Autonomous Landing of an Aerial Robot on a Moving Platform

Isabel Maria Pereira Castelo

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Dr. Meysam Basiri
Prof. Pedro Manuel Urbano de Almeida Lima

December 2021

Declaration

Acknowledgments

There are many people whom I would like to thank for contributing, either directly or indirectly, to this thesis. First I would like to thank my supervisors, Dr. Meysam Basiri and Professor Pedro Lima, without their guidance and support the completion of this dissertation would not have been possible.

Second, I want to thank my family, for supporting and valuing my future by allowing me to move away from home to pursue my dreams. A special thanks to my brother, for being a role model and showing what a real engineer should look like.

And lastly but not least important, I would like to thank all of my friends and colleagues from university, for turning this roller coaster experience into a pleasureful ride.

Resumo

Veículos aéreos autónomos têm recebido um acréscimo de interesse devido à sua versatilidade e aplicabilidade, no entanto as suas baterias limitadas requerem constante recarregamento. Sistemas cooperativos entre vários robôs proporcionam uma solução que torna estes sistemas totalmente autónomos ao utilizar um robô terrestre como uma plataforma de carregamento para os drones. Esta cooperação entre robôs aéreos e terrestres permite-os resolver tarefas de alta complexidade e ultrapassar algumas das suas limitações individuais.

Esta tese propõe uma solução baseada em processamento de imagem e visão que permita ao UAV detetar o UGV, planear a sua trajectória e autonomamente aterrar no veículo. O algoritmo de deteção por visão usa marcadores fiduciais, um tipo de marcador artificial com identificador único, mais especificamente ArUcO markers. De seguida, a aterragem é implementado a partir de uma máquina de estados, que efetua as tomadas de decisão de alto nível do drone, e um controlador em cascata, constituido por três controladores independentes para os comandos de atitude e altitude, que são posteriormente enviados para o autopilot no controlador interno.

Esta implementação foi testada tanto no Simulador Gazebo 7 como no drone real. Em ambos os sistemas, vários testes foram feitos, tal como teste de limitações do hardware, experiências de afinação do controlador, testes de consistência, e vários testes de voo. As experiências feitas mostram todas o robô a, bem sucedidamente, aterrar na plataforma de recarregamento.

Palavras-chave: UAV-UGV, Aterragem Autónoma, Aterragem por visão, Sistemas Cooperativos de multi-robôs, ArUcO Marcadores, Pixhawk Autopiloto

Abstract

Unmanned Aerial Vehicles (UAVs) have had an increase of interest due to their versatility and variety of applications, however their limited battery life requires constant manual recharging. Heterogeneous multi-robot systems provide a solution to make these systems fully autonomous, by allowing an Unmanned Ground Vehicle (UGV) to serve as a recharging station for the aerial vehicle. The cooperation between aerial and ground robots allows them to solve high complexity tasks and overcome their individual limitations.

This work proposes a vision-based approach that allows an aerial robot to detect a mobile charging station, perform a trajectory planning and autonomously land on the vehicle. The vision-based detection algorithm makes use of fiducial markers, a type of unique identifiable artificial marker, more specifically the ArUcO markers. Afterwards, the landing is implemented with the help of a state machine, that overviews the high-level behaviour of the UAV, and a cascaded controller, with three independent controllers for the Attitude and Altitude commands, which are then sent to the autopilot for inner-loop control.

This implementation was tested both in the Simulator Gazebo 7 and on the real drone. On both systems several tests were performed, such as testing the limitations of the hardware, controller tuning experiments, tests for consistency, and a variety of flight tests. The range of test run show that the robot successfully detected and landed on a charging pad.

Keywords: UAV-UGV, Autonomous landing, Vision-based landing, Cooperative Multi-robot Systems, ArUcO Markers, Pixhawk Autopilot

Contents

List of Tables

List of Figures

Nomenclature

CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradients
DOF	Degrees of Freedom
DQL	Deep Q-Learning
DQN	Deep Q-Networks
EKF	Extended Kalman Filter
ENU	East-North-Up
FLC	Fuzzy Logic Control
LKF	Linear Kalman Filter
LOS	Line-of-Sight
LQR	Linear Quadratic Regulator
MAV	Micro Aerial Vehicle
MRS	Multi-robot Systems
NEU	North-East-Up
PID	Proportional Integral Derivative
PN	Proportional Navigation
ROS	Robot Operating System
SLC	Sliding Mode Control
SSH	Secure Shell
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UKF	Unscented Kalman Filter
w.r.t	with respect to

Chapter 1

Introduction

A fully autonomous robot needs to be able to adapt to all circumstances around it, in an unpredictable environment this is harder to achieve with only one robot, leading to the formation of heterogeneous multi-robot systems (MRS) [31] to complete a variety of tasks. These systems can be composed with robots of various types and capabilities, such as Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs). The robots are designed to cooperate with each other and humans, in order to successfully accomplish high complexity tasks.

Further developing these cooperative heterogeneous MRS, brings advantages to several fields [37], such as surveillance, rescue operations, environmental monitoring, security measurements, emergency response systems and even household chores.

This introductory Chapter starts by placing the thesis in the context of this research topic. This is followed by a topic overview and the objectives, where the goal of the thesis is clarified. The Chapter ends with an outline of the thesis.

1.1 Motivation

The Durable project aspires to accelerate the deployment of renewable energies, in the Atlantic, area through the use of aerospace technologies. Expanding the use of renewable energy is a crucial step to be taken in the present, since it is a sustainable resource that does not run out. Renewable energy solar plants require constant inspection of physical damage and electrical connections, and occasional maintenance for the proper functioning of the system.

UAVs need to have a limited size and consequent small payload in order to be easy to operate and maneuver, and to be versatile and applicable in different scenarios. However, this results in a small power autonomy, which highly constrains UAV missions, resulting in short-range flights and a consequent limited applicability. However, the advantages they bring to a variety of fields of work greatly exceed their issues, such as being easier and cheaper to deploy than manned aircraft and providing a safe reconnaissance and flight to areas of difficult access. UGVs [13], on the other hand, are able to carry a large payload, hence have an extended battery life. A possible solution for this problem is to employ an heterogeneous team of ground and aerial robots, where the ground robots can act as a mobile charging station for the UAVs to use in between flight operations.

This leads to the necessity of the UAV to fully autonomously land on the UGV for recharging, so as to make the UAV fleet more dependable, by flying autonomously and uninterrupted for long time periods.

By using UGVs as a recharging platform for the UAVs, this heterogeneous MRS provides a safeguard for the UAVs limitations, increasing/distributing the payload capacity and run time, and cooperating perception-wise to achieve a broader field of view for inspection, as well as using the UGVs for a close up view and for repairing.

To achieve the landing, real-time vision-based localization and tracking during flights is a fundamental step. Vision-based navigation has proven to be a promising research area for autonomous navigation due to the rapid development of computer vision. First, the visual sensors can provide abundant real time information of the surroundings; second, their anti-interference ability makes visual sensors highly appropriate for perception of dynamic environments; third, it's a lightweight sensor, which pairs well with the UAV's small capacity.

Successfully accomplishing a fully autonomous UAV landing, benefits not only the inspection and recharging needed for the Durable project, but also other fields and applications. Such as a drone based delivery service, or even a landing to transfer sensor data acquired during the flight.

1.2 Objectives

The main goal of this thesis is to design an onboard system for the UAVs to detect, reach and land on a possibly moving UGV.

The plan, as previously mentioned, is to exploit and use a vision-based approach. The main objectives are as follows:

1. Vision-based UGV detection - Obtain the UGV's pose relative to the UAV by using an artificial landmark;

2. State estimation (Relative Localization) - After obtaining the target's relative pose, compute the best angles to move towards the target;

3. Motion Control for landing - Define a high level control state machine that, depending on the current state of the UAV, determines the next state.

1.3 Thesis Outline

Chapter 2 contains some fundamental concepts necessary to understand the remaining document and implemented solution.

