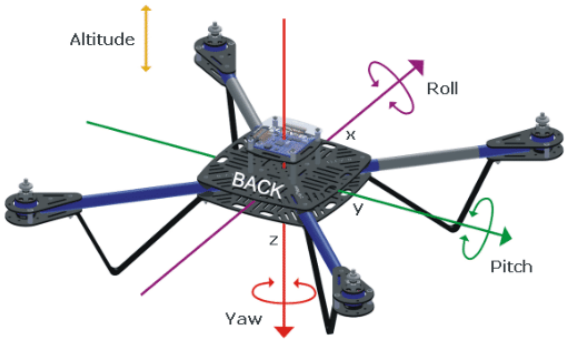

Figure 2.4: Drone rotations [3]

2.2.2 Drone navigation

In order to apply autonomous navigation to a drone, it is also important to understand the mechanics and principles behind manually navigating a drone. This section will discuss how manipulating the rotational speeds of the rotors on the drone enable it to move and rotate in the desired directions.

The four basic movements of a drone: vertical lift, roll, pitch and yaw movements are depicted in figure 2.4. Since drones move in a 3D space, there are three axis depicted on the figure.

Rotation along the x-axis is called roll, rotation along the y-axis is called pitch and along the z-axis it's called yaw. Roll rotation is used to move the drone in the left-right direction. Pitch rotation is used to move the drone forward or backwards. And finally yaw is used to rotate the drone body along the z-axis.

Drones feature multiple rotors which are aligned symmetrically and co-linearly. These rotors generate thrust force by rotating which then lifts the body of the drone [4]. There are many parameters which correlate to the amount of thrust force generated. The most significant ones are the pitch angle, the speed of the rotors and the diameter of the rotor propeller. All four of the rotors should generate the same amount of thrust in order to move the drone along the z-axis depicted in figure 2.4.

Figure 2.5 depicts the result of changing the speeds of the rotors. The double-lined arrows indicate that the rotational speed of the motor is increasing while the single-lined arrows indicate

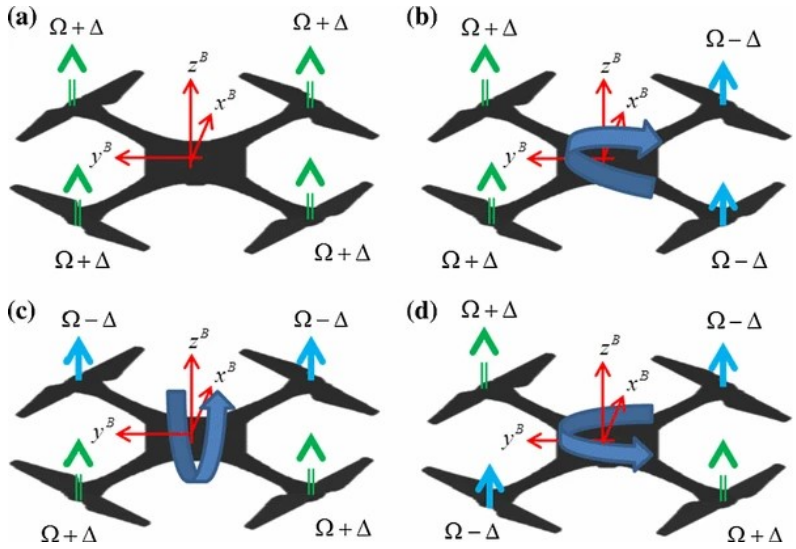

Figure 2.5: Drone flight modes [4]

a decrease in motor rotational speed [4]. Rotor rotations generate an unwanted reaction moment. In order to counteract this, adjacent rotors must rotate in opposite directions. This is achieved using mirrored props. Vertical lift is achieved by changing all of the rotor speeds by the same amount as depicted in figure 2.5 (a). This results in a change of altitude. By increasing rotor speed of two opposing rotors by the same amount results in roll motion as depicted in figure 2.5 (b). Similarly, pitch motion is achieved by decreasing the speed of the same rotors as shown in figure 2.5 (c). And finally, the yaw motion is controlled by increasing rotor speed of rotors on the same diagonal while decreasing the rotor speed of the rotors on the other diagonal. [13, 14, 3]

2.2.3 Sensors

A crucial requirement for a drone to be able to perform autonomous navigation is it's ability to orient itself in the environment and estimate its position. This is achieved by utilising the drone's onboard sensors. This section will discuss which tools and sensors commercially available drones use and briefly discuss their workings.

The most popular method for position estimation is using the inertial measurement unit (IMU) [15], [4]. This unit consists of three different sensors: an accelerometer, a magnetometer and a gyroscope. The accelerometer measures the absolute angle of the drone body relative to the gravity vector. This sensor is easily corrupted by noise caused by vibrations as these are small and brief accelerations. This sensor would require a low-pass filter in order to obtain usable measurements [16]. The gyroscope measures the angular velocity of the drone body, enabling relative attitude measuring [4]. Together, these measurements can be used to obtain information about the pitch and roll orientation/motion of the drone [4]. The yaw angel can be obtained by solely using gyroscope measurements, but this results in inaccurate measurements due to the bias of the gyroscope [4]. To prevent this, a magnetometer's data can be combined with the other sensors' data in order to obtain accurate measurements [4], the idea of combining sensors in order to increase the accuracy and reliability of a system is also called sensor fusion. Usually a Kalman filter [17] is used to achieve this. Drones using only a gyroscope are called 3-axis drones. They can only be stabilized by controlling the angular velocity and are the most difficult to control [4]. Drones using an accelerometer in addition to a gyroscope are called 6-axis drones. These drones

allow the user to also control pitch and roll motions but not the heading direction [4]. Drones using a gyroscope, accelerometer and a magnetometer are usually called 9-axis drones. These type of drones are the easiest and most reliable to control as they allow the operator to directly control the heading direction [4].

The drone also needs to be able to measure and track its own velocity and orientation as well as localize its 3D position in an area. Using an indoor motion capture system would be a highly accurate option and make the problem trivial [4], [15]. This type of system is able to provide measurements which are accurate to the millimeter. But this is not practical due to the need for indoor cameras which would be expensive. This cost would scale up drastically for larger indoor areas.

Another popular option for solving the localisation problem is using the global positioning system (GPS) signal. This signal requires line of sight with at least four satellites in order to provide proper working [4]. This is unreliable or at least difficult for indoor applications and is thus not relevant for this project [4], [15].

