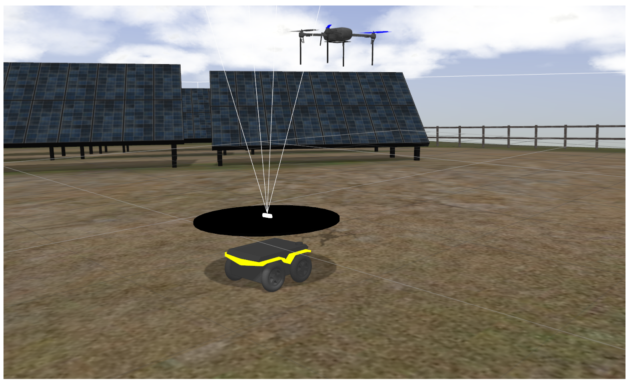


Cooperative vision-based Automatic Landing System for a Micro Aerial Vehicle

Martim Gard Brito de Vasconcelos Braga

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. Meysam Basiri

Examination Committee

December 2021

Declaration

Acknowledgments

I would like to thank my family for their friendship, encouragement and caring over all these years, without whom this project would not have been possible.

I would also like to acknowledge my dissertation supervisors Prof. Meysam Basiri and ISR staff for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends that supported me and were relentless in helping and motivating me to finish my studies.

To each and every one of you

Thank you.

Abstract

Nowadays the use of Unmanned Aerial Vehicles (UAV) is more popular than ever from hobby to military applications. The ability of stationary hover flight makes multi-rotor UAVs particularly interesting in inspection missions. However given the small flight time that common drones have, an autonomous landing system could mitigate the issue of constant need of human intervention in the landing and battery charging/replacing stages. The goal of this thesis is to control a quad-rotor UAV, from free motion to autonomous landing on a mobile charging station. The landing platform is placed on an Unmanned Ground Vehicles (UGV) that acts as the mobile charging station and processing unit of the UAV. The controller design seeks to achieve full autonomy in the landing process based on off-board sensors. To best guide the drone to the landing pad, a multi-stage controller was developed. With a system like this the robustness and accuracy in the landing process can be improved, allowing the UAV to use the battery of the UGV as a mobile battery range extender for longer inspection missions. As the UGV usually has a higher payload limit, it can have large batteries, more capable sensors and higher computational power. This work is specially relevant given that can be applied to the inspection of a solar farm, where a team of ground/aerial robots could reduce both the time of the inspection and the need for human intervention. The developed system was tested in a simulation environment and in 50 consecutive landing cycles achieved an average time of landing of 50 s, with a precision of 2 cm from the center of the landing pad and 100% success rate.

Keywords

Quadcopter; Automatic landing; ROS programming; ArduPilot, DroneKit-Python, Computer vision.

Resumo

Nos dias que correm o uso de Veículo Aéreo Não Tripulado (VANT) é cada vez mais popular, desde aplicações militares a uso recriativo. Visto que drones pequenos normalmente têm um tempo de voo reduzido, um sistema de aterragem automática pode mitigar o problema da necessidade constante de intervenção do usuário em aterrar e carregar e/ou trocar a bateria. O objectivo desta dissertação é de controlar um quadricóptero, de voo livre até aterrar numa plataforma de carregamento móvel. A plataforma foi colocada num Veículos Terrestres Não-Tripulados (VTNT) que fez o papel da plataforma de carregamento móvel e unidade de processamento do drone. A concepção do controlador visa autonomia total no processo de aterragem baseado em sensores exteriores ao drone. De modo a guiar o drone até à base um controlador multiestágio foi desenvolvido. Com um sistema como este a robustez e precisão no processo de aterragem podem melhorar, deste modo permitindo ao drone usar o robot terrestre com base móvel de extenção de bateria na realização de tarefas de inspecção mais longas. Como o robot terrestre não tem tanto limite de carga, pode ter baterias grande, sensores mais capazes e maior poder de processamento. Este trabalho tem especial relevância dado que pode ser aplicado na inspecção de parques solares, onde uma equipa de robots aério/terrestre pode reduzir tanto o tempo da inspecção como a necessidade de intervenção humana. O sistema desenvolvido foi testado em simulação e, em 50 ciclos de aterragens consecutivas, alcançou um tempo médio de aterragem de 50 s, com precisão de 2cm do centro da plataforma de aterragem e 100% de taxa de sucesso.

Palavras Chave

VANT; Aterragem automática; Programação ROS; ArduPilot; DroneKit-Python; Visão computacional;

Contents

List of Figures

Acronyms

API	Application Programming Interface
AR	Augmented Reality
CARMEN	Carnegie Mellon Navigation
FOV	Field of View
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IP	Internet Protocol
MAV	Micro Aerial Vehicle
MOCAP	Motion Capture System
MRDS	Microsoft Robotics Developer Studio
MSL	Mean Sea-Level
OS	Operating System
PID	Proportional-Integral-Derivative
RC	Radio Control
ROS	Robot Operating System
SITL	Software in the Loop
UAV	Unmanned Aerial Vehicles
UGV	Unmanned Ground Vehicles
VANT	Veículo Aéreo Não Tripulado
VTNT	Veículos Terrestres Não-Tripulados

1 Introduction

Contents

1.1 Motivation

Unmanned Aerial Vehicles (UAV) are becoming ever more popular in applications where an aerial point of view is an advantage, such as in military patrol, coastal and agriculture surveillance, infrastructures inspection, etc. With recent developments in technology, this type of vehicles has never been so accessible to the general public. Photography enthusiasts, for example, can get shots that in the past required far more expensive means to achieve the same goals, as helicopters or airplanes.

With such a broad array of applications the degree of complexity can drastically differ. On one end of the spectrum, there are consumer level UAVs that can be simply composed by an Inertial Measurement Unit (IMU), Global Positioning System (GPS) sensors and a camera, while on the other end there is highly specialized hardware to suit each desired application.

An UAV can be powered in two different ways: tethered, where it is connected via cable in order to get power; or by onboard battery supplies. The main disadvantage of the battery powered UAV is the UAV's range. Although not so relevant in large UAVs, the smaller ones usually have a weight constraint that implies the use of small capacity and very light batteries that result in a small range. For each desired application the best battery size/weight can be calculated [12] [13], in order to get the optimal desired range and endurance.

Developing an autonomous flying and landing system can largely improve productivity and flexibility to perform the desired task, minimizing or completely removing human intervention. There are many challenges that have to be overcome in order to implement an autonomous system able to complete the task mentioned above. Concerning the limited flight time of a small UAV, a possible solution is to develop an automatic landing system to recharge on a charging station. This would enable it to do the designated task without the need of the technical support to replace and charge the batteries. In order to implement such a system robustly, one of the biggest challenges is localization. For an UAV to land on a small platform, a high accuracy in positioning is required, given the small size of the landing platform. Despite the GPS being sufficient for navigation in wide open spaces, its accuracy is insufficient to solve the UAV landing problem. There are several different approaches that might include computer vision or radio signaling to aid with the approach to the landing platform.

To further enhance the capabilities and versatility of an UAV, working cooperatively with Unmanned Ground Vehicles (UGV) can reduce the time to complete the task, or make it more efficient. On the other hand, the UAV can aid the UGV in adverse terrain, where the UGV's on board camera does not provide the operator with enough helpful information. The aerial point of view of the drone can help plan an easier route for it to travel [14]. For example, in solar panel field inspections, the cooperative work between the two vehicles further optimises the inspection process by using an UGV to work as a mobile charging station. This latter application of UAV and UGV cooperation allows the team, in a field inspection mission, to complete its mission faster [15]. It also increases the inspection range of the UAV given that it is no longer dependent on a fixed charging station.

1.2 Automatic landing

The first demonstration of an automatic landing system occurred on August 23, 1937 by two American Army pilots. The system used was a C-14B plane with added instruments and radios. There were five beacons on the surroundings of the landing field. With this primitive system the first automatic landing was achieved [16]. After this in June 1965, research and development was made and implemented on commercial flights, with the aim of helping the pilots to land in adverse weather, or reduced visibility. This way the aircraft can land safely in conditions that otherwise the pilot might not be able to perform.

In this context, an automatic landing system, similar to the aviation sense of the word, is one that can successfully land the aerial robot without human intervention. The main challenge associated with the development of such a system capable of landing the UAV on a small platform is getting a precise location of the UAV relative to the platform. Indoors there are systems that are able to precisely track every movement based on visual cues (eg. Motion Capture System (MOCAP)). But outdoors, such a system is not viable, so other positioning aid techniques must be used. Computer vision plays an important role in many applications to overcome this problem.

Many different solutions were presented using visual aid to successfully land a UAV. Some use visual markers to detect the base and were able to land onto a moving target [17]. Or commercial solutions such as FlytBase's FlytDock software that allows to autonomously land any drone over a visual target [18].

1.3 UAV

UAV's date back to the nineteenth century when the Austrians attacked the city of Venice in Italy, using unmanned balloons that were loaded with explosives. Even though these balloons would not be considered UAVs in the modern sense of the word, they eventually led to other advances in the field. Until very recently this technology was reserved for military use only. This changed with miniaturization and drop in the cost of construction, which made this technology become ever more accessible to the public. There are numerous types and configurations of UAVs, varying in size and complexity as shown in figure 1.1 below.

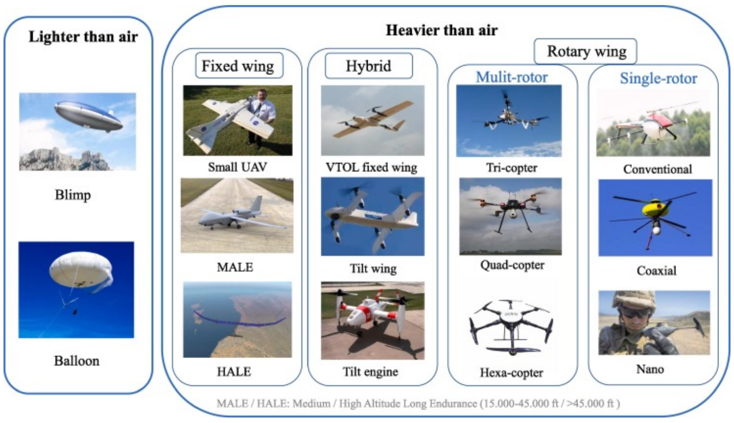


Figure 1.1: Different types of UAV. [1].

UAVs can be classified by operating altitude, endurance, operating range, payload and size [19]. In this dissertation we will focus on the group of multi-rotor UAV, more specifically a quad-rotor. Further advancements in technology have brought smaller and more power efficient electronics, which have allowed the creation of the micro UAV. These are very popular both amongst the military and enthusiasts, given it's agility and versatility being particularly useful in inspection, surveillance, mapping and search and rescue missions. Among these categories, the micro UAVs quad-rotor drones as seen in figure 1.2 are the most popular type.

Figure 1.2: Model of a quadcopter. [2].

Such UAV's usually have an autonomy constraint due to the fact that batteries, the most commonly used method to store electric energy in this aplication, are heavy. So the batteries have to be small in order to be light, which results in a flight time of only a few minutes even with a fully charged battery.

By increasing the number of on-board sensors, the total weight increases, which reduces the UAV's autonomy even further. So in order to operate for longer periods of time, the UAV needs to be charged very often. In a non-automated system the user has to land the drone, and change or charge the battery manually. By automating the landing process, the drone can land on a charging platform. This way it may be able to perform tasks repeatedly without continuous human intervention. The reduction in the amount of payload on the UAV achieved by reducing the number of on-board sensors, can further improve the flight time.

1.4 Objective and Contribution

This master thesis' objective is to implement a viable solution for an automatic landing system of a micro-quadrotor UAV. There are several different ways to implement an auto-landing system, either with on-board sensors or with sensors located near the landing platform. In this thesis the aim is to implement a system that minimizes the need for on-board sensors, so that the payload on the UAV is decreased and thus the flying time is maximized.

To achieve such a system a multi-stage control was developed. The automatic landing system was developed using vision sensors positioned on a UGV to reduce position uncertainty of the UAV itself and help guide it to the charging platform located on top of the ground robot.

The goal of this system is to improve the robustness and accuracy of the landing process, allowing the UAV to use the battery of the UGV as a mobile battery range extender for longer inspection missions. As the UGV does not have a payload limit, it can have large batteries, more capable sensors and higher computational power.