Chapter 3 analyses how other approaches addressed this issue in several topics, e.g. which landmarks were used; how they were detected; how the UAV is controlled; what are the several states that compose a landing; how the detection can be performed without landmarks; and what hardware was used.

Chapter 4 contains a description of the hardware used, including the UAV, the UGV and some of their parts. Then, focusing on the UAV, the several reference frames are analysed, followed by the equations that allow the control of the quadrotor and the analysis of the controller implemented, together with the autopilot's controller. Afterwards, the computer vision method used to detect the UGV, the pattern that goes on top of the UGV and the transforms necessary to obtain the accurate measurements, are defined. Then, it is explained how this will be implemented in ROS, such as the packages used for communicating with drones and the package used to implement the high level state machine. And finally, a flowchart with a succinct overview of the implemented code is presented.

Chapter 5 contains the results of the experiments performed in the simulator Gazebo. On the simulation, first some limitations of the UAV and its camera are analysed to be taken into account for the remaining tests. Then the entire tuning of the implemented controller is outlined along with its consistency. Afterwards, different scenarios from the one used to tune the controller are tested to show versatility of the implementation.

Chapter 6 shows the experiments performed on the real drone. First on the Motion Capture System; and then in a real environment, the tuning performed and the landing tests.

Chapter 7 outlines the overall conclusion of this thesis, what was implemented and achieved with this work, and then some future work.

Chapter A contains the GitHub repositories that were implemented during this thesis or for the overall project and used in this thesis.

Chapter 2

Background

This section of the report will overview the fundamental concepts necessary to understand the methodology used to autonomously land a UAV on top of a moving target.

2.1 UAV Controller

2.1.1 Reference Frames

There are 5 main reference frames in use to describe the dynamics of an aircraft, the frames in Figure 2.1 are the following:

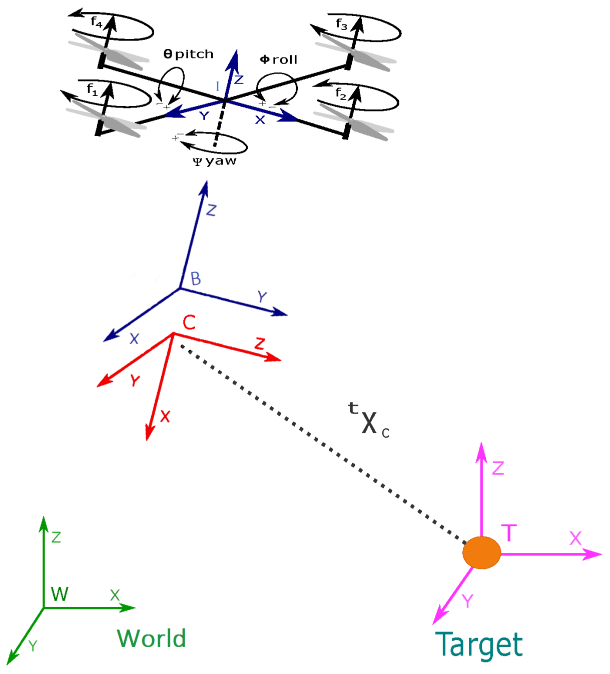


Figure 2.1: Reference Frames.

- Inertial Frame [**I**] - Inertial fixed frame whose with an East-North-Up (ENU) frame w.r.t. to the World frame;
- Body Frame [**B**] - Mobile frame whose dynamic behavior can be described relatively to the fixed frame;
- World Frame [**W**] - Earth fixed frame with a North-East-Up (NEU) frame;
- Camera Frame [**C**] - Fixed to the Body Frame, therefore performs the same rotations and translations;
- Target Frame [**T**] - Body frame of the mobile platform. Origin at the center of gravity of the UGV, NEU reference frame.

2.1.2 Equations of Motion

The orientation of the body frame relative to the World frame, can be obtained using the rotation matrix in (2.1). Which is obtained from the product of the rotation matrices of the x, y and z axes, whose angles are roll, pitch and yaw respectively.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi & cos\phi \end{bmatrix} R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{bmatrix} R_z(\psi) = \begin{bmatrix} cos\psi & -sen\psi & 0 \\ sen\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$^{B}R_W = R_x(\phi)R_y(\theta)R_z(\psi) \tag{2.1}$$

The forces acting upon a drone are the thrust/moment generated by the rotors, consequently lift and drag arises as air moves past the UAV. Lift pushes the object upwards, whereas drag is a sort of air resistance, slowing it down. And finally, the gravity.

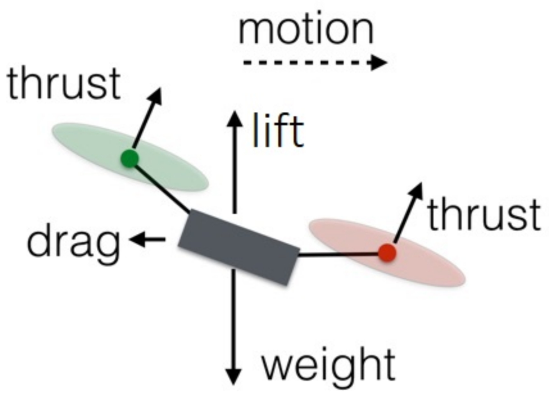


Figure 2.2: Forces applied to the UAV.[3]

The total force on the UAV in the body frame is given as,

$$F^B = F_g^B + F_T^B + F_{aero}^B \quad . \tag{2.2}$$

Where F_g is the gravitational force, F_T is the force of the thrust, and F_aero is the force resulting of the aerodynamic drag and lift coefficients, C_D and C_L.

The gravitational force is along the z axis of the world frame, therefore needs to be translated into the body frame,

$$F_g^B =^B R_g F_g^W =^B R_g \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \quad . \tag{2.3}$$

When modeling the UAV, it is taken in consideration that it is a rigid body, so the Newton and Euler equations are used to describe its dynamics. The force and moment equations are represented in (2.4a) and (2.4b) respectively,

$$F = \frac{\partial}{\partial t}[mv]_B + \dot{\omega} \times [mv]_B \quad , \tag{2.4a}$$

$$M = \frac{\partial}{\partial t}[I\dot{\omega}]_B + \dot{\omega} \times [I\dot{\omega}]_B \quad , \tag{2.4b}$$

where *v* is the vector of linear velocities, ω the vector of angular velocities, and m is the mass of the body. The inertia matrix I is represented in (2.5),

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \quad . \tag{2.5}$$

However, considering that the quadrotor is a rigid body symmetric about its *xz* and *yz* plane, and the rotation axes coincide with the principal axes, then all the inertia values between different axes are zero, and the final inertia matrix is composed of only its diagonal.

The matrix form of equation (2.4a) is represented as follows, where *(u, v, w)* and *(p, q, r)* are linear and angular velocities respectively,

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} + m \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ \omega \end{bmatrix} \quad , \tag{2.6}$$

and when equations (2.2) and (2.4a) in the matrix form are combined, and isolating for the accelerations in the body frame, the following is obtained:

$$\begin{aligned} \dot{u} &= rv - q\omega - gsen\theta \\ \dot{v} &= p\omega - ru - gcos\theta + sen\phi \\ \dot{\omega} &= qu - pv - gcos\theta cos\phi + \frac{T}{m} \end{aligned} \quad . \tag{2.7}$$

For simplicity, the drag and lift coefficients were omitted and a drag-free system is considered.

For the moment, using Euler's rotation equation in (2.4b),

$$\begin{bmatrix} Mx \\ My \\ Mz \end{bmatrix} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad , \tag{2.8}$$

where $I_x = I_{xx}, I_y = I_{yy}, I_z = I_{zz}$.