Simultaneous localisation and mapping (SLAM) problems rely on the combination different sensors onboard of the drone as for example:the LIDAR, camera and inertial measurement unit (IMU). A LIDAR fires lasers which reflect on the surfaces of the environment. By tracking the time between firing the laser from the LIDAR, receiving the laser after it has reflected of a surface and weighing in the speed of light, the distance between the drone and surface can be calculated. Using this procedure, an accurate three dimensional map can be formed of the environment of the drone. Unfortunately, the Tello Ryze drone is not equipped with a LIDAR. This would be too heavy and also too complex to implement on the current drone without significantly increasing its price. The Tello Ryze drone is equipped with an inertia measurement unit (IMU), vision positioning system and a camera. Video cameras have the advantage of being passive, they measure their surroundings instead of changing it like infrared or ultrasound based sensors. It is also useful for most of the indoor drone applications [18].

In order to reduce the latency for decoding images to instructions, it is possible to carry an external development-board containing a processor with the purpose to do the Image decoding.

2.3 Autonomous navigation

Autonomy in navigating robotics is a spectrum. It ranges from tele-operated vehicles to full autonomy. The former is where an operator is in control while an algorithm follows guidelines and prevents certain actions such as colliding with objects, while the latter is having the system make its own decisions based on sensory data without the interaction of a human operator. Full autonomy in robotics is split up into two categories. The first one being the heuristic approach, where a robot or vehicle follows simple instructions. The benefit of this approach is that there is no need to gather vast amounts of information about the environment in order to achieve autonomy. The downside is that this approach is not guaranteed to be optimal. An example of this would be an automatic lawnmower that detects the edge of the lawn, turns a random amount of degrees and proceeds to mow in that direction until it reaches another edge. This approach does not guarantee that the whole lawn will be mowed in the most optimal way, but it does perform a sufficient job for this application. The alternative approach is the optimal or classical approach where, in contrast to the heuristic approach, a vast amount of data about

the environment is necessary to be collected in order to plan an optimal path from the current position to the desired goal. In this algorithm, the robot or vehicle builds and repeatedly updates a map of the environment and locates itself in it. Then, it calculates the optimal path to the goal considering obstacles and other rules. And finally, it converts the path into movement instructions and executes those. Note that these two approaches can be used within the same system by using the heuristic approach when the data of the area is limited and switching to the optimal approach when navigating through known areas. The classical approach can also be used on a known low resolution high level map of the environment in order to generate a global path. While the heuristic part can be used to avoid dynamical obstacles that the global path does not consider [19].

2.4 SLAM algorithms

2.4.1 Introduction

The autonomous navigation challenge can be defined as four main tasks: localization, mapping, path planning, and locomotion [18]. Mapping is where the system uses its sensors in order to detect walls and obstacles and use this information to build a map of the environment. Localization is knowing where the robot is relatively to the previous known position in the generated map. The localization and mapping problems are usually solved together because it results in a better solution then solving them separately. This is known as the simultaneous localization and mapping (SLAM) problem [18].

A difference is made between metric and topological maps [20]. A metric map accurately depicts the environment that the robot explored. This can be used by the robot to accurately track its location and perform precise route repetition. This map scales in complexity the larger the area is.

A topological map uses a set of images as goals in order to navigate to the final destination. When the current goal is reached, the next image in the set is updated as the current goal. This process keeps repeating until the destination is reached. The main downside with this approach is that it fails when even one single goal is unable to be reached [20].

These two methods combined result in a topometric map [20]. Here, movement information is used together with sensor readings in order to form a locally consistent map. By making the map only locally consistent and not globally consistent, it can more efficiently scale to larger environments compared to a metric map.

Planning the path is heavily depended on the localisation and mapping tasks. If the robot does not know where it is and its surroundings, it is impossible to plan a path efficiently.

Locomotion consists of the physical movement of the robot and obstacle detection.

More relevant for this thesis is the visual SLAM. The goal is to build a 3D map of the environment with only or primarily the camera as sensor [21],[6].

The most common approach for localisation based on visuals is through the use of landmarks. These can be lines, corners, edges or others depending on the applied algorithm [18],[12],[7].

Automated Guided Vehicles (AGVs) used to rely on specific landmarks to navigate around. This

results in AGVs being configured for specific areas for them to operate. In addition, those areas also need to be prepared for the AGVs [18].

No sensor is perfect and all sensors have some kind of error. But combining measurements of different sensors to try and predict unknown variables leads to increased accuracy. The Kalman filter is an algorithm that tries to predict unknown variables by observing and tracking measurements over time. This algorithm essentially knows when a certain sensor is more reliable than others at a certain time and uses that information to make more accurate predictions. An advanced variant of this algorithm is commonly used in SLAM algorithms [18].

2.4.2 Localization with a Monte Carlo particle filter

This technique relies on a system with a LIDAR and odometry measuring in order to position itself on an existing map [5]. It uses the noisy LIDAR and odometry measurements with a Kalman filter in order to achieve a position estimate on the map. The particle filter works by reading LIDAR measurements to determine the possibility of the system being in a certain position on the map. It virtually places randomly orientated copies of the system on the map, which are called particles. The virtual LIDAR measurements of all the particles are then calculated and each one of them is compared with the LIDAR measurements of the real system. The more similarity in measurement, the higher the chance that particular particle was correctly placed on the map. A probability distribution for position estimates is generated using this similarity. The filter repeats the placement of particles with more of them being placed in higher probability areas than lower probability areas. While the actual system navigates, the particles also navigate according to the odometry measurements while accounting for noise. The filter continually keeps comparing these LIDAR measurements to the real measurements, removing the lower probability particles and keeping the high probabilty ones. Eventually, the majority of the particles will be close to the correct position of the system and thus enabling localization in an existing map [5]. Figures 2.6, 2.7 and 2.8 depict this process.