The UAV used in this study is a micro Quad-rotor which, due to its small size, has a very limited range. The goal is, with the aid of the ground robot, to enable the UAV to perform repeatedly more extensive tasks that otherwise would require user intervention to charge or change the UAV's batteries. To achieve this, the work was divided into 3 main goals:

1. **Off-board vision-based system** to detect and accurately estimate the UAV's position relative to the charging platform.

2. **High-level motion control system** to precisely control and guide the aerial robot in space, and successfully land it on the platform.

3. Develop a **simple solution** that is easily used on a wide range of quadcopters and transferable to real hardware.

1.5 Thesis Outline

This thesis is structured in five chapters.

The first chapter is intended to expose the motivation behind the dissertation and also presents the basic concepts that are involved in the project.

The second chapter briefly describes the previous work done in the area of study.

The third chapter describes in detail the solution found for the automatic landing system and its implementation.

The fourth chapter presents the thesis results and experiments.

The fifth chapter presents the work conclusions and further potential improvements to the system.

2 State of the Art

Contents

2.1 Introduction

In order to develop a system capable of achieving the set goal, an understanding of each subsystem is required. For this reason, it is important to identify all topics of interest to further investigate.

There are two main approaches to solve the localization problem in the UAV landing procedure. One possible solution is to have the main sensors to aid with the landing on the drone itself and the other one is to have them close to the landing area.

2.2 On-Board Sensors Landing Systems

A common way to solve the problem of localization of the drone relative to the landing platform is through computer vision. Compared with typical sensors, a camera is lighter and can provide rich information about the UAV self-motion and external environment.

Indoor landing provides a controlled environment when compared with landing outdoors, as it has much less external disturbances. A low cost approach was developed in [20], where by using a Nintendo Wii remote infrared camera, the drone can detect points on the platform. These points are Infrared LED's, that are positioned in a pattern so that the flight controller can estimate the position and orientation of the UAV once the infrared LED pattern is identified.

In [3], two different computer vision algorithms were used in parallel in order to achieve a more robust landing system. Firstly a red blob tracking was implemented to get an estimate of the position of the UAV relative to the platform. In parallel to this, a corner tracking algorithm was implemented in order to get a robust motion tracking for when the target platform is not inside the Field of View (FOV) of the camera. In order to locate the landing platform, a visual cue can be placed on the landing platform, usually a red circle or blob. This is a simple technique that helps with the positioning of the UAV relative to the landing platform.

A possible algorithm is proposed and presented in figure 2.1. The UAV flies using GPS navigation and, when it gets close to the landing platform, begins to capture continuous images in order to search for the visual cue placed on the platform, in this case a red blob. If the platform is located, the tracking procedure is carried out when the cue is in the FOV of the camera. When the position of the platform relative to the UAV is estimated, the drone starts its descent until it reaches an altitude of less than 20cm from the platform. At this point it starts the landing procedure.

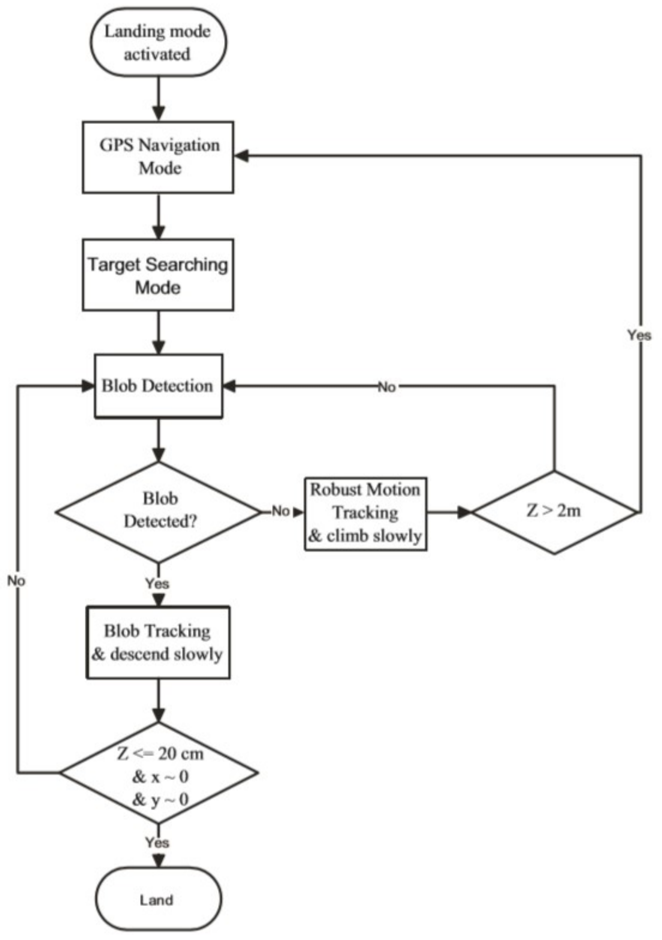


Figure 2.1: Proposed design for an automatic landing system [3].

This is a simple technique that helps with the positioning of the UAV relative to the landing platform.

The landing system needs to be robust. If not, any external disturbance that shifts the UAV unexpectedly, causing the platform to move out of the FOV of the camera, would result in the UAV getting lost.

So a more robust tracking system is required. A solution that was found to prevent the UAV from missing the target was to extract more features out of the image, such as corners. This optical flow, once computed, allows the horizontal speed to be estimated. This, in turn, allows the UAV to hover and ascend slowly until it detects the target. The robustness of this system is based on collecting a range of features from the whole image around the platform and not only on the blob itself.

Another approach was used by [21], where a pattern was designed to overcome the limitations of other models where the target is only visible close to the landing pad. A set of concentric white rings was drawn on a black background. As each ring is analysed individually the pattern can be identified even if the whole platform if not inside the FOV. Although the pose estimation is very good using this technique, no successful landing was performed.

In [22] infrared cameras were used to detect the landing platform. This method relies on the difference in temperature between the target and its surroundings. To achieve this, the target object was covered with a black powder to increase the efficiency of thermal radiation. This proved to be a very fast and reliable way to identify the target but no landing or pose estimation was carried out.

Using a model helicopter, successful landings were performed [23]. This was achieved by locating a H-shaped maker on a helipad using an on-board camera. The results were robust and repeatable, but the algorithm relies on the assumption that the marker used has a well defined geometric shape and the camera is perpendicular to the ground.

2.3 On-Ground Sensors Landing Systems

Compared with the on-board sensors approach, this technique allows for the use of high resolution cameras alongside high computer processing power. In [24] a triple camera system was used and, by extracting key features of the images, a robust 3D position and orientation were obtained. A stereo camera system was implemented in [25] and [26]. This technique relies not only on an RGB camera but also on a 3D point cloud.

An infra-red stereo camera was also used in [27], in this application the UAV's could be successfully tracked and positioned. However, in a noisy background such as a cloudy/overcast sky or hot objects showing in the image frame, the system proved not to be very reliable. In [4], a set of radio signal emitters on a charging platform as shown in figure 2.2 has been used. This increased the pose estimation precision of the UAV's to about 1 cm. As both the location system and the charging platform emit radio signals, interference between the two has to be taken into account.

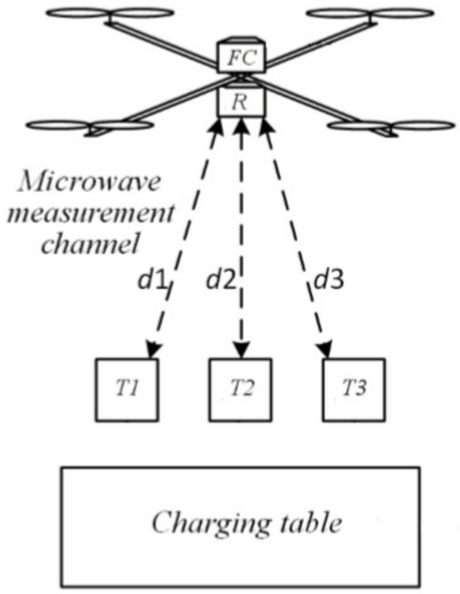


Figure 2.2: Scheme of radio emitters network to locate the UAV when landing [4].

2.4 Flight Controllers

Proportional-Integral-Derivative (PID) controllers are possibly the most commonly used for a feedback linear control system. A Proportional-Derivative Controller design was presented by Erginer and Altug [28]. The developed controller was designed to control the UAV's altitude. To do this it controls the pitch, roll and yaw angles. Alongside this, a visual-based tracking system was successfully developed, allowing the UAV to approach the target and hover above it at the desired altitude. This type of linear controller can be a good option when the system's dynamics are not known; otherwise, it can lead to sub-optimal performance.

In feedback linearization control technique, a nonlinear feedback is added to a nonlinear system so that, when designing the controller, an approximated linear system can be considered. To develop a flight and landing controller, a feedback linearization controller was developed to autonomously land a quadrotor UAV onto a moving platform [29].

Backstepping nonlinear controls can also be used when designing a landing controller. This type of controller offers a recursive way of design that divides the system into subsystems. This allows the designer to start developing the controller at a known and stable stage of the system and "back-out" new controllers that will progressively stabilize each of the outer systems. Such a system was used to land a rotary UAV using a tether in [30] and [31].

Neural networks can be incorporated into a controller design. A neural network has the ability of learning, so that given a set of observations and a class of functions, the network can learn what function will solve a given problem in an optimal way. An intelligent auto-landing controller was designed to enhance the landing safety in different wind conditions [32]. This makes for a much more reliable controller, given that it has the ability to adapt the system's parameters to new situations that it was not trained for to begin with.

2.4.1 PID Controller

As previously stated, PID controllers are one of the most common forms of feedback controllers. With this type of controller, having a control variable, an actuator to allow the system to act on the control variable, and a form to read the state of the system, an accurate and responsive correction is made in order to stabilize the system in the desired set point. The output $u(t)$ of a PID controller can be expressed by the following equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)\,dt + K_d \frac{de(t)}{dt} \tag{2.1}$$

Where $u(t)$ is the output of the controller at the time t, $e(t)$ is the value of the error at that same instance, and K_p, K_i and K_d the gains for each part of the controller. In this application there are 3 variables to be controlled, X, Y and Z, so there is a PID controller for each of these control variables.

As can be seen in equation 2.1, the proportional part of the controller has a gain of K_p and, as the name would suggest, it has an effect on the control signal proportional to the system error, this being the difference between the current state and the set point.

K_i is the gain for the integral component of the controller. This gain applies an effort proportional to the integral of the error, which in turn removes the steady state errors which must be identified over time. Therefore it cannot be eliminated or identified in the instantaneous readings of the error or its derivative.

The derivative part has a gain of K_d, which acts like a damper on the control effort, proportional to the derivative of the error, slowing the state's approach to the set point. In order to get a stable system, the gain values must be adjusted to suit the controlled system. Besides the gains, there are other further parameters that need tuning. These are the minimum and maximum control effort, to ensure that the output does not surpass the physical limit of the actuator, and the integral windup limit, which sets the maximum effect that the integral part may have on the control effort. The integral windup limit is important due to the fact that this component can easily saturate and destabilise the system's response.

2.5 Robot Development Tools

In order to develop the proposed system it is necessary to choose a platform or framework capable of integrating the different types of hardware and software solutions. It is important that this platform aids in the development of the solution as well, by running already developed packages and libraries. This way, the result solution can be turned into a library and help the community to add further developments using this library as a stepping stone.

Carnegie Mellon Navigation (CARMEN) [33] is an open source software designed with ease-of-use, extensibility, and robustness in mind. It was developed by Carnegie Mellon University as a modular solution aiming at easing robot programming. By being a modular solution, it enables the easy installation and removal of add-on software. CARMEN is written in C programming language and supports C/C++ and Java as well. CARMEN was developed with the goal of solving robot navigation and mapping , so there are no packages for computer vision. Also, the latest software update was released in 2008 [34] and so it is no longer being supported. We may then conclude that CARMEN is not the best platform to develop the desired system.

Another existing system that can be used is OpenRDK [35]. OpenRDK supports C and C++ programming languages only. This platform introduced the ability to use a multi-threaded and multi-process structure. With this, it allows the developer to use multiple robots, computer vision and various hardware simulators. It was successfully used in various RoboCup competions, but it seems to no longer be supported or developed as the latest release was in 2010 [36]. Although being more capable than CARMEN, OpenRDK should not be considered for the development of this solution as it is also no longer being supported.