Leading to the angular rate equations (2.9),

$$\begin{aligned} \dot{p} &= (M_x - (I_z - I_y)qr)/I_x \\ \dot{q} &= (M_y - (I_x - I_z)pr)/I_y \quad . \\ \dot{r} &= (M_z - (I_y - I_x)pq)/I_z \end{aligned} \tag{2.9}$$

Because of the *xz* and *yz* symmetry, $I_x \approx I_y$. Which leads to simplifying equation (2.9),

$$\dot{r} = M_z/I_z \quad .$$

2.1.3 Linearized State Space Model

It is often more convenient to consider the linear dynamics around the point of interest rather than the entire nonlinear model when doing control design and analysis.

For a stable hover control simulation, it can be assumed that the UAV has very small ϕ (roll) and θ (pitch) angle changes, and makes only ψ (yaw) motion in the x-y plane. The small angle approximations are $sin(x) \approx x$ and $cos(x) \approx 1$

Having this defined, and since *(u, v, w)* are linear velocities, it is safe to assume that ,

$$\begin{cases} \dot{x} = u \\ \dot{y} = v \\ \dot{z} = \omega \end{cases} \quad \begin{cases} \dot{\phi} = p \\ \dot{\theta} = q \\ \dot{\psi} = r \end{cases} \quad . \tag{2.10}$$

By applying this in equation (2.9):

$$\begin{aligned} \ddot{\phi} &= (M_x - (I_z - I_y)\dot{\theta}\dot{\psi})/I_x \quad , \\ \ddot{\theta} &= (M_y - (I_x - I_z)\dot{\phi}\dot{\psi})/I_y \quad , \\ \ddot{\psi} &= M_z/I_z \quad . \end{aligned} \tag{2.11}$$

It is necessary to solve equation (2.2) in the World frame to obtain the equations of motion. Combining with the acceleration in the inertial frame we get:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = m \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} +^B R_g^{-1} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \quad , \tag{2.12}$$

$$\begin{cases} \ddot{x} = \frac{T}{m}(sen\phi sen\psi + cos\phi cos\psi sen\theta) \\ \ddot{y} = \frac{T}{m}(cos\phi sen\theta sen\psi - sen\phi cos\psi) \\ \ddot{z} = \frac{T}{m}cos\theta cos\phi - g \end{cases} \quad \begin{cases} \ddot{x} = \frac{T}{m}(\phi sen\psi + \theta cos\psi) \\ \ddot{y} = \frac{T}{m}(\theta sen\psi - \phi cos\psi) \\ \ddot{z} = \frac{T}{m} - g \end{cases} \quad . \tag{2.13}$$

The state vector of the quadrotor, which is mapped to the DOF, is as follows:

$$X = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \quad . \tag{2.14}$$

The state vector defines the quadrotor's states, whereas a control input consists of four inputs

$$U = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^T \quad . \tag{2.15}$$

Where U_1 is the thrust, and (U_2, U_3, U_4) represent the roll, pitch and yaw moments respectively,

$$\begin{aligned} U_1 &= T \quad , \\ U_2 &= M_x \quad , \\ U_3 &= M_y \quad , \\ U_4 &= M_z \quad . \end{aligned} \tag{2.16}$$

To conclude, without considering the air resistance and the lift, the dynamic model of the quadrotor can be represented as:

$$F(X,U) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\omega} \\ \dot{\psi} \\ \frac{U_1}{m}(\phi sen\psi + \theta cos\psi) \\ \frac{U_1}{m}(\theta sen\psi - \phi cos\psi) \\ U_1 - g \\ (U_2 - (I_z - I_y)\dot{\theta}\dot{\psi})/I_x \\ (U_3 - (I_x - I_z)\dot{\phi}\dot{\psi})/I_y \\ U_4/I_z \end{bmatrix} \quad . \tag{2.17}$$

2.1.4 Cascade Controller

Cascade control involves the use of two controllers with the output of the first controller providing the setpoint for the second controller [7]. The inner loop controller is nested inside the outer loop one, and independently, the inner and outer loop are each a Feedback controller, since the output signal is being passed to the input of the controller [23]. Such a system can give an improved response to disturbances.

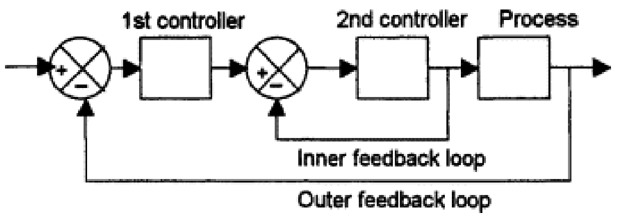


Figure 2.3: Cascade controller.

When controlling a UAV [26], the Outer Loop represents Position controller. The Position controller receives the current and desired attitude, and outputs the desired roll and pitch angles that will remove the error between the two inputs. The same goes for the Altitude controller, that outputs the necessary Z velocity that will remove the error between the UAV's altitude and its target's altitude.

On the other hand, the Inner Loop, or the Attitude and Altitude controller, receives the outputs of the Outer Loop controller as its inputs, and computes the necessary motor commands to achieve the desired inputs.

2.1.5 PID Controller

In a control loop, the controller receives a certain setpoint request, and compares it to a measured feedback. The difference between these two is called the error ε, and the purpose of the controller is to eliminate this error. In a PID controller [1], three constants are set to define the importance given to the Proportional, Derivative, and Integral interpretation of the error, which can be seen in Figure 2.4.

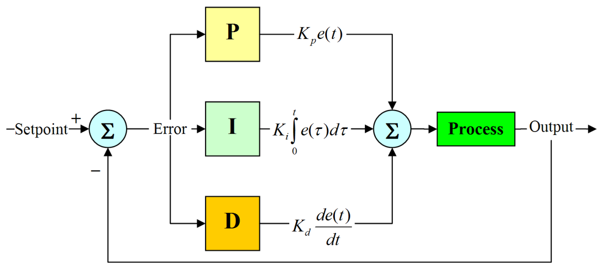


Figure 2.4: PID controller.[2]

Proportional, refers to the error being multiplied directly by a proportion constant K_P. The smaller the value of this constant, the faster the controller stabilizes, but may stabilize while it still has error. The larger it is, the smaller the steady-state error is, but the loop becomes more likely to become unstable. Integral represents the sum of error over time. The larger the value of K_I, the easier it is to reduce a persistent steady-state error. However, this may lead to an overly oscillatory response. Derivative is the rate of change during a given interval, which means, this controller corrects present error in comparison with the previous error. Larger values of K_D, diminish the effect of P and I. In conclusion, to have a good PID controller, there must be an equilibrium between the three constants K_P, K_I and K_D, in order to obtain the desired response. Therefore, tuning of these constants needs to be performed [29] when implementing the PID controller.

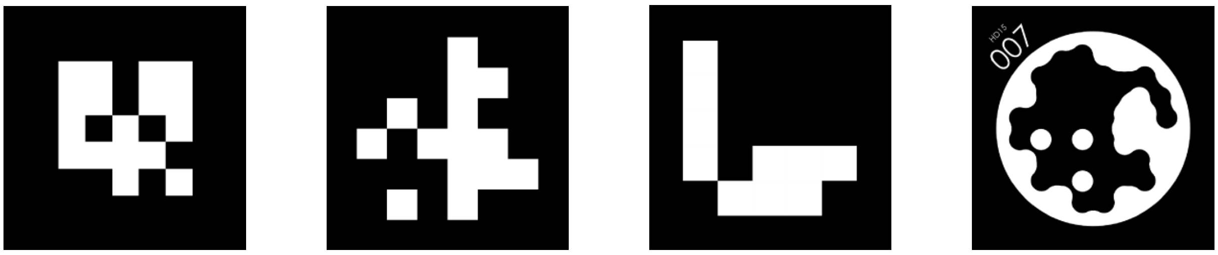
[2]https://en.wikipedia.org/wiki/File:PID-feedback-loop-v1.png

2.2 Fiducial Markers

Fiducial Markers are a type of artificial landmarks of known size and shape used to provide a point of reference or measurement. Fiducial Marker Systems are computer vision algorithms created to identify the fiducial markers in an environment. Applications for these markers range across various areas, such as robotics or medical imaging, but where initially designed to be used in augmented reality. Most markers are black and white due to being more detectable from higher distances and angles.