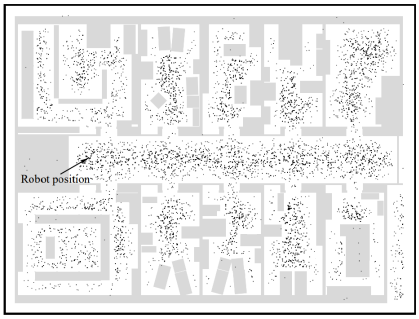

Figure 2.6: First step in Monte Carlo localization [5]

Figure 2.7: Second step in Monte Carlo localization [5]

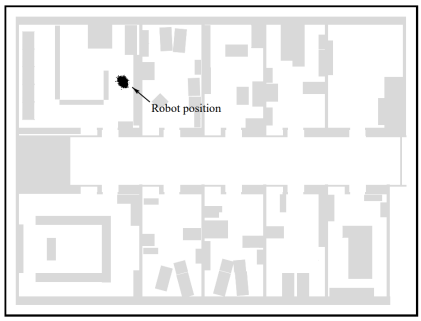

Figure 2.8: Third step in Monte Carlo localization [5]

In figure 2.6, the particles are spread uniformly all over the free space of the map because the system is still uncertain of the correct position. Figure 2.7 depicts the localization algorithm after 1 meter of robot movement. As the robot begins to move, the system can eliminate a majority of the incorrect particles down to two symmetrical positions. Notice how there are two locations with the majority of particles and how those positions are both in a doorway. The system still has some difficulty in determining the correct position as these two possibilities result in similar lidar measurements. After another two meters of movement, the system can pinpoint the correct

position of the robot as shown in figure 2.8 [5] .

2.4.3 GraphSLAM using Pose graph optimization

Pose graph optimization also uses LIDAR and odometry measurements in order to build a map. The technique starts with taking a LIDAR measurement, moving to the next position and making another LIDAR measurement. Each position will be saved as a node. The movement is measured by odometry tools, such as a wheel encoder, in order to virtually estimate the new position of the system relative to the previous position. This distance between nodes is saved as an 'elastic' edge. As previously mentioned, all odometry measurements have error which create an offset between the actual and estimated relative position. This results in an error while constructing a map using LIDAR data. This error accumulates for each relative position measured and interpreted using only odometry measurements. This error can be corrected when the system moves to a previously visited position. This can be detected by overlapping LIDAR measurements. When this is the case, loop closure can be utilized in order to correct previously accumulated errors due to incorrect relative positions of the virtual systems. This compresses or stretches the 'elastic' edges. For each revisited position, the error declines and an increasingly accurate map is created. [22]

2.4.4 ORB-SLAM 3

ORB-SLAM 3 [21] is a recent (2021) milestone in the development of visual SLAM. It builds further on ORB-SLAM [23] and ORB-SLAM visual-inertial [24]. These systems are the first ones to take full advantage of short, mid and long- term data association. Short-term data association means that data collected by the camera in the last few seconds is taken in consideration for calculations [21]. Mid-term data association means that map elements or data that is close to the camera is taken into consideration for calculations. Finally, long-term data association recognizes and matches map elements that have already been visited once before [21]. This makes it possible to detect loops and make loop closures, which allows the system to correct any potential drift or localization error accumulated over the navigation process. It is also possible that the system becomes lost and is not able to map or localize in the current area. If this is the case, the system will start with a new disconnected map until it is able to localize itself in the original map. When this happens, the disconnected map will be merged with original into one map [21].

Using these methods, zero drift can be reached in areas that are mapped. This results in a reusable map where true and accurate localization can occur [21].

The difference between visual SLAM and visual odometry (VO) is that Visual SLAM has the goal of building a map of the environment around the agent while calculating the position of that agent in the map with the help of the on-board sensors, the camera in particular. VO is not focused on building a map, it puts the focus on calculating the motion or movement of a camera relative to the images of the environment [21] .

2.4.5 LSD-SLAM

Large-Scale Direct Monocular SLAM or LSD-SLAM is developped by Jakob Engel, Thomas Schlöps, and Daniel Cremers and attempts to solve the SLAM problem using a monecular camera [6]. This method not only tracks camera motion based on images but also allows to build

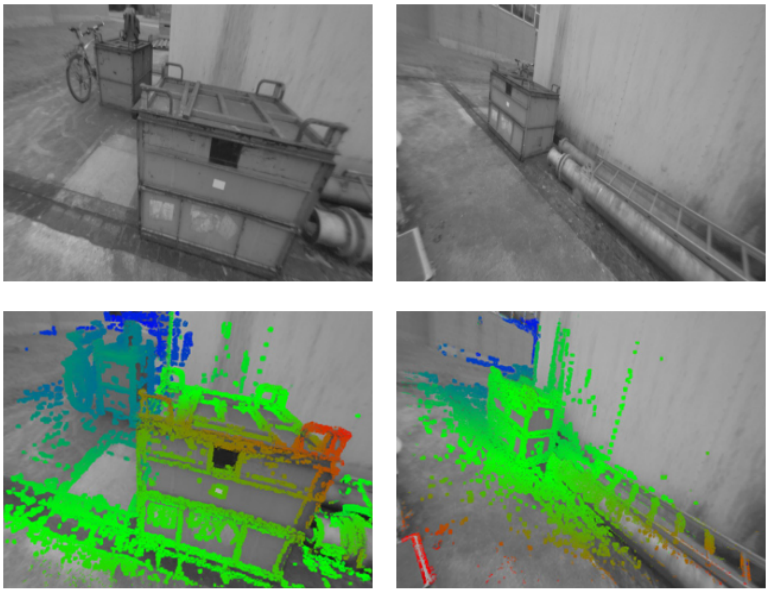

Figure 2.9: LSD-SLAM detecting points in images [6]

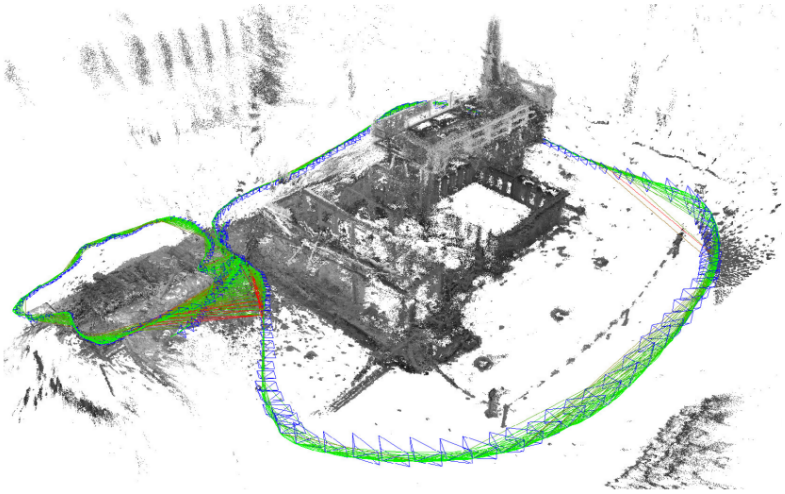

Figure 2.10: Result of LSD-SLAM [6]

consistent and large scale maps of the environment. The map is presented similarly to the GraphSLAM method where the nodes are the keyframes. Visual based SLAM methods usually use one type of feature such as edges or corners. This method analyzes the gradient of the intensity, or brightness, of the camera images in order to localize the camera while simultaneously creating a map of the environment. Using image intensity offers more usable data per image than only extracting one type of feature. The mapping is achieved through a method called direct image alignment [25]. Figure 2.9 and 2.10 show the algorithm in action. Figure 2.9 depicts four images. The top two images are two keyframes. while the bottom images of figure 2.9 have their estimated depth color-coded. After repeating this process while moving the camera around an environment, the environment can be mapped as shown in figure 2.10.