Microsoft developed Microsoft Robotics Developer Studio (MRDS) [37] as an attempt to simplify robotics development and draw programmer's interest to their Operating System (OS) (Windows). Being a system that is compatible with Windows only, it limits the OS choice and consequently the hardware that can be used. MRDS is mainly compatible with .NET, C#, C++ and IronPython programming languages. This framework allows for 3D simulations as well as supporting various packages from speech recognition to simpler tasks such as navigation. MRDS was last updated in 2012 and it only has support up to Windows 8. [38] As this framework is no longer maintained and due to .NET dependencies it lacks support for a variety of hardware, so it is not a good option to be used as a base to develop this dissertation.

Robot Operating System (ROS) was created and developed by Standford University and further developed later by Willow Garage in order to create an open-ended collaboration framework that would be easy to use and develop on. ROS is mainly written in C++ and in Python but users can choose to develop it also in Octave, Lisp and Java. Nowadays ROS is currently the most used platform in robotics.

By being open source, hardware support is ever growing due to the ease of adding support for new hardware or adapt existing drivers to the desired new hardware. ROS is compatible with Gazebo physics simulator allowing to simulate various systems, like arm manipulators, UAV's, cameras, and mobile robots, etc. In addition, ROS already has many packages for navigation, computer vision, mapping and control systems. Same as the hardware support, these packages are constantly being added by the community.

The community also maintains a wiki page about the available packages and also tutorials on various subjects. ROS has its annual convention and is actively supported with annual distribution releases. The latest release was in May 2020. All things considered, ROS seems to be the best platform to develop the desired system on, since it has the best hardware/software support.

2.6 Existing Related Implementations

An automatic landing and charging system was developed in [39]. This system consists of an UAV with an on-board camera. The drone starts by navigating using GPS to guide it to close to the charging pad location. When this is accomplished, it switches to visual positioning method, using a ArUco tag located on top of the charging platform. By developing a controller that uses this relative positioning method to guide the drone to the base and successfully land it was accomplished. This allowed the aircraft to recharge using wireless or wired methods.

Compared with on-board systems, off-board systems have the advantage of being able to use high-quality image systems paired with high computational resources. A ground-based approach was developed in [40] where a stereo camera was used to estimate a 3D position of a quad-rotor and with this achieved an autonomous hovering and automatic landing system. The developed system was capable of locating the Micro Aerial Vehicle (MAV) while flying at 6 m altitude, increasing the accuracy of both longitudinal and lateral positions to 1 m, compared with 3 m of GPS positioning in the same conditions.

3

The Implementation

Contents

3.1 Concept

Before diving deeper into the details of the implementation, we should first review the goals of the project and highlight all of its different parts. The aim of this thesis is to develop various methods to achieve a system capable of reliably land a quad-rotor drone on a stationary mobile platform using external visual servoing. A flight mission can be described in 5 main steps as shown in figure 3.1:

1. **Recharge** on the platform to start the mission cycle;
2. The **takeoff** where the UAV begins the flight;
3. The **mission execution** where it executes the desired tasks;
4. The **detection** of the flying UAV;
5. **Landing** where it finishes this cycle.

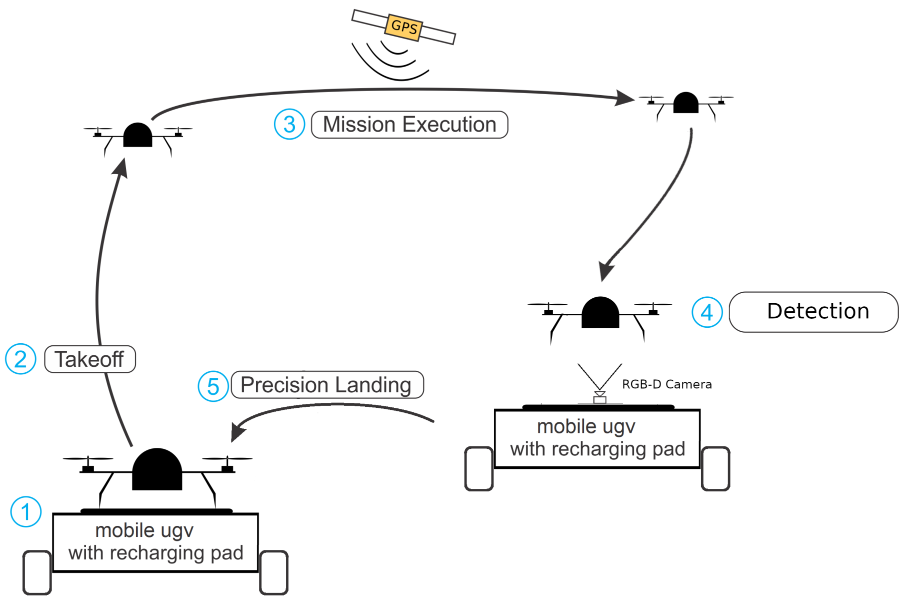


Figure 3.1: Flight mission scheme.

This thesis will focus on the detection and precision landing parts of the mission, that includes precisely detect, guide to the landing location and land the aircraft.

The major problem is to precisely locate the UAV with the most part of the sensors located near the platform. As previously stated, GPS accuracy is not good enough to guide the drone to the base, so in order to accomplish a robust landing system other methods had to be developed.

The system starts after the MAV completes its mission and needs to recharge. At this stage the drone will be flying using GPS, so in the first stage the ground station will receive a signal to start the retrieval procedure of the aircraft to recharge its battery. The GPS coordinates of the base are sent to the UAV and handled by its autopilot system. The autopilot flies it to the desired coordinates.

In order to reduce pose uncertainty a visual marker was added to the underside of the drone. When it arrives to the given location it switches to the vision based controller. The vision based controller will receive the relative position of the UAV and feed it to a PID controller to guide it closer to the platform. This relative position is the estimated position of the marker or, in case of the marker not being detected, the position calculated by the depth camera is used in its place.

3.2 Hardware

For the choice of hardware it is important to choose options that are compatible with ROS, in order to simulate and develop the system. With this in mind a Clearpath robotics Jackal UGV, such as the one in figure 3.2, was chosen to be used as the mobile landing station, as the manufacturer has a very complete package of the robot for simulation and real world use [41]. For the simulations carried out in this thesis, a base Jackal equipped with a GPS sensor, a 1.0 m diameter round landing platform and a RGB-D camera located at the centre of the platform was used, as shown in figure 3.3.

Figure 3.2: Clearpath robotics UGV Jackal [5]

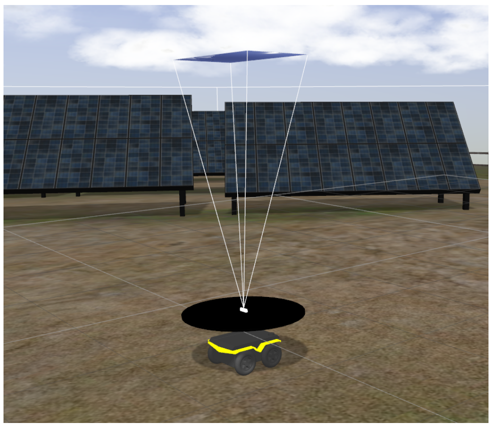


Figure 3.3: Simulated hardware.

The RGB-D used is an Intel Realsense D435, such as the one in figure 3.4, with a depth resolution of 1280x720 and a FOV of 85.2*58º and a depth range of around 10 m. As this RGB-D is commonly used in robotics, it is also thoroughly documented in ROS applications.

Figure 3.4: Intel Realsense D435 [6]

Figure 3.5: 3d Robotics Iris drone [7]

The selected UAV was the 3d Robotics Iris drone shown in figure 3.5 for being compatible with ROS and also for having the necessary software to run the system.

In this thesis the chosen hardware was used only in simulation, but due to the way it was implemented it should work just as well in the real hardware.

3.3 Methodologies

The landing platform will be placed on an UGV that will act as the ground station of the system. In order to reduce the payload on the UAV, and also allow the use of a better sensor along with better computational power, a stereo RGB-D camera was placed at the centre of the landing platform to guide the drone to its target.

3.3.1 PID Controller

In figure 3.6 a block diagram of the system used in this project can be seen.

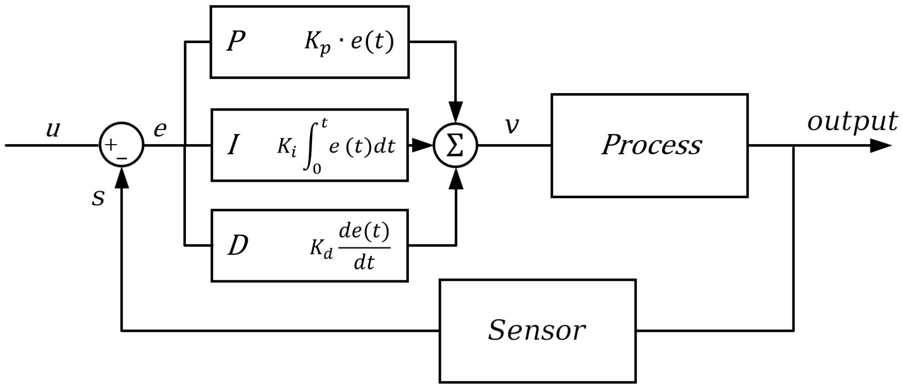


Figure 3.6: PID Controller Block Diagram.

Where u is the setpoint or desired position, in this case $X_{setpoint}$=0 m ,$Y_{setpoint}$=0 m, the $Z_{setpoint}$ is dynamic, while the drone is not centered on top of the platform ($XandY$ >= 0.1 m) the setpoint is 1 m, when it's centered ($XandY$ <= 0.1 m) it switches to 0.15 m;

s is the position read by the sensor;

e is the error of the system, that is, the difference between the set point and the current state of the system.

v is the output of the PID that, in this case, is the velocity calculated to drive the drone to the desired position.

The $Process$ is the change in position of the UAV once the output of the PID is applied.

The dynamics of the system can be tuned to achieve the desired behaviour by tuning the Gains (K_p, K_i and K_d). These influence how fast or slow the system gets to the desired set point, how fast it stabilizes on the set-point and how it reacts to external disturbances.

In the system in question, not all the coordinates are to be handled in the same manner so different Gains were set for handling the rate of change of speed in the horizontal plane (X and Y) and descent speed (Z) in the vertical plane.

3.4 Detection

The detection system is composed of 2 sub-systems. In the first stage, the system uses a depth-based pose estimation. In the second stage it uses an ARTag marker to estimate the drone's pose. The second and more precise pose estimation method is used whenever the marker is detected.

3.4.1 RGB-D

To reduce the drone's pose inaccuracy, once close to the landing location a RGB-D camera was used for detecting the drone and get a more accurate pose estimation. To achive this, the sensor was positioned in the centre of the platform. By doing this, the best possible FOV is achieved and also the ideal position to center the landing drone on the landing platform. To convert the depth measurements of the camera to a 3D coordinate system, rs2_deproject_pixel_to_point was used. This is a function provided by Intel, the camera manufacturer, that given a 2D pixel location on an image and the depth measurement, it can map this into a 3D point in the world frame.

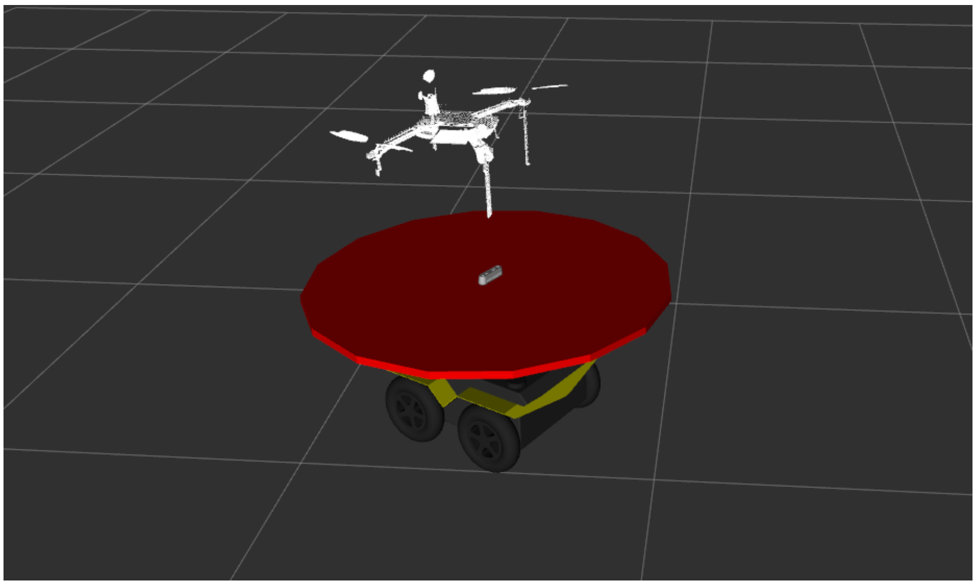


Figure 3.7: Projection of Depth camera's point cloud.