Most square markers are based on the ARToolkit library, an open-source computer tracking library for creation of augmented reality applications that overlay virtual imagery on the real world. ARTag is based on ARToolKit, but uses a different method to codify the internal id of the marker. AprilTag is an improved version of ARTag. And ArUcO is another more recent combination of AprilTag and ARToolKit.

A study comparing the efficiency of different fiducial markers was conducted by [21], where the three previously mentioned markers were considered, as well as a circular-shaped marker STag, as in Figure 2.5.



(a) ARTag (b) AprilTag (c) ArUcO (d) STag

Figure 2.5: Fiducial Landmarks.

When analysing the performance of these four algorithms regarding the distance to the marker, AprilTag and ArUcO both showed consistency in gradually losing accuracy with higher distances, ARTag on the other hand was affected by the presence of shadows in distances higher than 1.75m. As for performance with different marker orientations, STag showed better accuracy, having ArUcO and ARTag occasionally outperform it, however STag was also less stable in its measurements.

One of the most important factors to consider when choosing an algorithm for a small system is its computational cost. In the evaluation performed by [21], AprilTag proved to be the most computationally heavy out of the four, followed by STag. While ARTag and ArUcO had the least computational needs. To conclude, all but ARTag showed high detection rates, STag had better performance when it comes to position measurents, and AprilTag with orientation measurements. However, ArUcO was the most consistent throughout all performance tests and a close second in both position and orientation.

2.3 Finite State Machine

A Finite State Machine (or Finite State Automata) [18] is defined by a tuple $(\sum, S, s_0, \delta, F)$, where:

- $\sum$ is a finite set of symbols representing events;

- *S* is a finite, non-empty, set of states, representing actions;
- $s_0 \in$ *S* is the initial state;
- δ is the state-transition function: $\delta : S \times \sum \longrightarrow S$;
- *F* is a, possibly empty, set of terminal states, a subset of S;
- *O* is a, possibly empty, set of outputs.

A Finite State Machine keeps track of the current state, and the list of valid state transitions from and to each state. A Deterministic Finite State Machine, is one where there is only one transition for any allowed input. For example, if a state outputs a certain letter 'x', only one transition can possibly be triggered, therefore there can only be one path leading out of a state for a certain output.

Chapter 3

State of the Art

This section of the report will overview the different techniques previously used to autonomously land a UAV on top of a moving target.

3.1 Artificial Landmarks

An artificial landmark is a recognizable artificial feature used for navigation/localization. Vision-based methods often prefer performing localization by placing a set of landmarks in known positions. By using landmarks, the robot can obtain its position with respect to (w.r.t.) the landmark. Therefore simplifying the localization process, and obtaining a better accuracy and stability than it would with natural landmarks. To serve its purpose, the landmark has to be distinguishable from its environment. In this research area, it is common to use landmarks to identify the UGV. This is done by placing the landmark on top of the platform.

Both Baca et al. [3] and Falanga et al. [11] researches were to be implemented in the MBZIRC competition. The landmark used for both was the same, as seen in Figure 3.1(a). In Polvara et al. [32], the choice was very similar, as in Figure 3.1(b).

In Lange et al. [22], the landing pad is identified through the unique radius of each ring, which is represented in Figure 3.1(c). In this research, as well as in others, the landmark is distinguishable from a distance as well as when close, in this particular case, the radius of the rings gradually gets smaller when closer to the center.

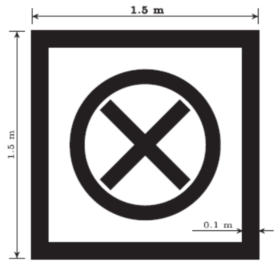
The currently most used type of artificial landmarks are the Fiducial Markers. In Araar et al. [2], this unique landmark is used in a particular way, as in Figure 3.1(d). A set of markers are arranged gradually smaller from the outside to the center of the landing pad, to allow the identification of the landmark both from a great distance, as well as up close. Similarly to Araar et al. [2], de Souza et al. [10] uses a set of larger and smaller ArUcO markers, as in Figure 3.1(e), again to allow the landing pad to be detected from several distances.

As opposed to what was said above regarding the format of the landing pad, Hui et al. [19] uses a white circle with 20cm of radius as a target, as in Figure 3.1(f). To find it, it has in consideration the area and shape, since they are both previously known. This was proved to be very ineffective, since it is an easily misrecognized shape and color in a natural environment.

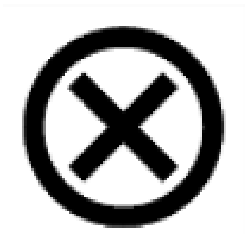
A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other. The image pre-processing required in a CNN is much lower as compared to other classification algorithms. They are formed by several layers, the main ones are the convolutional layer, which uses a kernel or filter that reduces the image into an easier to process form by extracting high-level features; and the pooling layer, which is responsible for reducing the spatial size of the previously convolved feature. The layers on a CNN can have numerous architectures, some will be further mentioned on the next section.

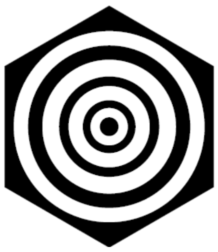
In Cabrera-Ponce and Martinez-Carranza [9], a vertical mexican flag is located near the H-shaped landing pad. The drone first flies towards the flag, then observes the landing pad. The images that are recognized by the CNN are in Figure 3.1(f).

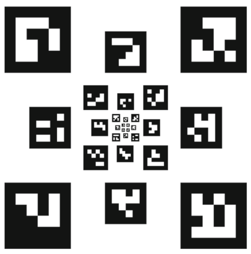
In Lee et al. [24] the marker for landing is a red object, typically a red rectangle on top of the moving ground vehicle.

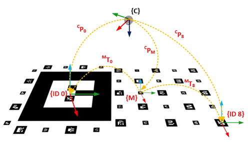


(a) Baca et al. [3] and Falanga et al. [11]

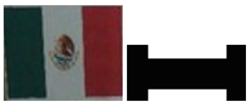

(b) Polvara et al. [32]


(c) Lange et al. [22]


(d) Araar et al. [2]


(e) de Souza et al. [10]

(f) Hui et al. [19]


(g) Cabrera-Ponce and Martinez-Carranza [9]

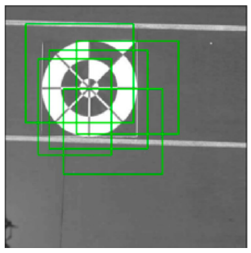

(h) Nguyen et al. [28]

Figure 3.1: Artificial Landmarks.

On all the aforementioned papers, all the details regarding the shape of the artificial landmark were previously known by the UAV. This reduces the errors due to misrecognition and consequent failure. Amongst all the landmarks shown, the less computationally heavy ones to detect are those with Fiducial type of markers. These have the big advantage of being automatically detected by an open-source algorithm. The landmarks presented in Figures 3.1(a), (b), (c) and (f) were chosen due to being geometric figures and symmetric, which provides an equal measurement from every side of the platform. In the first two, by adding more than one geometric figure, the landmark becomes more easily distinguishable.

3.2 Landing Pad Detection

As could be observed in the previous section, all artificial landmarks had a big contrast in color and used only black and/or white. The best way to filter an image with these characteristics is by applying an adaptive threshold. This takes an image as input, and, in a generalized way, outputs a binary image representing the segmentation. For each pixel in the image, a threshold has to be calculated. The threshold is found for each single pixel by interpolating the results of subimages. An alternative is to define a fixed threshold, which in certain cases can result in a faster method. The latter was applied in Lange et al. [22]. Additionally, invariant moments, which are features of an image that are unchanged when under transformations, are then used to perform pattern recognition to identify the landing pad.