2.4.6 Extended Kalman Filter

The Extended Kalman filter (EKF) is a mathematical approach to solve the SLAM problem using state-estimation. It is also used for fusing sensors as mentioned in section 2.2.3. It presents

the problem as a vector consisting of the position and orientation of the drone as well as each position of the landmarks from the environment. The filter uses a model that describes how the vehicle can transfer between states in order to predict the movement. This model is called the state transition model. For example, the model of a car prohibits it from moving directly left or right. In order for this vehicle to move left or right, it also must move forwards or backwards. The EKF takes the previous estimation of the state and all the sensor data from the observation model as input. It then compares the prediction from the model to the observed state from the previous estimation. It uses these parameters to determine how much the prediction from the model is starting to deviate from the state according to the measurements. When the model is not accurate enough, the filter switches over to the observations and vice-versa if the model becomes accurate again [26, 27].

2.5 Path finding algorithms

In order to navigate from the current position to the goal in a generated map, the path needs to be found between these two positions while avoiding obstacles. The shortest path between these two positions can be found by using a path finding algorithm.

2.5.1 A*

The A* path finding algorithm is an example of a search-based method and starts by turning the map into a graph with nodes and edges. For each node surrounding the starting node, the distance from the starting node to that node is calculated. Then, the shortest possible distance from that node straight to the goal node is added. This distance ignores any obstacles and is added to the current distance to the starting node. The sum of the distance from the starting node to the current node and the straight-line distance from the start to the goal is the absolute minimum path length. The same steps are then repeated for the node(s) with the lowest absolute minimum path length until the goal node is reached. When this happens, it is certain that this is the shortest route to the goal [28].

The downside of this approach is that the cost to compute the shortest path becomes exponentially more expensive the larger the area is [28].

2.5.2 Rapidly-exploring random trees (RRT)

The RRT path finding algorithm is an example of a sample-based graph approach and starts with defining a start node and goal area. Each iteration of the algorithm starts with choosing a random position anywhere on the map, with a preference for areas with few nodes. Along the path from the closest node to that selected position, another node is placed. The new node is placed a predetermined maximum distance from the closest node and are connected with an edge. If this edge crosses a wall or obstacle, no new node is placed. In order to reliably advance the tree to the goal, a position occasionally gets selected inside the goal area. Repeating this process results in a rapidly expanding tree with branches reaching everywhere on the map. When a node is placed within a certain threshold of the goal, a path is found. Using this algorithm usually results in chaotic paths with many turns. The path can be smoothed by using the alternative RRT* approach where new nodes can cause edges to be reconnected if that results in a shorter

path from the start to other nodes. However, this method takes longer to run as opposed to regular RRT [29].

2.5.3 Probabilistic road-map (PRM)

The probabilistic road-map approach is also a graph based approach. This method samples a predetermined amount of nodes randomly across the environment. Then, each node is connected to another predetermined amount of neighbouring nodes that are closest to that node. The shortest path can then be found using a path planning algorithm [30].

2.6 Other algorithms

2.6.1 Teach and repeat

The teach and repeat algorithm is another approach in order to solve the autonomous navigation problem [20]. The principle is that a robot is manually driven once along a chosen path. The robot records its surroundings during this guided drive. These images are then saved together with the corresponding input. These images and inputs are then later used to re-navigate the same path. This algorithm uses a topometric map as explained in section 2.4.1 [20]. There are two possible methods for teach and repeat [20] . The direct visuals method directly compares the full images from the teach with the repeat runs. This is the simplest method but the main problem with this approach is that in order to re-navigate a path, there cannot be any significant changes in the environment. If certain obstacles are moved after the guided drive or the lighting is different, the teach and repeat algorithm will not be a able to recognise the new images and will therefore be unable to navigate. [20].

The other method is the feature based method which extracts recognisable parts of an image like corners, edges and points to create landmarks. These recognisable parts are also referred to as keypoints. These are then used to compare the teach and repeat runs. These landmarks can be used to detect and correct offsets during runs.

2.6.2 AprilTag

AprilTags are a kind of visual fiducial [12]. These fiducials are artificial landmarks designed to be easily and robustly detected by lower resolution cameras [12]. They consist of black and white squares in a pattern, similar to QR-codes and bar codes but not completely the same [12]. An example of an AprilTag is given in figure 2.11. The difference between QR-codes and visual fiducials is that QR-codes need to be captured at a fairly high resolution while a human has to align the camera to the QR-code [12]. These have a payload in the range of hundreds of bytes while fiducials have a much smaller payload at about 12 bytes [12]. The benefit of these fiducials is that they are designed to be automatically detected and localized in pictures or camera feed [12]. Even in situations where detection could be more difficult such as: when the drone is tilted; the area is unevenly illuminated; the tag is partly obscured; the tag is at a further distance or the images have a lower resolution [12].

The orientation and position of AprilTags are not important for detection [12].