In figure 3.7 the point-cloud of the depth camera can be seen, where each white dot represents a depth measurement value for a given pixel. The system detects the closest point to the camera and by knowing the intrinsic camera parameters, a conversion between image frame and the real world can be made. This allows for an estimation of the distance between pixels and the distance in the real world. By having the depth measure for a specific pixel, a 3D coordinate in the real world (x, y, z) can be estimated, giving the relative position of the drone in relation to the camera. So, to run this detection method an empty open environment is assumed.

This method is not the most precises detection method as it assumes the closes point to the camera as the position of the drone, this can be the left front leg or other point on the drone's body that not it's centroid, as in an ideal detection method would be desirable. This method is good as it can detect the drone far from the base and provide a good enought of a pose estimation for the system to guide the UAV close to the platform. This method is capable of detecting the drone up to about 8 m in height. The depth FOV at maximum height can be seen in figure 3.8 below.

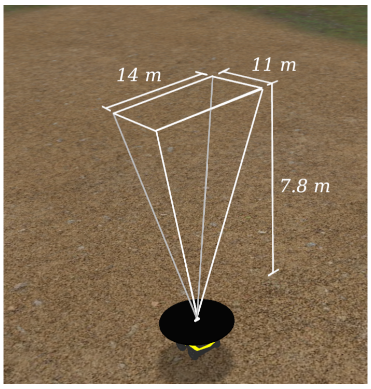


Figure 3.8: Projection of Depth camera FOV.

3.4.2 Fiducial Markers

Fiducial markers have been used in the recent past in computer vision for being computationally cheap and very precise in relative pose estimation. These markers, figure 3.9, of known pattern and size, when attached to an object can serve as a reference point of location, orientation and scale.

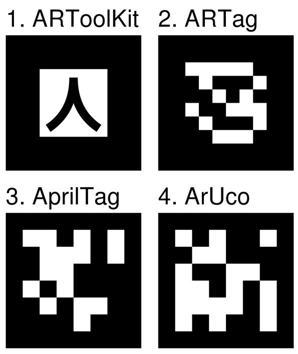


Figure 3.9: Different types of Augmented Reality (AR) markers. [8]

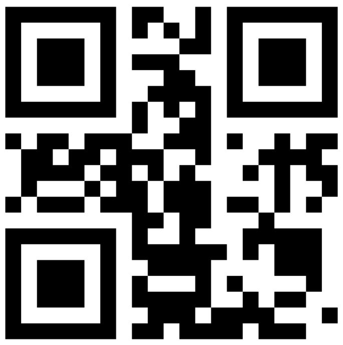


Figure 3.10: Exemple of QR code [9]

The most popular type of fiducial marker is the QR code, figure 3.10, that carries much more information than the typical marker used in robotics but, by doing so, it has a much more intricate design, which makes it almost impossible to correctly detect where variable light conditions and image distortion may blur the image of the marker. A typical fiducial marker is very easy to manufacture as it can be printed and glued to any surface that needs to be identified. This makes this marker a good solution to precisely locate the drone in the air once close to the platform, by fixing a marker to the underside of the UAV, such as the one shown in figure 3.11.

In robotics, the fiducial markers used tend to have less information coded in the marker than QR Codes, but as a trade off can be more robustly detected in all sorts of conditions. There are many types of fiducial markers and corresponding identification systems. The chosen system was ARTag tracking library. ARTag is short for Augmented Reality Tag, and consists in a dual tone square symbol with a solid black background and a 6x6 grid of high contrast interior cells like the one shown in figure 3.12.

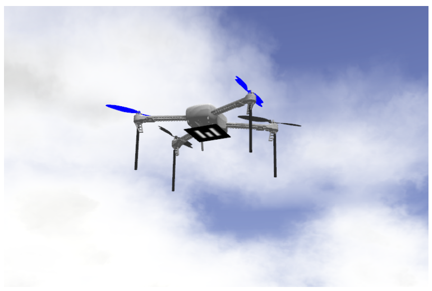


Figure 3.11: Iris drone model with ARTag

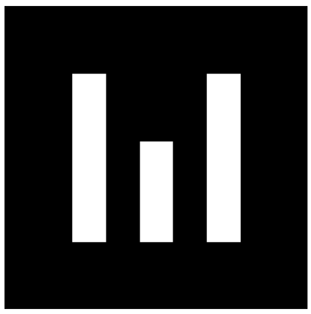


Figure 3.12: ARTag marker used

The library used to identify these markers was ar_track_alvar, which is an evolution of the standard ARTollkit [42]. One of the biggest challenges in detecting this type of markers is having this done in variable light conditions. To overcome this difficulty, Alvar uses adaptive thresholding, which consists in calculating a threshold value for each pixel using the values of the surrounding pixels; and for a more stable pose estimation, it uses an optical flow based tracking system. In order to find a marker, the detection algorithm uses an edge based approach. The edge pixels are found and used to link them into segments, which once grouped form quadrangles. The corners of these quadrangles are used to calculate a homography that allows the interior of the marker to be transfered into the world frame This approach makes it more robust to changing light conditions.

This library also has the ability to work with stereo camera, which provides a more precise pose estimation. An identified marker can be seen in figure 3.13 where, when close to the camera, the ARTag on the underside can be successfully detected, in this image illustrated as a purple square.

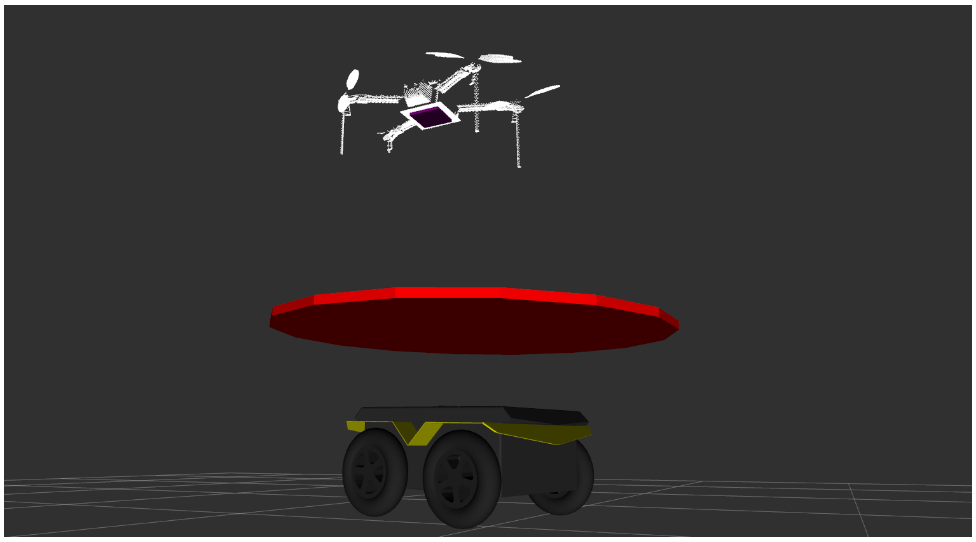


Figure 3.13: ARTag being detected in simulation

In the developed system the marker used is a 13x13 cm marker. This size marker was chosen in order to be the biggest possible without interfering with the thrust of the propellers. This method by itself would be insufficient as it can only detect the marker up to 3m away. But in conjunction with the first method of detection, the system can guide the drone using the first method that is able detect further away, but estimates the posing less accurately, and than when it enters the operating range of this marker detection method switch and get a very precise pose estimate.

3.5 Development Environment

3.5.1 Ardupilot

Ardupilot is an open source autopilot system that supports a wide range of different vehicles, from multicopters, helicopters and fixed-wing aircrafts, to rovers, boats and submarines [43]. This software is very popular and has a large user base which provides quick bug fixes and testing. The manufacture of the selected drone, 3DR's Iris, also supports Ardupilot, which was one of the reasons why this software was chosen for this thesis.

This autopilot software has the basic goal to provide control over the UAV. A basic view of the overall autopilot system can be seen in figure 3.14:

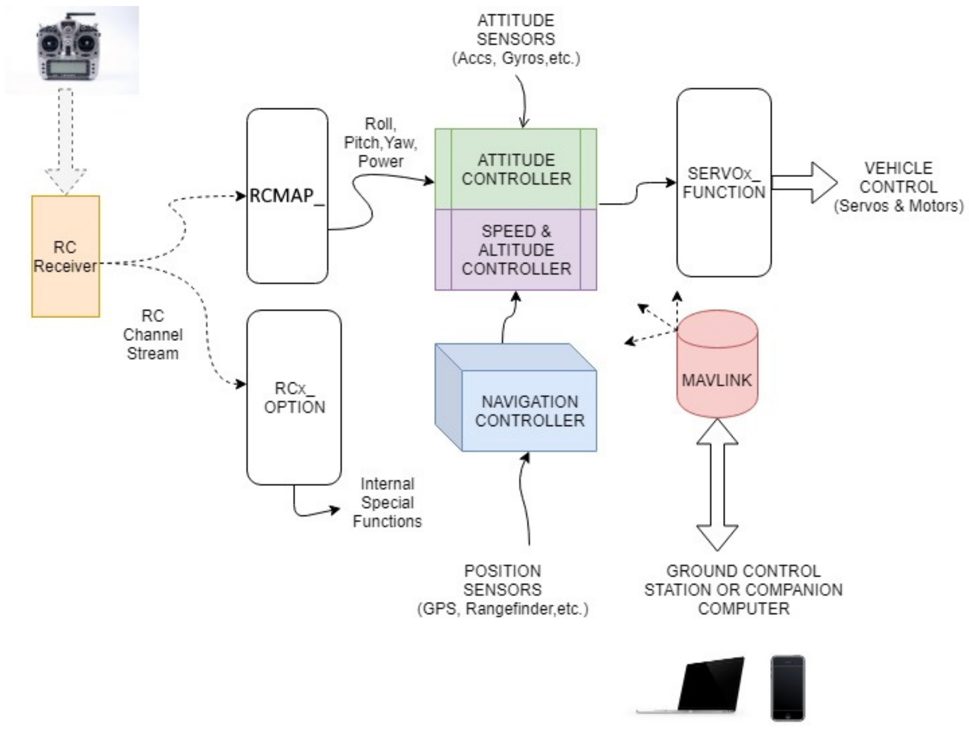

ATTITUDE SENSORS (Accs, Gyros,etc.)
RC Receiver
RCMAP_
Roll, Pitch,Yaw, Power
ATTITUDE CONTROLLER
SPEED & ALTITUDE CONTROLLER
SERVOx_ FUNCTION
VEHICLE CONTROL (Servos & Motors)
RC Channel Stream
RCx_ OPTION
NAVIGATION CONTROLLER
MAVLINK
Internal Special Functions
POSITION SENSORS (GPS, Rangefinder,etc.)
GROUND CONTROL STATION OR COMPANION COMPUTER

Figure 3.14: Ardupilot overall architecture. [10]

There are two main ways to control the UAV in Ardupilot; one of them is to use a conventional Radio Control (RC) controller, and the other is to use a ground control station. In order to communicate with the UAV, the ground control station uses MAVLink, which is a lightweight communication protocol. To enable compatibility with other software, MAVLink has other libraries such as MAVproxy, which is a command-line ground station software. This communication protocol allows the ground control station to send control instructions that get interpreted and converted to Roll/Pitch/Yaw/Throttle values in order to perform the desired tasks. In the the developed system the landing controller sends velocity and position commands to the autopilot, this gets interpreted in the corresponding controller and converted in instructions to the drone's motor as can be seen figure 3.14. To perform these actions the autopilot has inner control loops for attitude, speed and navigation controllers. MAVLink communications works in both ways, as the UAV sends information about its current state such as GPS and local position, IMU data, battery state and much more, and the ground station sends out control commands as previously explained. This communication is made by an established Internet Protocol (IP) connection between the ground control station and the drone's computer that gets established before the flight begins.