The remaining works described in the literature followed the adaptive threshold approach. In Hui et al. [19], this was followed by eroding away the boundaries of the circle with a median filter. Then a Hough transformation is applied, to verify if the observed shape is according to the previously known standards. Similarly, in Baca et al. [3] and in Falanga et al. [11], after performing the adaptive threshold, since the algorithm knows the shapes of the landing pad, it detects the square, followed by the circle and then the cross. Additionally, Falanga et al. [11] uses a Perspective-n-Point approach to estimate the pose of the camera, in relation to the landing pad, given a set of n 3D points in the world and their corresponding projections in the image. In Lee et al. [24] after the RGB raw image is resized, it is passed through a Gaussian filter to reduce noise and transformed to HSV (Hue, Saturation, Value). The transformation is necessary in order to analyse the red marker in any environment. Then, an adaptive threshold is performed to obtain only the region of interest, and consequently a morphological filter is used to reduce noise in a binary image. Finally, the center of mass is obtained from the contour of the marker.

In Araar et al. [2], a marker based kind of landmark was picked. This allowed them to use a visual fiducial system, AprilTag, which creates a target from an ordinary printer, and then computes the precise 3D position, orientation and distance of the tags relative to the camera. This method allows for a unique identification of the landmark, which is less prone to errors. Since the landmarks were of the same type, de Souza et al. [10] and Borowczyk et al. [8] also used an open source tag tracking, AR Track Alvar and AprilTag, respectively. Also, in Borowczyk et al. [8], to avoid target loss, the gimbaled camera centers the image onto the AprilTag as soon as visual detection is achieved, this is done by implementing a controller for the Pitch θ and Yaw ψ angles of the Gimbal. When the Micro Aerial Vehicle (MAV) is close to the landing pad yet too far to cut off the motor, the gimbaled camera cannot perceive the whole AprilTag, so a fixed, wide-angle camera is used to provide measurements

YOLO is an object detection algorithm that requires only one forward propagation pass through the CNN to make predictions. This means that a single neural network is applied to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. YOLO v2 is a variant of YOLO, where the bounding boxes are generated based on the trained ground truth. In Nguyen et al. [28], the latter is used combined with a lightDenseNet, to allow the marker to be detected from a long and close distance with real-time speed. The DenseNet architecture is used to solve the problem of vanishing information in deep CNNs by adding simple connectivity pattern to ensure the maximum flow of information between layers. In each dense block, the feature maps of the current layer are concatenated with feature maps from previous layers as an additional channel. Since detecting a simple marker with a background is a small dataset, only two dense blocks are necessary. Then the marker used in Nguyen et al. [28] and the five bounding boxes to detect it can be seen in Figure 3.1(h).

Single Shot Detector (SSD) is a CNN that detects multiples objects through predictions of bounding boxes around them with the capability to learn up to twenty classes of objects. In Cabrera-Ponce and Martinez-Carranza [9], an SSD with 7 Convolutional and Pooling layers is used for the detection of the landmark. The pictures are classified in comparison to the original images. With the bounding boxes, a ratio of the intersected area over the union area is used to perform the proper detection. In the last layer, a Non-maximum Suppression, a post processing step that selects the best bounding box for an object by "suppressing" all others, is applied, to clear the unnecessary bounding boxes, and remove duplicate predictions.

During the landmark detection process, in Polvara et al. [32], only the xy-plane is considered, for simplicity. Therefore this phase is performed at a fixed altitude. A CNN receives four grey-scale images, they are then processed by three convolutional layers, for feature extraction, and two fully connected layers.

Combining the variety of artificial landmarks analysed in the previous Section, fiducial landmarks are chosen to be placed on top of the UGV and serve as a detection pattern due to their uniqueness and fast detection. The comparison done in Section 2.2 also led to conclude the choice of the ArUcO library for the marker detection.

3.3 UAV Control

A High-level controller computes complex high-level reasoning tasks, while the Low-level controller continuously computes low-level control processes. In Falanga et al. [11], a nonlinear control to drive the quadrotor towards the desired trajectory is implemented using a High and Low level controller. The high-level controller takes the difference between the desired and estimated position, velocity, acceleration and jerk as input, and returns the desired collective thrust and body rotations. These rotations are passed as input to the low-level controller, which computes the necessary torques to be applied to the rigid body.

In Araar et al. [2] an EKF filter is used for sensor fusion to estimate the relative pose of the quadrotor w.r.t. the landing pad. This pose is used as the control's feedback to keep tracking the ground vehicle. To regulate the quadrotor's position and heading, 4 PID controllers are designed for the quadrotor's position and heading.

Similarly to the previous paper, a normal Kalman filter is used to combine the data from IMU, GPS and visual measurements in Borowczyk et al. [8]. The Ground Vehicle has a mobile phone that has both a GPS and an IMU, therefore sends the information regarding these two units to the MAV through a wireless link. Two types of controllers are used in this reasearch. First a Proportional Navigation (PN), which uses the fact that two vehicles are on a collision course when their direct Line-of-Sight (LOS) does not change direction as the range closes. This controller provides acceleration commands perpendicular and parallel to the LOS vector, however the acceleration in the z axis is disregarded due to this position not changing. When the MAV is close to the Ground Vehicle, a more precise controller is needed, so a Proportional Derivative (PD) controller is used. By using two controllers, it is necessary to switch between them without disturbing the tracking of the AprilTag, so this is performed when the output of the PD and PN controllers are similar and before the visual target acquisition.

In Hui et al. [19], when the quadrotor is hovering, the angles roll, pitch and yaw, respectively (ϕ, θ, ψ) tend to approach zero. The nonlinear dynamics hence can be linearized, and the 6-Degrees of Freedom(DOF) pose of the UAV can be independently controlled. With the linearization and the decoupling, autonomous stabilization flights are achieved by three independent PID controllers, one for each angle.

In Lange et al. [22], the distance measurements performed with the camera need to be corrected for the current angles of the UAV. This is necessary because the camera is fixed on the frame of the UAV and it is not tilt-compensated. Therefore, the control starts by receiving the values of (x, y, z) and corrects these estimates using the current pitch and roll angles. The corrected position estimates are then used as input for a PID controller that generates the necessary motion commands to keep the UAV steady above the center of the landing pad.

An Underactuated system is one that cannot be commanded to follow arbitrary trajectories in configuration space. This is due to the system having a smaller number of actuators than degrees of freedom. A fully actuated system is one where this number is equal. In Ghamry et al. [16], an Under-actuated subsystem is used, composed of a Linear Quadratic Regulator (LQR), followed by a Sliding Mode Control (SMC). The states of the UAV are estimated with the aid of a 24 camera OptiTrack motion capture system. A LQR is a method that provides optimally controlled feedback gains to enable the stability and high performance of close-loop systems. A SMC is a non-linear control method that alters the dynamics of a non-linear system by applying a discontinuous control signal. Forcing it to ”slide” along a cross section of the system's normal behaviour. In Ghamry et al. [16], the LQR is used to to obtain the desired position in x and y, and the SMC to converge the actual values of the Euler angles ϕ and θ to their desired values obtained from the LQR.

A Fully-actuated subsystem, contained of a SMC, is also used with the objective of minimizing the error in the altitude and yaw angle, to lead them towards their desired values.

Baca et al. [3] uses a Pixhawk flight controller to obtain the data regarding the UAVs position, velocity and orientation in the world frame. The positions obtained from this controller suffer heavy drifts in long periods of time, therefore it is corrected by differential RTK GPS using a LKF for fusion, if GPS is available. The vertical position is estimated from the PixHawk aswell, but it is corrected by not only the differential RTK GPS, but also the TeraRanger rangefinder, and the landmark detector algorithm.

Model Predictive Control (MPC) is used to control a process under a set of constraints. In Baca et al. [3], the MPC tracker is used since its predictive nature provides trajectory tracking optimizing actions over the future, making it ideal for tracking moving targets. The MPC is used in two modes. A simple low-level positioning mode, used mainly for short-distance position changes. And a high-level trajectory planning mode that uses a precomputed path plan.