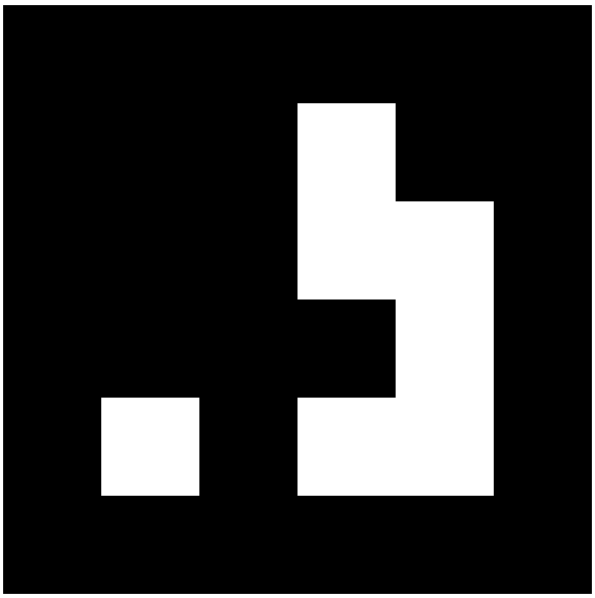

Figure 2.11: An example of an AprilTag [7]

These benefits make AprilTags more suitable for autonomous navigation based on landmarks than QR or bar codes.

Visual fiducial systems are mostly used in augmented reality applications. These tags then serve as positions from where real-world objects are then augmented. These tags can also be attached to objects so the position of said object can be accurately tracked by cameras [12].

Apart from position of the tag, these tags can also be used to depict commands for autonomous systems [12]. This way, a human operator can flash command cards which tell the robot to execute certain commands such as: "move forward", "stop", "follow this" and "pick up this crate" [12]. Alternatively, these tags can be attached to the walls of a building and instruct the robot on how to navigate around the building or close a loop and make the algorithm correct any possible accumulated errors during navigation by giving it a true position on the map [12].

The tags are detected by attempting to localize four-sided regions which are also called "quads" where the interior is darker than the exterior [12]. This is done by calculating the gradient direction and magnitude at each pixel of the image. The gradient in an image of a pixel is a vector which points to the direction where the brightness increases the most [12]. After this step, pixels with similar gradient and magnitudes are clustered together into components. Olson [12] mentions that this approach is sensitive to noise, where even modest amounts of noise can create a false edge. Due to the nature of AprilTags, a low-pass filter can be used as there will be no significant loss of data due to the sharp gradients surrounding the tag itself. Next, line segments are fitted to connected components. After the segmentation step, the task is to find line segments which combine into four sided shapes also known as quads. After the tags are detected, the bits of the payload need to be read by detecting the black and white squares. In order to be robust to lighting conditions, which can vary between tags but also within the same tag, a

varying threshold is used to detected black and white squares. This threshold varies depending on location. [12]

2.7 Conclusion

ORB-SLAM 3 [21] and LSD-SLAM [6] are the most relevant approaches for solving the SLAM problem for this thesis. These methods rely on a camera which is one of the components of the Tello Ryze drone. The Monte Carlo particle filter [5] and GraphSLAM [22] on the other hand show the potential benefit of adding a lidar to the current drone or selecting a drone which comes equipped with one. After the map is built, one of the path finding algorithms can be used to make the drone navigate the environment. Autonomous visual navigation can also be achieved without solving the SLAM problem. Examples of methods that do not need a map to navigate are the Teach and repeat algorithm [10] and AprilTags [7]. These two methods also only require a camera as they mainly rely on images. The AprilTag method requires the least amount of setup in order to test the system. This is why it will be used to test and validate the system.

Chapter 3

Method

3.1 Introduction

Robotic Operating System (ROS) will be used for implementing visual based autonomous navigation. ROS works with nodes which use publishers to send messages to a certain topic and subscribers to read those messages from a certain topic. This way, the code can be presented clearly as a schematic. A schematic of the code for this project is depicted in figure 3.1.

The input node is responsible for receiving user input. This input can be from a keyboard if the corresponding keyboard-input node is active, but can also be a joystick-input node. Depending on the preference, only one node needs to be activated to suit the needs of the user while the other nodes don't need to be activated. In case of multiple active input nodes, input will be received from both nodes at the same time. The messages generated by these nodes are given in table 3.1. Their corresponding descriptions are also given in table 3.1 even though the names are self-explanatory.

Note that there is a message to instruct the drone to take a picture and upload it. In normal circumstances it is not necessary to publish this message as the drone node does this automatically. The purpose of this message is for debugging, when the developer wants to manually decide when the next picture is taken in order to analyze it. The last given movement instruction will

Table 3.1: Generated messages and their corresponding descriptions

Message	Description
start	Drone performs lift-off sequence
quit	Drone performs landing sequence
move_forward	Moves the drone forwards
move_backwards	Moves the drone backwards
move_left	Moves the drone to the left
move_right	Moves the drone to the right
move_up	Makes the drone ascend
move_down	Makes the drone descend
turn_left	Turns the drone counter-clockwise
turn_right	Turns the drone clockwise
picture	Drone will take a picture

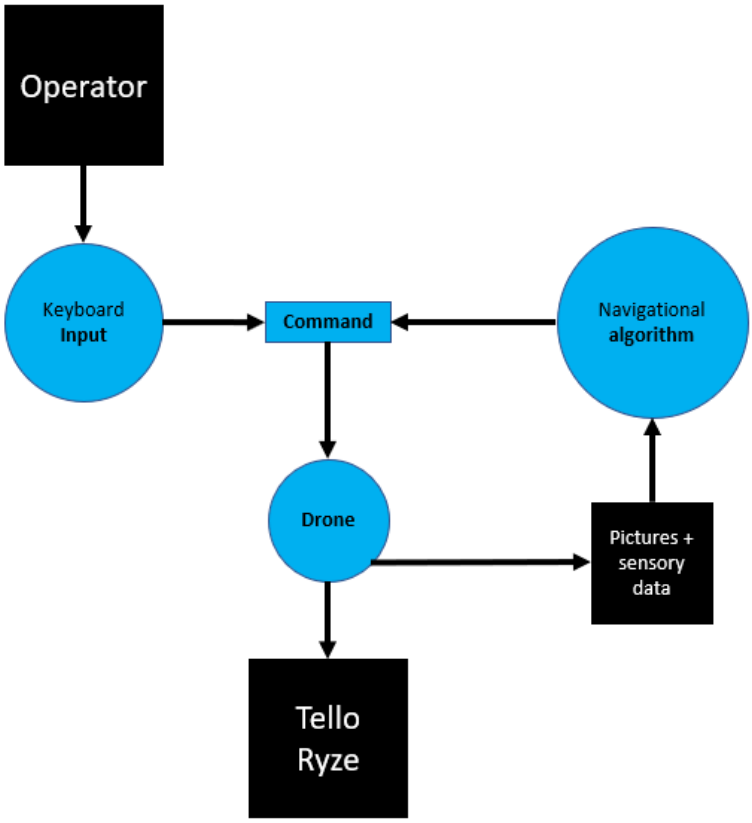

Figure 3.1: A diagram of the software

be executed indefinitely until another instruction is given. In other words, if a move_forward command is sent, it will keep moving forward. In order to make the drone stay still and hover, any other message not depicted in table 3.1 can be given. This also acts as a safeguard against unrecognized messages. These messages act as the commands for the drone and will be sent to the "command" topic. Each input node is a publisher to this node. This topic will also be used by the drone node. The drone node is the node that is responsible for the communication between the laptop and the drone. This node is a subscriber node for the "command" topic and will read or consume each message in the topic. These messages come in as strings and will be converted to the correct corresponding movement of the drone. This node also outputs sensor data such as battery level and camera images at 20Hz. The following sections will explain the purpose of each node and how each node works more in detail.