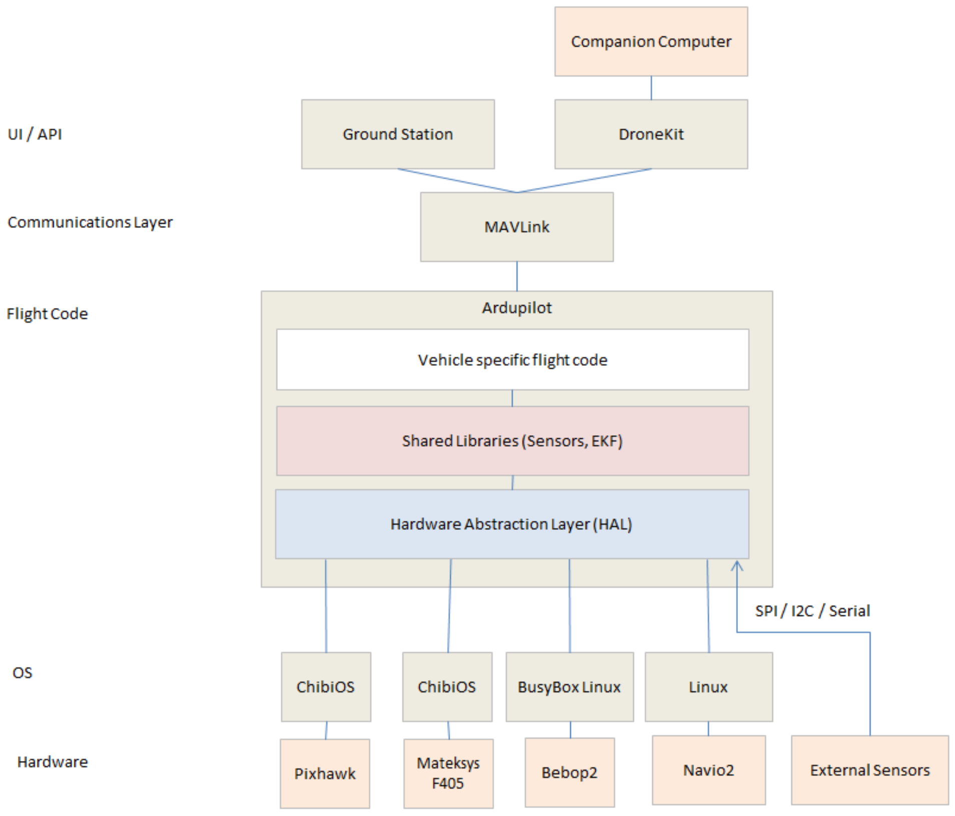


Figure 3.15: Basic structure of ArduPilot. [11]

In figure 3.15, the architecture of the Ardupilot system can be seen. ArduPilot also offers the possibility to run over SITL, and this allows it to run without any hardware. In this mode the sensor data comes from a flight simulator dynamic model. A big advantage of this system is that a flight simulation, with no hardware involved, is a fast and cheap way to test systems without the added risk of damaging physical components caused by a fault in the system, resulting in a potential crash and damage of expensive hardware. There are various flying modes for the autopilot, each mode with a goal in mind; the modes that are used here are the GUIDED mode and LAND mode. The GUIDED mode is meant to be used to dynamically guide a UAV to a desired location wirelessly, using a telemetry radio module and ground station application. The LAND mode is used to make the drone descend until it lands, and shut down the motors once it has landed. In this project, this mode is only used in the last stage of the landing process when it is less than 0.2 m high above the landing platform, it descends the rest of the way, and turns off the motors.

3.5.2 DroneKit

DroneKit is a cross-platform Application Programming Interface (API) that allows the development of Python applications for drones that communicate over MAVLink, using a low-latency link, with an Ardupilot flight controller [44]. These applications run on the MAV's companion computer and further increase the capabilities of the autopilot software by performing computationally heavy tasks that need a low-latency link, as are computer vision or path planning algorithms. The API offers a methodical access to the connected vehicle's state and parameter information and also enables not only direct control over the vehicle's movement and operation, but also mission management. DroneKit also provides a comprehensive set of functions and classes that make MAVLink commands easier to understand.

In this project, DroneKit-Python was used, which allows the development of python-based applications. In this project the main use of this development platform consisted in handling the data concerning the positions where the drone needs to be, and forwarding it on to the ArduPilot flight controller. This data is transfered using MAVLink, so to start the program a connection must be established between the autopilot and the DroneKit application.

Some Dronekit functions were used without any modifications while others were adapted to perform the desired task. The function **fly_tojackal (jackpose, altitude)** was created using Dronekit's **simple_goto (location)**. This funtion receives a GPS location, in this case the coordinates of the landing pad (jackpose), and the desired altitude, it checks whether the drone has arrived to the goal location, something that the base **simple_goto** function is unable to do. The **simple_goto** function receives a point with GPS coordinates in the $LocationGlobal$ or $LocationGlobalRelative$ frames. Both use WGS84 coordinate system for latitude and longitude but, in the Global frame, altitude is given in relation to the Mean Sea-Level (MSL) while in the GlobalRelative it is given in relation to the home position. This is the position where the drone was first initialized.

The function is defined as shown below:

```python
def fly_tojackal(jackpose, Altitude):
    """
    Flys to given position and altitude and checks if arrives to given pose
    """
    print("Flying towards jackal ...")
    # Prints the given location
    print("jack lat - ", jackpose.latitude)
    print("jack long - ", jackpose.longitude)
    print("jack alt - ", jackpose.altitude)
    point1 = LocationGlobal(jackpose.latitude, jackpose.longitude, jackpose.altitude
                                                                + Altitude)
    vehicle.simple_goto(point1)

    currentLocation = vehicle.location.global_frame
    distance = get_distance_metres(currentLocation, jackpose)

    # gets current position to check if it has arrived
    while distance >=0.3:
        currentLocation = vehicle.location.global_frame
        distance = get_distance_metres(currentLocation, jackpose)
        print("distance - ", distance)
        time.sleep(1)
    print("reached destination starting final approach")
```

The function receives a position in the $LocationGlobal$ frame, in this case it receives the landing pad location ($jackpose$), and a desired altitude, ($Altitude$). This variable is used to set the height of the drone in relation to the landing pad. After issuing the command to fly to the given location, it enters a loop to wait until the current position of the drone is within 0.2 m of the landing pad. To compare the current position with the one of the base, a DroneKit function was used **get_distance_metres(aLocation1, aLocation2)**, which receives two sets of coordinates and returns the distance between the two objects in metres.

To send the velocity commands to the drone, the function **send_ned_velocity(velocity_x, velocity_y, velocity_z, duration)** was used.

```
def send_ned_velocity(velocity_x, velocity_y, velocity_z, duration):
    """
    Move vehicle in direction based on specified velocity vectors.
    """
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        0b0000111111000111, # type_mask (only speeds enabled)
        0, 0, 0, # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z, # x, y, z velocity in m/s
        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
        0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)


    # send command to vehicle on 100 Hz cycle
    for x in np.arange(0,duration,0.01):
        vehicle.send_mavlink(msg)
        time.sleep(0.01)
```

The function generates a SET_POSITION_TARGET_LOCAL_NED MAVLink message that specifies the speed components of the vehicle. The message is resent to the vehicle at a 1 Hz rate but the refresh rate was modified to 100 Hz for a faster adjustment of the speed commands.

3.5.3 Gazebo Simulator

Gazebo is an open source simulator and physics engine [45]. It offers a very realistic 3D world with high-quality graphics that allow the users to simulate their systems in an environment that is as close to the reality as possible, as seen in figure 3.16. The simulator includes interfaces for a wide range of simulated sensors out of the box, and with an ever growing support community, new hardware is added to the list of simulated sensors every day.

The simulated world and robot models can be designed to suit each desired application, and this allows it to accurately simulate in complex environments. Plug-ins can also be developed to enable Gazebo to further improve its capabilities. There are other physics engines available but the Gazebo's already developed compatibility with Ardupilot's Software in the Loop (SITL) and ROS, makes it a perfect fit for this project.

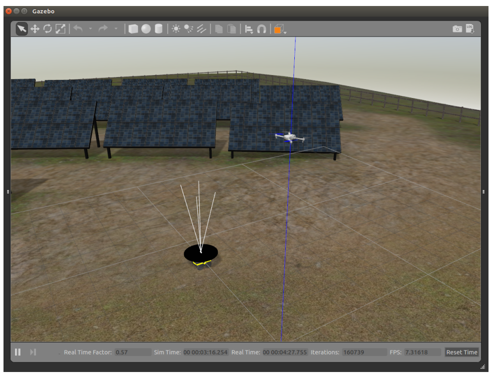


Figure 3.16: Gazebo world.

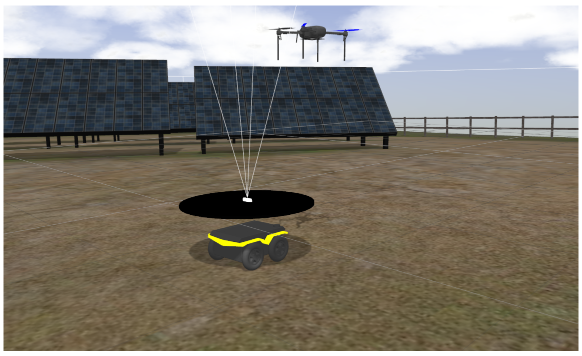
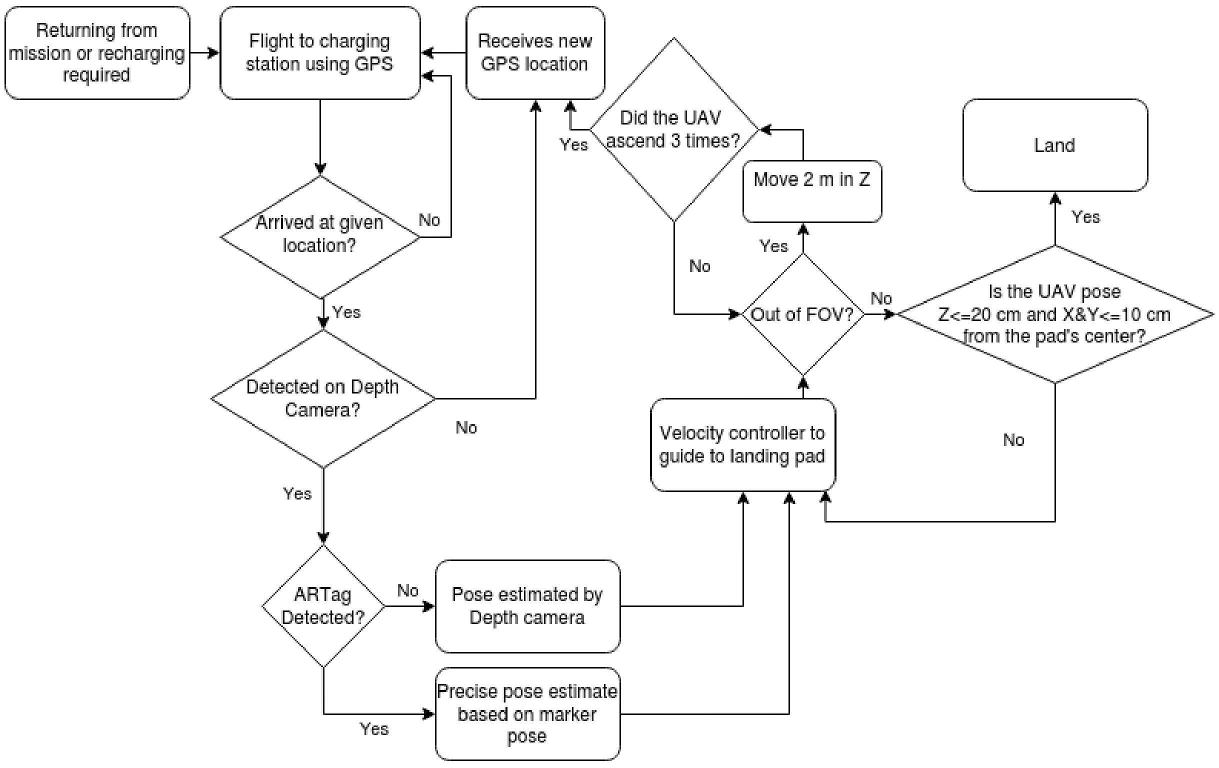


Figure 3.17: Modified Jackal Model

In order to run the developed system some modifications were made to the robot models in Gazebo. In the Jackal model, as shown in figure 3.17, a round landing platform with a Realsense camera in the centre was added. In the Iris model, as seen earlier in figure 3.11, an ARTag was added along with changing the color to a lighter shade of grey to improve the detection of the marker. The simulated world of a solar farm is a modified version of a world available with the Jackal model. The modified world is available in DURABLE's GitHub repository [46].