In Lee et al. [24], the flight control system of the quadrotor has an attitude controller, a velocity controller and a position profile. The velocity commands are calculated using the relative distance between the current position and the target position. To calculate the x and y position errors, the local image positions that were detected through the camera are used. So the destination position can be represented with the result from the vision detection algorithm. Therefore, the control system needs to be connected to the flight control system to receive the current position and then update it; and to the image processing system to get information regarding the target.

Fuzzy Logic Control (FLC) is a controller whose output is determined by the fuzzy logic that exists between the minimum and maximum damping states. Fuzzy logic works by executing rules that correlate the controller inputs with the desired outputs. There are three steps that characterize this controller. These steps are the fuzzification of the controller inputs, the execution of the rules of the controller, and the defuzzification of the output to be implemented by the controller. In de Souza et al. [10] this type of controller is used to drive the UAV towards the target. Receives as inputs the horizontal distance between the UAV and the landing spot in the X and Y axis, *dX* and *dY* respectively. The output variables adopted for the controller are linear velocity commands in the UAV frame (v_x, v_y, v_z). The strategy imposed by this controller is to reduce the variables dX and dY until the vehicle is near the landing pad, and then the UAV begins the descent process, actuating in v_z.

An Artificial Neural Network (ANN) was developed with the objective of learning the behaviour of the FLC. By using the FLC to train the ANN, the steps required to classify the network become unnecessary. The ANN works by receiving the position of the marker, recognizing its position error and returning 3D velocity commands to the aircraft.

Reinforcement Learning is used when an agent learns which actions to take in a given state, by taking actions and receiving rewards. Q-learning does not evaluate the value of being in each state independently, but each state-action pair. This is denoted by $Q(s, a)$, where *s* is the state, and *a* the action. Q-learning introduces the Q-table, which calculates the expected future rewards for each action in each state. Deep Q-Learning (DQL), introduced the use of Neural Networks with Deep Q-Networks (DQN), instead of the Q-table, when the number of state action pairs was too big and made the Q-table too computationally heavy. First, a state is fed to the Neural Network. Then, since all the past experience is stored, the current prediction is compared to the previous ones. Neural networks work by updating their weights, so a Loss function is created by taking the sum of the squared differences of the Q-values and their targets. Finally, to choose which action is the best, the Q-values are passed through a softmax function. This is a layer placed right before the output of the CNN that transforms each real value that it receives into a values between 0 and 1 so it can be interpreted as a probability.

In Polvara et al. [32], the vector of parameters θ of the DQN needs to be adjusted for the UAV to align with the landmark. So, as mentioned before, a Loss function is defined as $L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)}[(Y_i - Q(s, a; \theta_i))^2]$. Where *D* is the dataset of experiences, *E* is used to uniformly sample a dataset at each iteration *i*. The network $Q(s, a; \theta_i)$ is used to estimate actions at runtime, and Y_i is the target. To perform the vertical descent, an agent has to take the right action in order to progress through a sequence of *N* states. However, it is extremely difficult to obtain positive reward because the target-zone is so small. So, there was a need to partition the experiences, dividing the dataset in D^+, D^- and $\tilde{D}$ for positive, negative and neutral experiences respectively. To avoid overestimation, the target is estimated through double DQNs. The final Touchdown manoeuvre, in Polvara et al. [32], is performed with the aid of a PID controller, as is shown in Section 3.4.

3.4 Landing Strategies

State Machines are a practical way of simulating sequential logic. In the particular case of the UAV landing on a Moving Platform, several constraints need to be taken into account so the landing is successful. Some researches followed this approach when defining their landing procedure.

One of the major applications of UAV-UGV cooperative systems is the monitorization, detection and fighting of forest fires. In Ghamry et al. [16], the autonomous system is used for this purpose, where the UGV acts as the leader, that commands a certain operation on the UAV, and then the latter follows the UGV throughout the operation. Since the UAV always knows where the UGV will be, the whole execution is facilitated by the UGV using a pure-pursuit controller to plan its path, which means that the UAV knows at all times the path that the UGV is taking, therefore simplifying the whole tracking and landing operation.

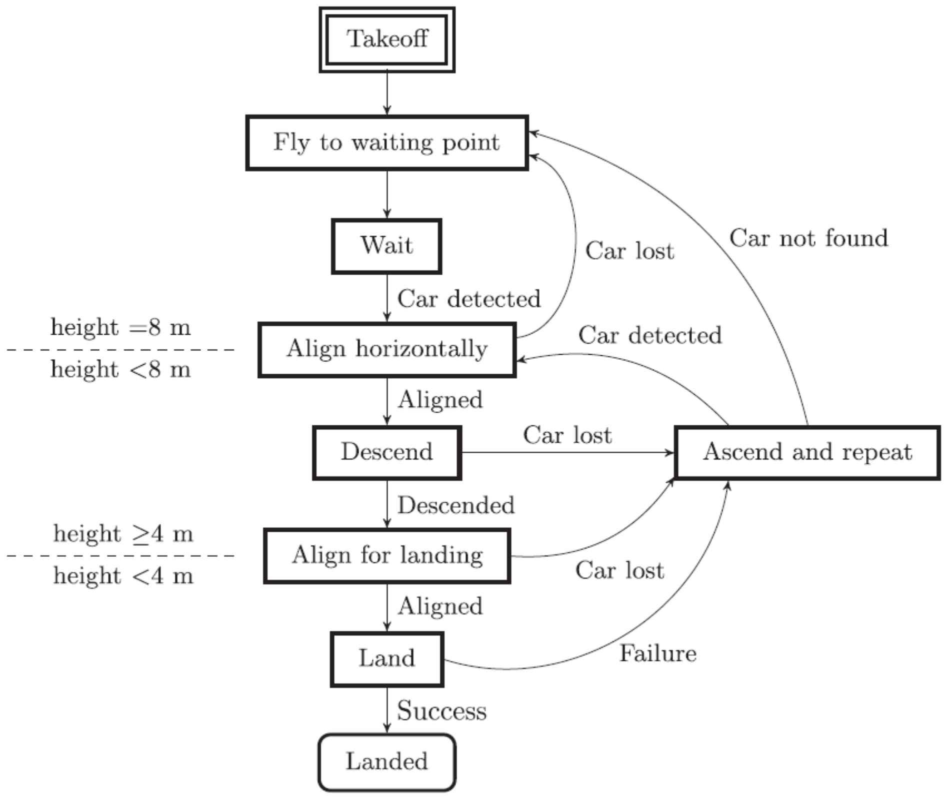


Figure 3.2: Baca et al. [3] State Machine.

In Baca et al. [3], the need to simplify the landing operation due to the competition in place (MBZIRC

2017, which they won first place), led to the UAV waiting for the UGV in the center of a previously known track which the UGV follows in a repetitive way during its movement. Therefore, the following steps, which can be observed in the diagram in Figure 3.2, are followed to perform the landing. Once the quadrotor takes off, it moves towards the center of the track and waits for the ground vehicle to pass its line of sight. Once it does, predicting its movement is facilitated due to knowing the shape of the track. Once the UAV is properly aligned with the car, descend state is activated and the height decreases to 4m, since this is the lowest height at which it is still possible to follow the car for this particular UAV and camera. Subsequently, the next step is to land, therefore it is required that the UAV is horizontally aligned within 0.3m of the center of the landmark. Additionally, it is mandatory that the current trajectory of the UGV is straight. Afterwards an aggressive descent is performed once the UAV is at a 1.5m distance of the landing platform. For all the steps in the state machine, if any parameter is not met, all consequent steps are aborted and the UAV ascends and repeats them.

The team that won second place on the MBZIRC 2017 challenge was Falanga et al. [11], their approach to this topic was different, where the path taken is not known neither is it constant.

In Araar et al. [2], regarding the automatic landing, the diagram that shows the steps taken can be seen in Figure 3.3.