3.2 Input node

The purpose of the input node is to read keyboard input provided by the operator, convert the input to movement instructions and finally publish these instructions to the command topic. It is written in the C++ language to allow fast execution time and low latency between operator input and the publishing of the command. Like the other nodes, this process can be interacted with in the form of a terminal. This node is unique because the input of the user is immediately processed instead of needing to press enter after each command. The keybindings and their corresponding messages are given in Table 3.2.

Table 3.2: Keyboard inputs and their corresponding messages

Keyboard key	Message
e	start
q	quit
w	move_forward
s	move_backwards
q	move_left
d	move_right
i	move_up
k	move_down
j	turn_left
l	turn_right
space	picture

3.3 Drone node

The drone node is responsible for listening for messages published in the command topic, interpreting the messages as commands and finally, send them to Tello drone so it can perform those instructions. In addition, this node receives images made by the camera of the drone at 20Hz. After receiving an image, the node encodes it and publishes it to the image topic. Communication with the drone is achieved using the djitellopy python library. Whenever a message is published on the command topic, the callback function "command" is called in the node where the data is the command itself as a string format. This function compares the received instruction "data" to each known instruction, such as the movement commands explained in 3.2 or the "picture" command which orders the drone to take picture. Note that the drone already does this without the need of the instruction, this is purely available for debugging purposes. When the message is not one of the recognized instructions the drone will hover. The commands send to the drone are performed by the "send_rc_control" method of the djitellopy library. This method is similar to sending instructions with two joysticks using an RC-controller. It takes in 4 integer numbers, ranging from -100 to 100. The first two numbers correspond to manipulating the roll and pitch respectively of the drone. A negative number for the first input causes the drone to accelerate to the left while a positive number accelerates the drone to the right. A positive number for the second input causes the drone to accelerate forward while a negative number causes a backwards motion arising from a change in pitch. The last two input numbers correspond to manipulating the altitude and yaw of the drone. A positive integer as the third input causes the drone to ascend, while a negative number causes it to descend. A positive number for the fourth and last input causes the drone to turn to the right, while a negative number here results in the drone turning to the left. It is important to mention that the connection with the drone uses the User Datagram Protocol (UDP) connection which does not acknowledge that the "send_rc_control" command has been received, it simply executes the command if it has been received. It is therefor useful to repeatedly send the same message, even though the system keeps executing the command until another command is given.

This node also calculates the time between instructions with the rospy.get_time() method and broadcasts it using the rospy.loginfo() method. This code is purely for debugging and measuring purposes and has no significant impact on the performance of the software.

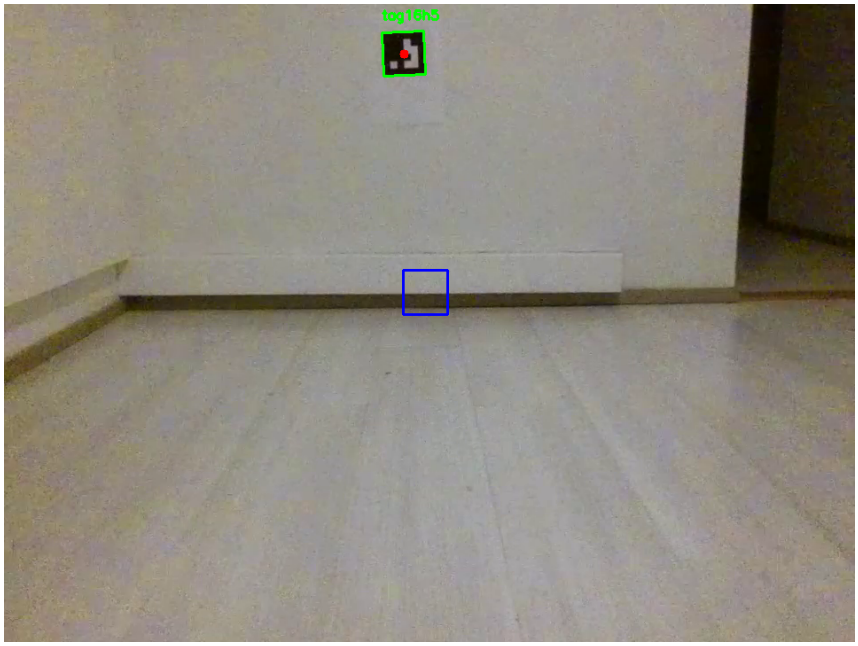

Figure 3.2: Algorithm node vision during tag detection

3.4 AprilTag algorithm node

The algorithm node is responsible for detecting AprilTags from images published from the "image" topic and generating instructions for the drone based on the position and size of the detected AprilTags. The callback method of the node gets called each time an encoded image is published on the image topic. This method first calls the detect method which first decodes the image after that, it detects every AprilTag of the "tag16h5" family. For each detected tag, the coordinates of the four corners and center are used to generate a movement instruction. The difference in coordinates from the four corners is used to obtain the size of the edges of the tag, this is achieved using the distance formula depicted in formula 3.1.

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.1}$$

Where x_1 and y_1 are the coordinates of one corner of the tag while x_2 and y_2 are the coordinates of an adjacent corner. The size of the edges of the tag is used to determine the distance between the tag and the drone. The center of the tag is compared to the static boundaries of the image. In addition to these measurements, the ID of the tag is also read which is necessary to determine at which position the drone is in the track and which instruction is to be performed while perceiving the tag. The vision of the node is depicted in figure 3.2.