3.6 Program Architecture

The landing program flowchart of the developed solution can be seen in figure 3.18. The landing controller gets the GPS coordinates of the landing base and commands the drone to go to that location.



Returning from mission or recharging required
Flight to charging station using GPS
Receives new GPS location
Did the UAV ascend 3 times?
Yes
Land
Move 2 m in Z
Arrived at given location?
No
Yes
No
Out of FOV?
No
Is the UAV pose Z<=20 cm and X&Y<=10 cm from the pad's center?
Yes
Yes
Detected on Depth Camera?
No
Velocity controller to guide to landing pad
No
Yes
ARTag Detected?
No
Pose estimated by Depth camera
Precise pose estimate based on marker pose
Yes

Figure 3.18: Program flowchart.

When it arrives to the given location, it checks if it is on the camera's FOV. If it is not, it gets a new updated GPS position and is sent there. Otherwise, if it is on the camera's FOV, it calculates the relative position to correct the previous GPS positioning error. Following this procedure, the PID controller guides it to the landing platform. The PID controller receives the position calculated by the depth camera algorithm, if the ARTag is detected, then this more precise method of pose estimating is used to feed the PID controller. When it gets to within 10 cm of the center of the base and less than 20 cm high, the command to change mode to LAND is given and the program ends. If for some reason the drone gets out of the camera's FOV before it lands, a recovery action takes place. It raises 2 m in an attempt to return to where the depth camera's FOV is wider so it can be detected once again. Usually, in the experiments that were made, it is enough to get back to the FOV and be detected again but if it is not, then it raises another two times checking if it's out of the FOV. If raising three times 2 m is not enough to get detected, than the drone receives a new GPS point to go to and starts the landing cycle again.

3.7 Coordinate System

To guide the drone to the landing platform all the coordinate frames must be aligned. The Realsense sensor calculates the relative position of the UAV in relation to itself so, to guide the drone in the intended way the commands must be given in the correct coordinate system. Gazebo uses a NWU (North, West, Up) coordinate system. It can be seen in figure 3.19 that North is the X axis in red, West is the Y axis in green and the Up is the Z axis in blue.

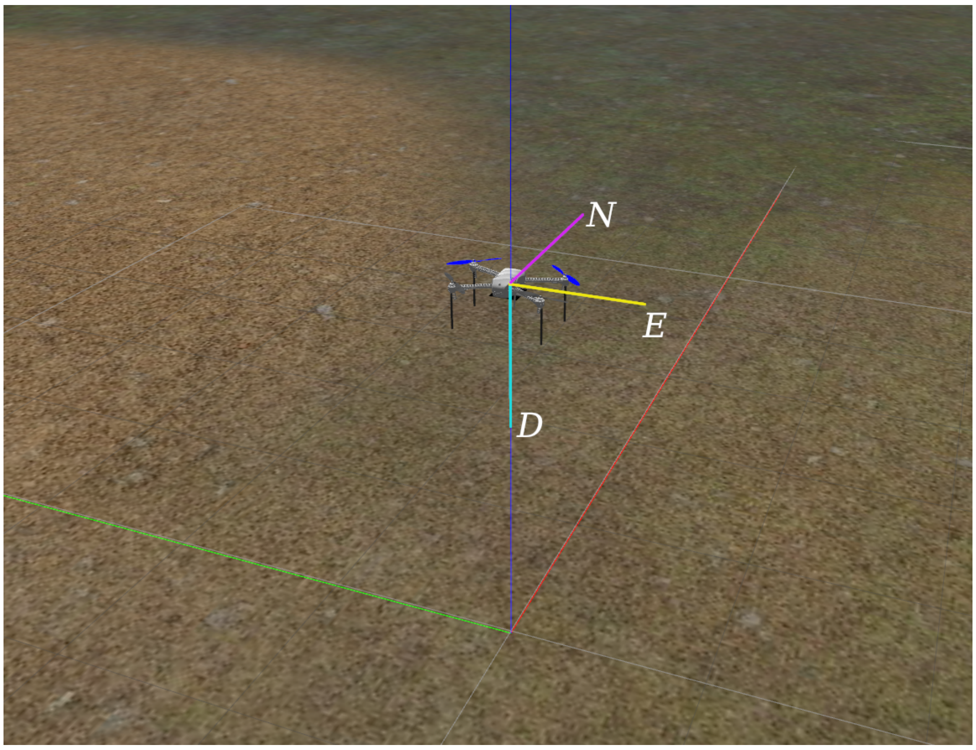


Figure 3.19: Gazebo coordinate system.

The drone receives a message in the NED (North, East, Down) format, illustrated in figure 3.19. The drone has a global NED frame where North is facing the front of drone, and East the right side. But the drone also has a local NED frame where North corresponds with Gazebo's X axis, East is the -Y of the West of the world frame. This transformation can be expressed by the transformation matrix 3.1 below.

$$\begin{bmatrix} X_{NED} \\ Y_{NED} \\ Z_{NED} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{NWU} \\ Y_{NWU} \\ Z_{NWU} \\ 1 \end{bmatrix} \tag{3.1}$$

To send the correct commands, the coordinates calculated by the sensor must be transformed to the correct frame. To do so, the coordinates are multiplied by the Rotation matrix as seen in equation 3.2:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = Rz \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.2}$$

Where Rz is the rotation on the Z axis

$$Rz = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

Where α is the angle of rotation of the base on the Z axis as seen in figure 3.20

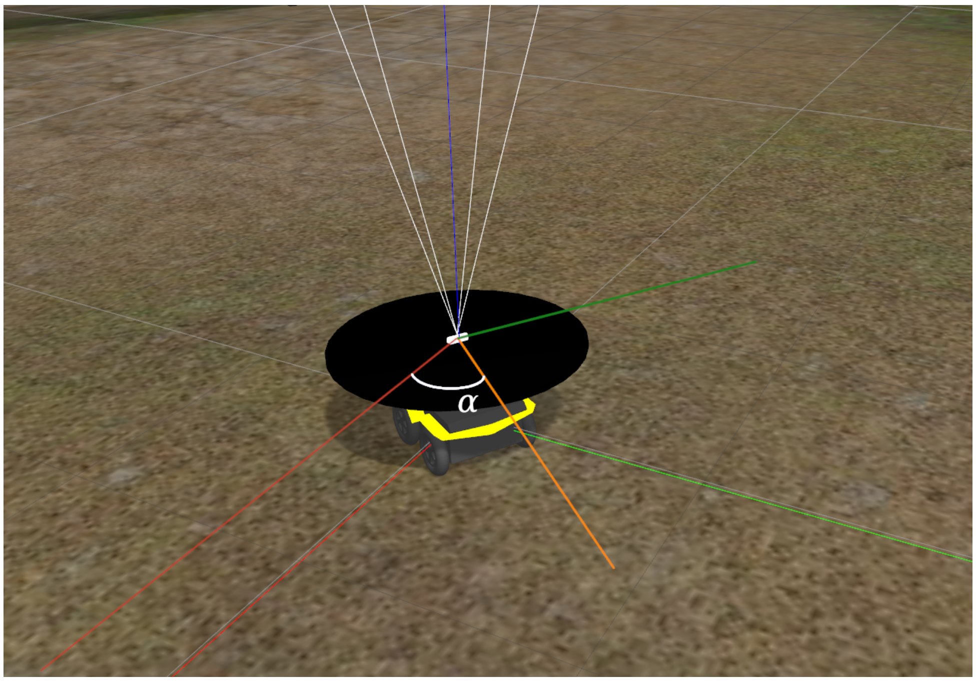


Figure 3.20: Illustration of the angle of rotation

4 Results

Contents

4.1 Results

The results of the developed system are presented in this section. The tests were conducted to test the various aspects of the system.

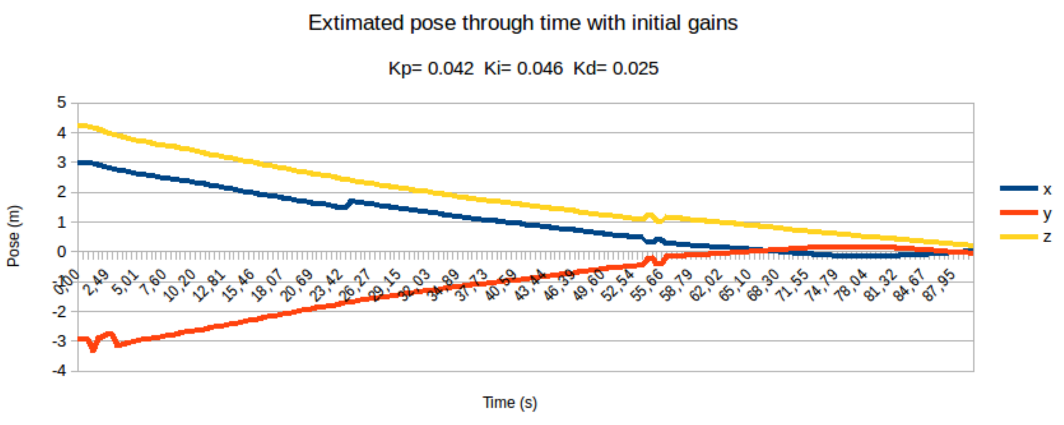
4.1.1 PID Controller

The job of the velocity controller is to center the drone above the center of the platform and make it descend until it lands. It achieves this by the use of three PID controllers, one for each coordinate, as previously stated. The initial gains of the PID controller caused the system to have a slow time of correction, and close to the setpoint it oscillated, so some tuning of the controller gains were required. Every gain has to be tuned in relation to the others, for the purpose of achieving a stable system, given that by increasing one gain to get a faster response can cause the system to overshoot and become unstable. With this in mind, the gains were increased in small increments, checking the system response and adjusting the other gains to compensate and achieve a stable system at all times.

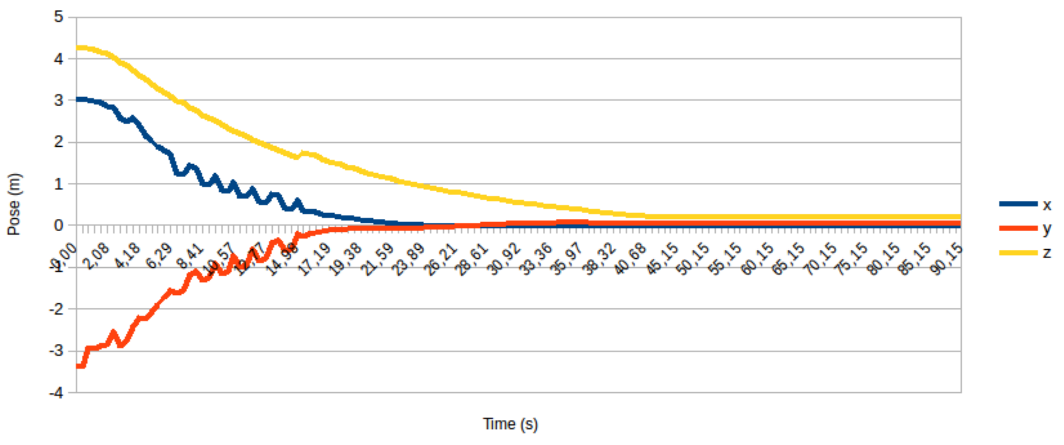
	K_p	K_i	K_d
Initial	0.042	0.046	0.025
Tuned	0.27	0.0107	0.027

Table 4.1: PID gain values for X an Y

In table 4.1 the values of the gains obtained with tuning are shown. In Z only the proportional gain was changed to increase the rate of descent. In order to validate the superior performance of the tuned system a set of tests was conducted.