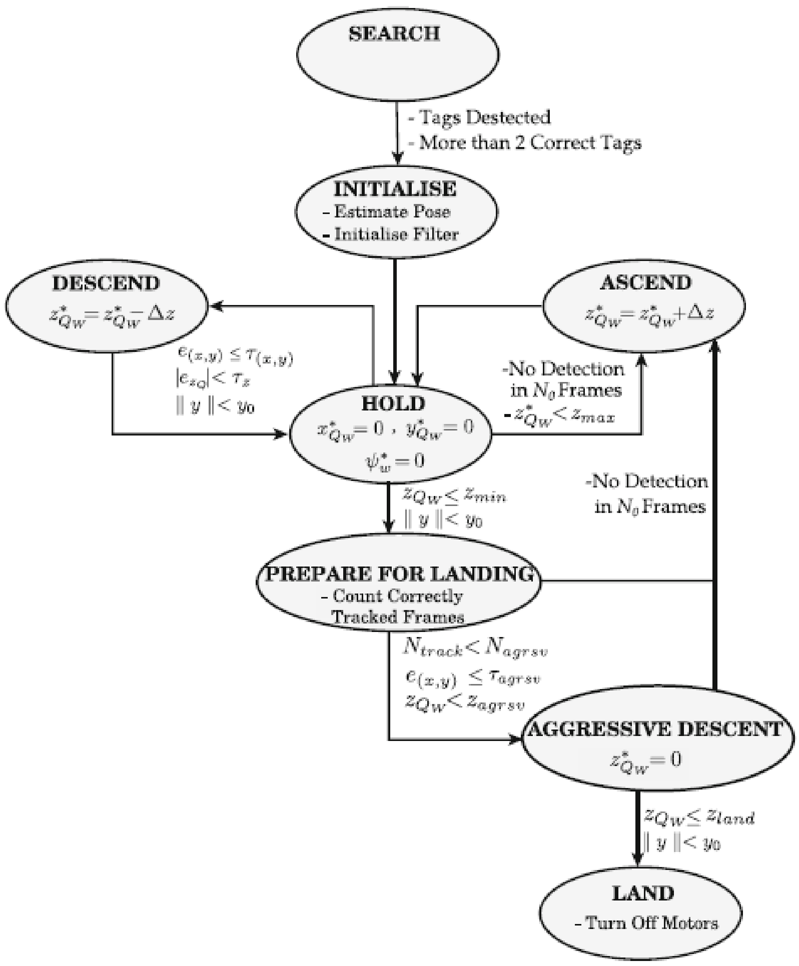


Figure 3.3: Araar et al. [2] State Machine.

First, when at least two markers in the landing pad have passed the check detection algorithm, the state of the filter is initialised and the quadrotor starts tracking the ground vehicle. Afterwards, the quadrotor navigates to the centre of the landing pad following a sequence of vertical waypoints. It keeps hold of its current waypoint and moves to the next one if the vehicle is within a cylinder, of radius and height, around the current waypoint. In order to reduce the impact of the ground effect, when a certain minimum height is reached, it stops descending and lands if the tracking has been successful in a consecutive sequence of frames. To perform the landing, the vehicle executes an aggressive descent by turning off the motors. In a case where any of the above conditions fail, the UAV ascends and attempts to repeat the process.

A Petri net, similar to a state machine, is also a five-tuple (P, T, A, w, x). Where P is the finite set of places; T is the finite set of transitions; A is the set of arcs from/to places to/from transitions; w is the weight function of the arcs; and x is a marking of the set of places.

Ghommam and Saad [17] follows three stages in order to perform the UAV's landing. These are the search, homing and landing phase represented in the Petri Net in Figure 3.4.

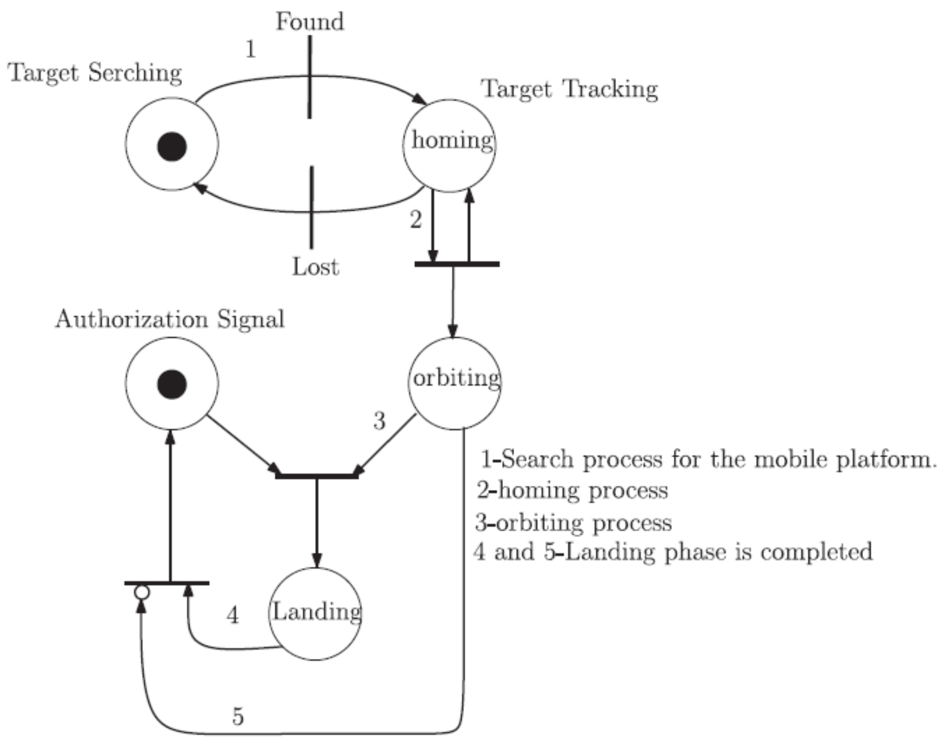


Figure 3.4: Ghommam and Saad [17] State Machine.

During the search phase, the UGV is not in the camera's line of sight yet, therefore GPS is required. Afterwards, since the distance between the two robots has become smaller, GPS becomes inaccurate, hence the camera is used to provide relative pose measurements. In this homing phase, the goal is to drive the quadrotor to match, with a certain accuracy, the position of the virtual target point directly above the UGV. This virtual target is defined by a sphere with known radius *l* and whose center coincides with the mobile platform's center of gravity (CoG). The quadrotor homes on a virtual target point $p_t(t) =$

$[p_{tx}, p_{ty}, p_{tz}]^T$ that moves on the sphere. Then, the landing guidance control phase is activated after a permission signal has been sent from the mobile platform to the quadrotor, and the quadrotor has matched with the position and velocity of the UGV. Otherwise it will keep orbiting the UGV in order to track it.

In Polvara et al. [32], a hierarchy of DQNs for addressing the different phases is used. The networks are able to automatically call each other in specific portions of the state space, reducing the complexity of the task. The overall landing problem is divided in three stages: landmark detection, descend manoeuvre, and touchdown. The latter corresponds to gradually reducing the motors power in the last few centimeters of landing.

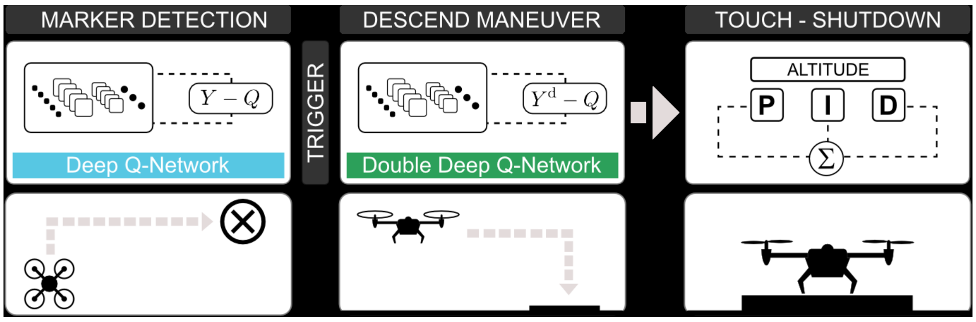


Figure 3.5: Polvara et al. [32] State Machine.