The green lines at the top of the image depict the border of the tag. The average size of these lines is used to determine the distance to the tag. The red dot, inside the green square ,depicts the middle of the tag and the blue lines in the middle of the image are the boundaries where the algorithm aims to keep center of the tag. This is achieved by generating appropriate movement instructions. For example, whenever the center of the tag is higher than the upper side of the boundary as depicted by figure 3.2, the instruction generated for the drone will be to ascend. While the center of the tag remains within the boundaries, the tag will be approached by sending the "move forward" instruction. These steps will repeat until the detected size of the edge of the tag becomes larger than the threshold.

3.5 Tag size detection measurements

The drone can detect the presence of a tag and perform instructions based on which tag it detects. Only being able to detect the ID of the tag is not sufficient for navigating. It also needs to be able to determine, or at least estimate, the distance between itself and the target tag. The algorithm decides the drone needs to search for the next tag when the drone has crossed a certain distance threshold and is close enough to the tag. The distance from the drone to the tag cannot be directly measured using only one image of the camera. As a result, the distance to the tag is extracted from the average size of the four edges of the detected tag. These measurements were performed at distances ranging from 30 to 700 cm. The brightness of the environment during the test were 280 and 60 lux. The environment of 280 lux is the result of illuminating a normal house room with a standard light bulb without sunlight. 60 lux is the result of illumination caused by natural evening light.

3.6 Time usage measurement

The time measurements were performed using the loginfo() method from the rospy library. It is important to know the delay between the system observing something with the camera and being able to react to it in order to determine a safe flight velocity. In addition, measuring the delays in different sectors allows developers to identify which sector could benefit the most from optimization.

3.7 Test flights

Test flights were performed over two different types of surfaces. The first surface is given left on figure 3.3 and depicts a standard wooden floor. The second surface is given on the right side of figure 3.3 and depicts a towel with a distinct pattern. Test flights were also performed in two different lighting conditions. The first measurement was in the standard lights on condition, around 280 lux. The second was performed in darker conditions, around 60 lux.

Figure 3.3: Left: A featureless wooden surface. Right: A surface with more features

Chapter 4

Results and conclusion

4.1 Results

4.1.1 Distance measurements

The first measurement is how the size of the tag correlates to the distance from the camera to the tag. The graph is given in figure 4.1. The distance from tag to camera is measured in centimeters while the size of the tag uses image coordinates. The measurements start at 30 cm where the average size of the sides is 18 (image coordinates). From 30 cm to 50 cm, measurements are taken at an increment of 5 cm. After 50 cm, the increment is increased to 50 cm per measurement. Until 250 cm there is a significant decline in size. After that point it stagnates until the end. According to the graph presented in figure 4.1, determining the distance between drone and tag becomes increasingly more unreliable the further away the drone is from the tag with this method.

4.1.2 Time measurements

The second measurement is the time necessary for the system to take a picture, detect tags, generate an instruction and execute that instruction. The pie chart depicting the time usage is given in figure 4.2. The total time for this process is 80,49 ms, the average of 120 time measurements. This time can be divided into 3 main parts. The first part is the time needed to take a picture, encode that picture and publish it to the "image" topic. This part takes on average 38,79 ms and is 48% of the total time. The second part is the time it takes to transfer the image, decode it, detect the tags and generate an appropriate command. This part takes 39,33 ms and is 49% of the total time. The last part is the executing of the movement command and takes 2,36 ms and is 3% of the total time.

4.1.3 Test flights

After performing the test flights, it was determined that this system has 2 limitations. In test flights above the wooden surface depicted left on figure 3.3, it was significantly harder to control the drone manually as well as autonomously as opposed to test flights above the second surface depicted right on figure 3.3. The drone has significantly more drift over the first surface. This is mainly caused by the visual positioning system, which relies on having distinguishable features

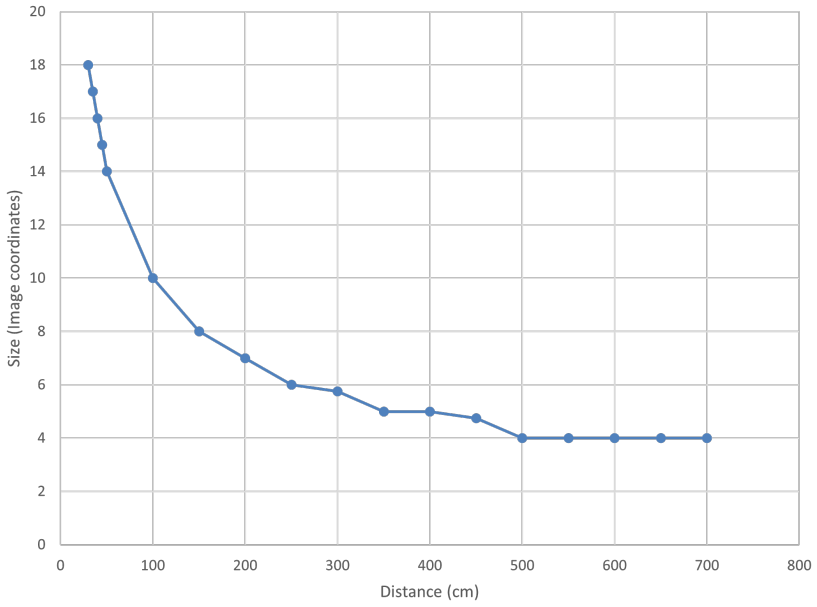

Figure 4.1: Tag size in image relative to distance from tag to camera

over the surface that it covers. When this is not the case, the drone has trouble hovering and can collide with obstacles. This means that floors such as depicted left in figure 3.3 are less suitable for navigation. In addition, extremely dark or bright rooms have the same effect. But in this case AprilTags also would not be able to be detected. The other limitation is the accuracy-speed trade-off. Setting stricter boundaries forces the drone to make corrections more often, hindering it to move in the desired direction but increasing the accuracy. Stricter boundaries allows the drone to navigate narrow spaces, while broader boundaries can be used for spacious environments. Whenever the drone is sufficiently close to an AprilTag, it starts turning clockwise or counter-clockwise until the next AprilTag in the sequence is detected. It then continues to rotate until the center of the tag, at least, has crossed middle of the boundaries. From that point on, it corrects itself by moving left or right. The problem with this approach is noticeable whenever the AprilTag is at further distances. The rotation sometimes overshoots by a small amount, this causes the drone to compensate this by moving left or right which sometimes results in the drone colliding with an obstacle.