(a)



(b)

Figure 4.1: (a) Plot of position over time of landing routine with initial gains (b) Plot of position over time of landing routine with tuned gains

Figure 4.1 shows the difference in performance between the tuned controller and the initial controller performing a landing routine. To test this, the same initial conditions were used in order to get comparable results. The drone would start from 5 m height and from 3 m away in the X an Y axis and approach the platform to land. At t=0 the approach starts and the end of the chart shows the moment when the drone lands.

A great improvement can be observed, as with the tuned gains the drone centers above the platform 3.4 times faster and lands 2.2 times faster. The jitter in the pose estimation can be justified by the switching between detection methods.

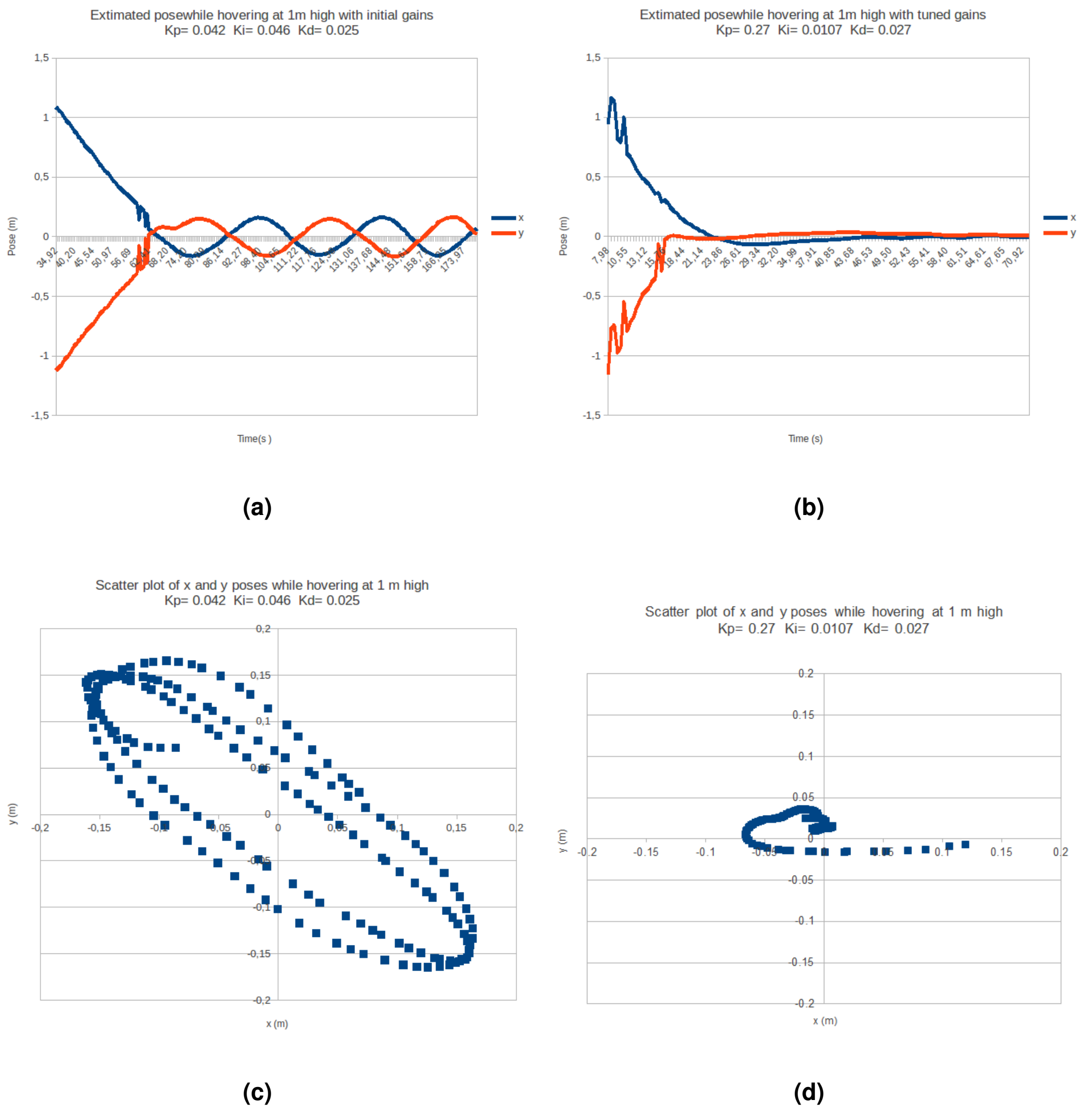


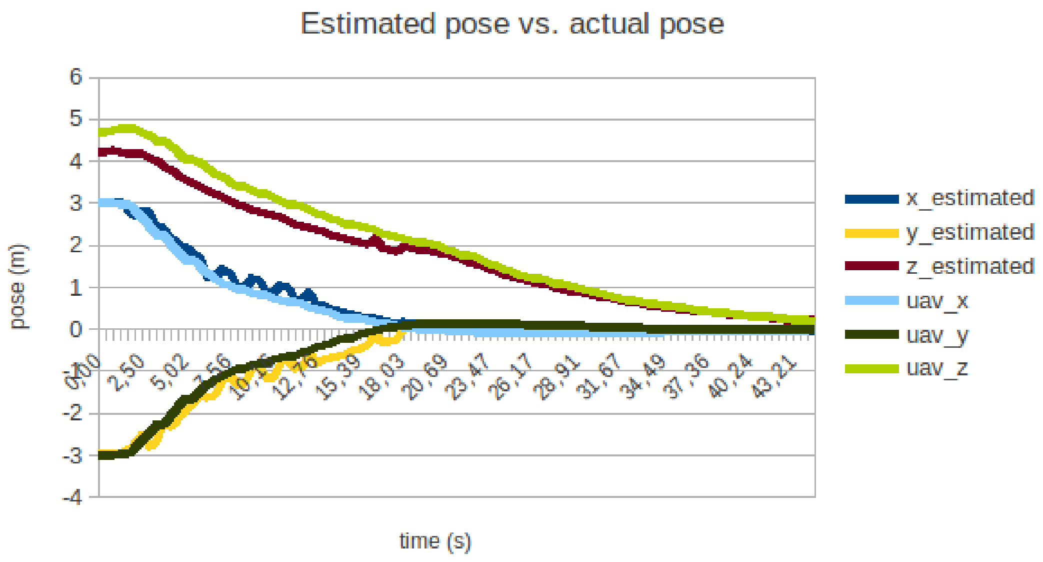
Figure 4.2: (a) Plot of position over time with initial gains (b) Plot of position over time with tuned gains (c) Scatter plot of X and Y with initial gains (d) Scatter plot of X and Y with tuned gains

To validate that the obtained system is stable, a hovering routine was made where the drone started at the same point as the landing routine but with the setpoint of Z at 1 m height, forcing the drone to remain above the platform and trying to center at x=0 and y=0.

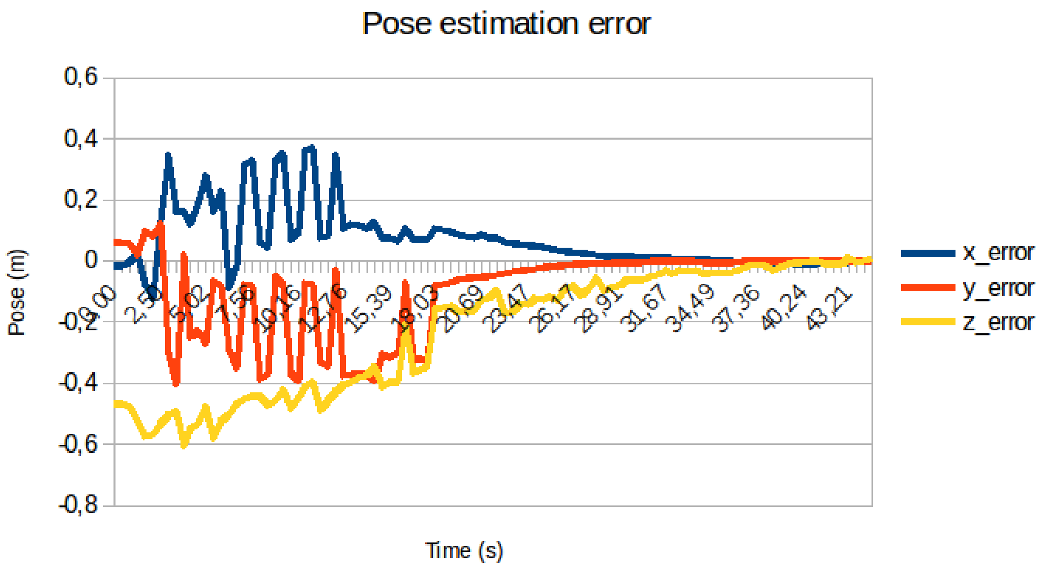
In Figure 4.2, it is clear that a much more stable system was achieved. Once the drone is above the platform with the tuned gains, it remains within 2 centimeters of the center, while the system with the initial gains oscillates around the setpoint with amplitudes in excess of 15 cm of the center.

4.1.2 Pose Estimation

To validate the chosen pose estimation method, the same landing routine was used.



(a)



(b)

Figure 4.3: (a) Plot of position over time of landing routine with initial gains (b) Plot of position over time of landing routine with tuned gains

In Figure 4.3 the error difference between the detected position and the exact position of the drone is shown. In the beginning, where the error is bigger, the depth camera positioning is used, but once the ARTag is detected, the position estimation becomes much more accurate.

	Error	X (m)	Y (m)	Z (m)
Depth camera pose estimation	Average	0.145	0.219	0.45
	Maximum	0.370	0.398	0.605
ARTag pose estimation	Average	0.027	0.015	0.068
	Maximum	0.107	0.826	0.172

Table 4.2: Error in pose estimation between methods statistics

In table 4.2 the difference in accuracy between the two methods is shown. It can clearly be seen that the error of the depth camera is about 4 times larger than with the ARTag positioning method.

To test if the system was able to land the drone successfully without the ARTag pose estimation some attempts to land the drone were made. In these tests the drone landed successfully, but in some tests one of the legs landed outside of the landing pad, although not enough off center to crash the drone. So it can be seen that even without the more precise detection method the drone was still able to land successfully. This is helped by the fact that the landing platform is 50 cm in radius and Iris has a dimension of 55cm motor to motor, so what was observed was that this extra 5 cm added to the centering error was what made one of the legs to land outside the platform. Were the platform to be smaller, the landing procedure without the ARTag positioning system would potentially become unsuccessful.

4.1.3 Landing

To validate the landing system a simulated launch and retrieve mission was simulated. The drone starts the flight from where it previously landed, and to get comparable results the landing pad in this test is assumed to be static.

The drone takes off to a height of 5 m and then proceeds to go to a given GPS location away from the landing base. When it arrives to the given point it receives the GPS coordinates of the landing pad. After this, the landing sequence is as previously described in section 3.6.

Average	X (m)	Y (m)	Landing Time(s)	$Time\sigma$	Success Rate
Drone pose	0.022	0.024	49.79	6.524	100%
Estimated	0.018	0.018			

Table 4.3: Repetitive Landing Statistics

50 landings were performed and the corresponding statistics are presented in table 4.3. These 50 landings were performed back to back to simulate a real application where the errors in the various

sensors would be cumulative. In table 4.3 it can be seen that the drone lands on average within 3 cm of the center of the base. It can also be seen that it takes on average about 50 seconds to land, counting from the moment when it's detected by the depth camera. All 50 landings were successful.

In figure 4.4, the landing position variation relative to the center of the landing pad is shown.