4.2 Conclusion

This paper presents a relatively simple to understand system that enables a Tello Ryze drone to navigate autonomously in an indoor and GPS-less environment based on Apriltags. This system is also easily expendable with new navigational algorithms. There are however a few requirements and limitations:

- The environment must be illuminated sufficiently
- The surface below the drone must have enough distinguishable features
- areas with draft wind should be avoided
- atmospheric obscurants such as smoke, fog and dust can hinder the drone's navigational capabilities

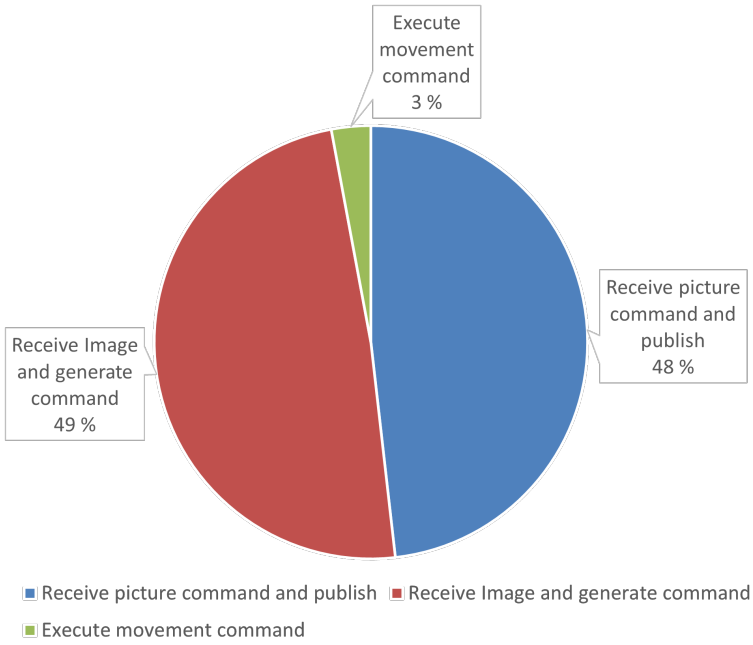

Figure 4.2: Pie chart depicting time usage during algorithm

The usage of only Apriltags is rather limiting on the potential of autonomous systems but are sufficient. They should be used together with other visual based SLAM algorithms in order to add another way to localize the UAV and map an environment. The system allows this as new algorithm nodes can be added to the existing structure. Images generated by the drone can be accessed in the ”image” topic which allows the integration of visual-based SLAM algorithms. As long as the commands generated by the new node are recognized as described in the method, the system can execute these commands. In order for this to work together with the AprilTag node, a new node or system which handles priority needs to be added.

Bibliography

[1] M.-H. Chae, S.-O. Park, S.-H. Choi, and C.-T. Choi, "Commercial fixed-wing drone redirection system using gnss deception," *IEEE transactions on aerospace and electronic systems*, pp. 1–15, 2023.

[2] *Autonomous flying robots: unmanned aerial vehicles and micro aerial vehicles.* Tokyo: Springer, 2010.

[3] S. Juan, S. Etigowni, S. Hossain-McKenzie, M. Kazerooni, K.-I. Davis, and S. Zonouz, "Crystal (ball): I look at physics and predict control flow! just-ahead-of-time controller recovery," 12 2018.

[4] H. Yang, Y. Lee, S.-Y. Jeon, and D. Lee, "Multi-rotor drone tutorial: systems, mechanics, control and state estimation," *Intelligent service robotics*, vol. 10, no. 2, pp. 79–93, 2017.

[5] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," pp. 343–349, 01 1999.

[6] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *Computer Vision ECCV 2014*, vol. 8690 of *Lecture Notes in Computer Science*, (Cham), pp. 834–849, Springer International Publishing, 2014.

[7] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, IEEE, 2016.

[8] W. Hulens Dries, QVan Ranst and Q. T. Cao, Ying, "Autonomous visual navigation for a flower pollination drone," 2022.

[9] D. C. Schedl, I. Kurmi, and O. Bimber, "An autonomous drone for search and rescue in forests using airborne optical sectioning," *Science Robotics*, vol. 6, no. 55, p. eabg1188, 2021.

[10] D. Dall'Osto, T. Fischer, and M. Milford, "Fast and robust bio-inspired teach and repeat navigation," *arXiv.org*, 2021.

[11] R. Tech, *Tello User Manual v1.4*. Ryze Tech. Available at `https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello\%20User\%20Manual\%20v1.4.pdf` [Accessed: 20 May 2023].

[12] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3400–3407, IEEE, 2011.

[13] R. F. Stengel, *Flight Dynamics*. Princeton, NJ: Princeton University Press,, 2015.

[14] N. Hall, "Aircraft rotations - glenn research center," Jul 2022.

[15] W. Chen, Z. Wang, D. Luo, B. Zhu, and C. Peng, "Onboard sensing for drone to fly through a gate with a rotating arm," in *Lecture Notes in Electrical Engineering*, vol. 644, pp. 1599–1608, 2022.

[16] P. Castillo-Garcia, *Indoor navigation strategies for aerial autonomous systems*. Amsterdam, [Netherlands: Butterworth-Heinemann, first edition. ed., 2017 - 2017.

[17] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[18] Y. Yasuda, L. Martins, and F. Cappabianco, "Autonomous visual navigation for mobile robots: A systematic literature review," *ACM computing surveys*, vol. 53, no. 1, pp. 1–34, 2020.

[19] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 2016.

[20] D. Dall'Osto, T. Fischer, and M. Milford, "Fast and robust bio-inspired teach and repeat navigation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 500–507, 2021.

[21] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[22] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[23] R. Mur-Artal, J. Montiel, and J. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147 – 1163, 10 2015.

[24] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[25] M. Pizzoli, C. Forster, and D. Scaramuzza, "Remode: Probabilistic, monocular dense reconstruction in real time," 05 2014.

[26] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

[27] T. Borangiu, "Extended kalman filter (ekf)-based local slam in dynamic environments: A framework," in *Advances in Intelligent Systems and Computing*, vol. 371 of *Advances in Intelligent Systems and Computing*, pp. 459–469, Switzerland: Springer International Publishing AG, 2015.

[28] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[29] J. Lee, O. Kwon, L. Zhang, and S.-E. Yoon, "A selective retraction-based rrt planner for various environments," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 1002–1011, 2014.

[30] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge, England: Cambridge University Press, May 2017.