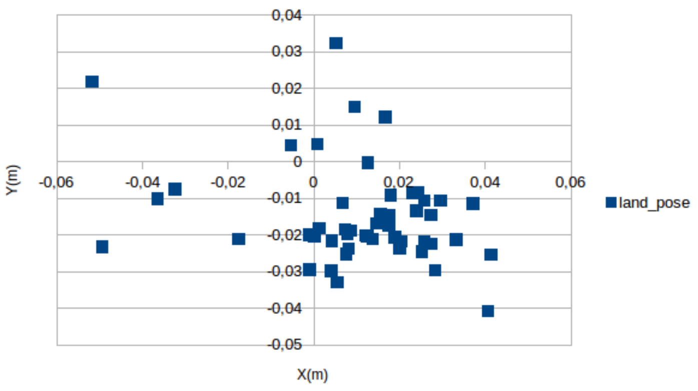
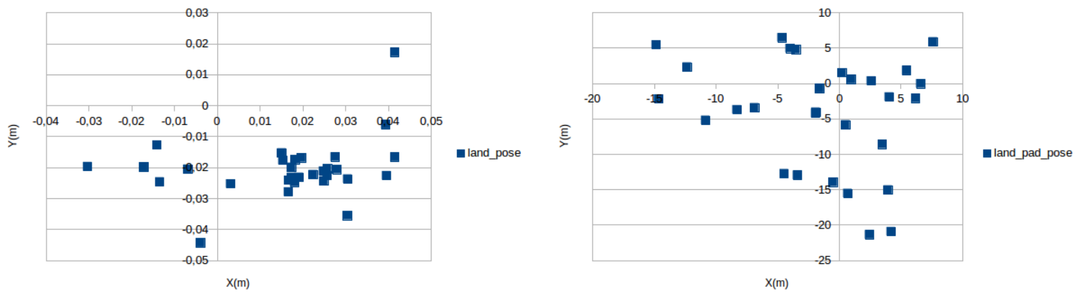


Figure 4.4: Scatter plot of landing position over 50 landings

To simulate a mobile platform another mission scenario was simulated. The drone has the same routine as before where it takes off to a height of 5 m and than proceeds to go to a given GPS location away from the landing base. Then it receives the GPS coordinates of the landing pad but this time the landing pad moves randomly from place to place while the drone is executing the task. To test this 30 landings were performed, and the respective statistics are presented in table 4.4.

Average	X (m)	Y (m)	Landing Time(s)	$Time\sigma$	Success Rate
Drone pose	0,022	0,022	65,91	10,20	100%

Table 4.4: Repetitive landing on mobile platform statistics



(a) **(b)**

Figure 4.5: (a) Scatter plot of the final landing pose in relation the the center of the landing pad. (b) Scatter plot of landing base position for each landing

In table 4.4 a larger standard deviation and longer average landing time can be seen, when compared with the landings on the static landing platform. In some of the landing attempts this is due to the base being still moving, resulting in a longer approach and landing time. Also in some of the studied cases the drone was transported on the platform from one place to another, thus simulating a mobile charging station. In figure 4.5 it can be seen in image (a) that the drone lands very accurately within a radius of 4cm of the center of the base. In image (b) the landing pad position in the field can be seen for each of the 30 landings.

5

Conclusion

Contents

5.1 Conclusions

This thesis achieved the proposed goals. As was shown in the previous section, an off-board vison-based system that is able to accurately estimate the UAV's pose, and a high-level motion control system that can control the drone guiding it to the landing base as well as make it land were successfully developed and implemented. As the designed system was developed and tested using Ardupilot and DroneKit, it can theoretically work with a wide range of MAVLink compatible devices and can easily be implemented in real hardware.

The results that were obtained after testing the systems were adequate. However, subjecting the UAV to disturbances in the detection component, such as change in the surrounding light conditions or in the environment, such as variable wind speeds or direction, would have been a good test to further validate the system.

5.2 Future Work

The most immediate future work would be to test and validate the proposed system on physical hardware. This task was originally intended to be part of this project but due to time constraints this was deemed as not feasible.

The depth camera detection algorithm is not a perfect solution as it assumes an empty open environment to estimate the pose of the drone. A possible solution would be to train a convolution neural network or use other image processing algorithms to be able to detect the drone in the air and get a depth measurement from the area identified as a drone.

Further tuning of the system could be done in order to allow the system to perform landings onto a moving platform. The system would need to correct the drone's position faster, and a trajectory planner would be needed as well.

Fusion between on-ground and on-board solutions could further improve reliability and robustness of the system. With this, obstacle avoidance and a more reliable landing system could be achieved.

Bibliography

[1] G. T. Black, K. Cohen, and C. Ronflé-Nadaud, "Integration in the national airspace (europe and usa)–uav classification and associated missions, regulation and safety, certification and air traffic management," in *Multi-Rotor Platform-based UAV Systems*. Elsevier, 2020, pp. 1–25.

[2] "GeeksforGeeks matlab | rgb image representation," https://www.geeksforgeeks.org/matlab-rgb-image-representation/, accessed: 2021-3-20.

[3] H. W. Ho and Q. Chu, "Automatic landing system of a quadrotor uav using visual servoing," 04 2013.

[4] E. Shirokova, A. Azarov, and I. Shirokov, "The system of wireless energy transfer," 01 2019, pp. 1060–1064.

[5] "Clearpath Robotics webpage," https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/, accessed: 2021-8-20.

[6] "Intel Realsense D435 sensor webpage," https://www.intelrealsense.com/depth-camera-d435/, accessed: 2021-6-15.

[7] "3DR Iris drone photo webpage," https://quadcopterarena.com/wp-content/uploads/2015/12/3d-robotics-iris-plus-review.jpg, accessed: 2021-8-15.

[8] "ARTag photo webpage," https://commons.wikimedia.org/wiki/File:Comparison_of_augmented_reality_fiducial_markers.svg, accessed: 2021-9-15.

[9] "QR code photo webpage," https://commons.wikimedia.org/wiki/File:QR_Code_Example.svg, accessed: 2021-9-15.

[10] "Ardupilot overall scheme image website," https://ardupilot.org/copter/docs/common-basic-operation.html, accessed: 2021-07-20.

[11] "ArduPilot architecture photo webpage," https://ardupilot.org/dev/docs/learning-ardupilot-introduction.html, accessed: 2021-6-15.

[12] M. Biczyski, R. Sehab, J. F. Whidborne, G. Krebs, and P. Luk, “Multirotor sizing methodology with flight time estimation,” *Journal of Advanced Transportation*, vol. 2020, 2020.

[13] L. W. Traub, “Optimal battery weight fraction for maximum aircraft range and endurance,” *Journal of Aircraft*, vol. 53, no. 4, pp. 1177–1179, 2016.

[14] L. Cantelli, M. Mangiameli, C. D. Melita, and G. Muscato, “Uav/ugv cooperation for surveying operations in humanitarian demining,” in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013, pp. 1–6.

[15] K. Yu, A. K. Budhiraja, and P. Tokekar, “Algorithms for routing of unmanned aerial vehicles with mobile recharging stations,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5720–5725.

[16] “Article on important dates on aviation webpage,” https://www.thisdayinaviation.com/23-august-1937/, accessed: 2021-4-20.

[17] Y. Feng, C. Zhang, S. Baek, S. Rawashdeh, and A. Mohammadi, “Autonomous landing of a uav on a moving platform using model predictive control,” *Drones*, vol. 2, no. 4, 2018. [Online]. Available: https://www.mdpi.com/2504-446X/2/4/34

[18] “FlytBase webpage,” https://flytbase.com/precision-landing/, accessed: 2021-5-20.

[19] R. PS and M. L. Jeyan, “Mini unmanned aerial systems (uav)-a review of the parameters for classification of a mini uav.” *International Journal of Aviation, Aeronautics, and Aerospace*, vol. 7, no. 3, p. 5, 2020.

[20] K. Wenzel, P. Rosset, and A. Zell, “Low-cost visual tracking of a landing place and hovering flight control with a microcontroller,” *Journal of Intelligent and Robotic Systems*, vol. 57, pp. 297–311, 01 2010.

[21] S. Lange, N. Sünderhauf, and P. Protzel, “Autonomous landing for a multirotor uav using vision,” *In Workshop Proceedings of SIMPAR 2008 Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots*, 01 2008.

[22] G. Xu, Y. Zhang, S. Ji, Y. Cheng, and Y. Tian, “Research on computer vision-based for uav autonomous landing on a ship,” *Pattern Recognition Letters*, vol. 30, no. 6, pp. 600–605, 2009.

[23] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, “Visually guided landing of an unmanned aerial vehicle,” *IEEE Transactions on Robotics and Automation*, vol. 19, no. 3, pp. 371–380, 2003.

[24] C. Martínez, P. Campoy, I. Mondragón, and M. A. Olivares-Méndez, “Trinocular ground system to control uavs,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 3361–3367.

[25] Ma, Yan, “Coordinated landing and mapping with aerial and ground vehicle teams,” 2012. [Online]. Available: http://hdl.handle.net/10012/6993

[26] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[27] W. Kong, D. Zhang, X. Wang, Z. Xian, and J. Zhang, “Autonomous landing of an uav with a ground-based actuated infrared stereo vision system,” 11 2013, pp. 2963–2970.

[28] B. Erginer and E. Altug, “Modeling and pd control of a quadrotor vtol vehicle,” in *2007 IEEE Intelligent Vehicles Symposium*, 2007, pp. 894–899.

[29] H. Voos and B. Nourghassemi, “Nonlinear control of stabilized flight and landing control for quadrotor uavs,” in *7th Workshop on Advanced Control and Diagnosis ACD 2009, Zielo Gora, Poland*, 2009.

[30] B. Ahmed and H. R. Pota, “Backstepping-based landing control of a ruav using tether incorporating flapping correction dynamics,” in *2008 American Control Conference*, 2008, pp. 2728–2733.

[31] B. Ahmed, H. R. Pota, and M. Garratt, “Flight control of a rotary wing uav using backstepping,” *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 20, no. 6, pp. 639–658, 2010.

[32] S. M. B. Malaek, N. Sadati, H. Izadi, and M. Pakmehr, “Intelligent autolanding controller design using neural networks and fuzzy logic,” in *2004 5th Asian Control Conference (IEEE Cat. No.04EX904)*, vol. 1, 2004, pp. 365–373 Vol.1.

[33] M. Montemerlo, N. Roy, and S. Thrun, “Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit,” *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 3, pp. 2436–2441 vol.3, 2003.

[34] “Carmen Robot Navigation Toolkit webpage,” http://carmen.sourceforge.net/home.html, accessed: 2021-7-30.

[35] D. Calisi, A. Censi, L. Iocchi, and D. Nardi, "Openrdk: a modular framework for robotic software development," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 1872–1877.

[36] "OpenRDK website," http://openrdk.sourceforge.net/index.php?n=Main.HomePage, accessed: 2021-07-30.

[37] J. Jackson, "Microsoft robotics studio: A technical introduction," *IEEE robotics & automation magazine*, vol. 14, no. 4, pp. 82–87, 2007.

[38] "Microsoft Robotics Development Studio webpage," https://docs.microsoft.com/en-us/previous-versions/microsoft-robotics/dd146032(v=msdn.10), accessed: 2021-7-30.

[39] Z. Jiang *et al.*, "An autonomous landing and charging system for drones," Ph.D. dissertation, Massachusetts Institute of Technology, 2019.

[40] D. Pebrianti, F. Kendoul, S. Azrad, W. Wang, and K. Nonami, "Autonomous hovering and landing of a quad-rotor micro aerial vehicle by means of on ground stereo vision system," *Journal of System Design and Dynamics*, vol. 4, no. 2, pp. 269–284, 2010.

[41] "Clearpath Robotics simulating jackal website," https://www.clearpathrobotics.com/assets/guides/kinetic/jackal/simulation.html, accessed: 2021-2-20.

[42] "AR Track Alvar wiki ros webpage," http://wiki.ros.org/ar_track_alvar, accessed: 2021-4-30.

[43] "ArduPilot webpage," https://ardupilot.org/, accessed: 2021-9-15.

[44] "DroneKit-Python webpage," https://dronekit-python.readthedocs.io/en/latest/about/index.html, accessed: 2021-7-15.

[45] "Gazebo simulator webpage," https://ardupilot.org/, accessed: 2021-8-15.

[46] "DURABLE git webpage repository," https://github.com/durable-ist/Multi_Robot_Simulation, accessed: 2021-08-30.

A

Code of Project

A.1 Code Repositories

All of the used code is available at GitHub, as well as the developed code.

Some packages were modified to suit the desired function and are available ate GitHub at Detection Repository.

Detection GitHub	https://github.com/MartimBraga/Detection
Ardupilot code	https://github.com/ArduPilot/ardupilot/tree/master/ArduCopter
Dronekit code	https://github.com/dronekit/dronekit-python
Original PID code	https://github.com/roscitra/Quadcopter-range-control-system-using-PID-method-Python-
Simulation enviroment	https://github.com/durable-ist/Multi_Robot_Simulation