As in Figure 3.5, the first DQN, responsible for performing marker detection, is trained to receive a positive reward when the trigger action was enabled within a target area. A negative reward was instead given if the trigger was activated outside the target area. The second network, responsible for the vertical descent, is trained following the same concept. Only once the two networks have been trained is possible to assemble the state-machine pipeline.

3.5 Detection without Landmarks

Deep Deterministic Policy Gradients (DDPG) is a policy-based deep reinforcement learning algorithm designed to work with both continuous state and action spaces. The key to make the Loss function converge, is to use a replay buffer, as in Polvara et al. [32], and to have a separate network to calculate the target y_t. Therefore they end up with a DDPG with two neural networks to approximate a greedy deterministic policy and the *Q* function. As previously stated, in reinforcement learning, an agent interacts with an environment and tries to maximize the accumulated reward in each time step. In Rodriguez-Ramos et al. [33], the communication channel between the agent and the environment is ROS, this communication plan is represented in Figure 3.6. The position and velocity of both the UAV and UGV are ground truth data. In the training phase, this data is obtained noiseless directly from the Gazebo simulator. Whereas in the testing phase, noise is added. In the real flight testing phase, the UAV and Mobile Platform position and velocity information is solely relied on and provided by an OptiTrack motion capture system.

Reward shaping is a method for engineering a reward function in order to provide more frequent

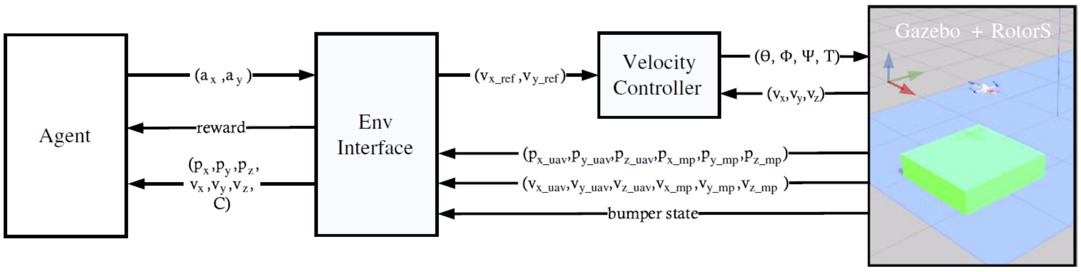

Agent
(a_x, a_y)
reward
$(p_x, p_y, p_z, v_x, v_y, v_z, C)$
Env Interface
(v_{x_ref}, v_{y_ref})
Velocity Controller
(Θ, Φ, Ψ, T)
(v_x, v_y, v_z)
$(p_{x\_uav}, p_{y\_uav}, p_{z\_uav}, p_{x\_mp}, p_{y\_mp}, p_{z\_mp})$
$(v_{x\_uav}, v_{y\_uav}, v_{z\_uav}, v_{x\_mp}, v_{y\_mp}, v_{z\_mp})$
bumper state
Gazebo + RotorS

Figure 3.6: Diagram of Rodriguez-Ramos et al. [33] reinforcement learning simulation framework.

feedback on appropriate behaviors. This is necessary in large domains, where reinforcement signals may be few and far between. In Rodriguez-Ramos et al. [33], where the agent is meant to generate continuous control actions, the reward is designed so that it rewards smooth actions with respect to time:

$$shaping_t = -100\sqrt{p_x^2 + p_y^2} - 10\sqrt{v_x^2 + v_y^2} - \sqrt{a_x^2 + a_y^2} + 10C(1 - |a_x|) + 10C(1 - |a_y|) \quad ,$$

and $r = shaping_t - shaping_{t-1}$, where p_x and p_y are the position of the UAV w.r.t. the Moving Platform; v_x and v_y are the velocities in the same condition; a_x and a_y are the action space, represent the reference velocities, and each variable is weighted by a different coefficient depending on its level of importance. The C coefficient rewards the agent as soon as the UAV lands on the UGV and the velocity references are decreased to their absolute minimum. Using this shaping method allows speeding up the reinforcement learning in general, and increases the speed of convergence.

In Ghommam and Saad [17], to locate the UGV, a sphere with known radius and whose center coincides with the Center of Gravity (CoG) of the mobile platform, a ”virtual target”, is used to drive the UAV towards it. To perform this, combines GPS with vision-based navigation. When using vision-based navigation, the expressions for measurements are obtained by performing feature point extraction techniques. Ghommam and Saad [17] considers linear and small velocities on the UGV, and that the UAV never loses track of the target.

In Yang et al. [40], the UAV needs to land in an unknown environment. It starts by creating a 3D point cloud map of the environment by using visual SLAM. Then, a two dimensional grid map is set up based on the previous point cloud. The height of each grid, in the grid map, is computed by projecting the map points of the graph into the corresponding grids. Then, a region segmentation is performed to smooth the height of the grid map, based on the mean values. To choose the desired landing area, it computes the area farthest from an object, and lands. This approach is very different from the others shown since it is performed in an unknown area. All the altitude measurements are performed combining triangulation of the motion of the camera and the atmospheric pressure measured by a barometer sensor. The transformations between images are performed initially using homography and the fundamental matrix.

In Backman et al. [4], an assistive system for a pilot to land a UAV is designed. The pilot provides target linear XYZ velocities for the on-board flight to follow. The approach is summarised in Figure 3.7, it consists of learning a compressed latent representation of the environment to perceive the location and structure of the landing platforms; and learning a policy network to provide control inputs to assist the pilot in successful landing. The purpose of latent space learning, i.e. a compressed representation of data, is to reduce the time required for the policy network to converge.

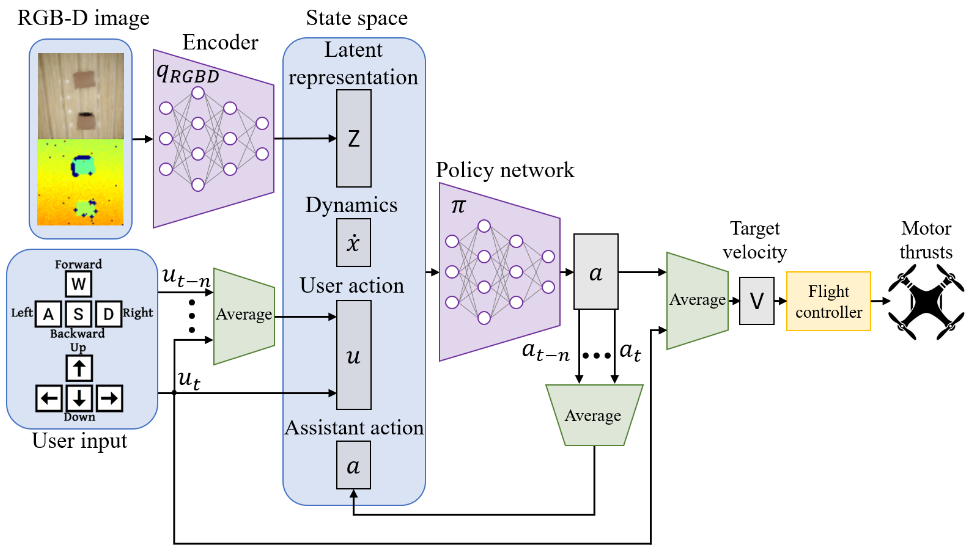


Figure 3.7: System architecture of Backman et al. [4] to assist pilots in landing.

A variational autoencoder is an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process. A Cross-modal Variational Auto-encoder (CM-VAE), trains the latent vector from multiple data sources. In Backman et al. [4] this is used to reconstruct the RGB images. Receives a single input containing the RBG and depth parameter $x_{RGBD} = [x_{RGB}, x_D]$, and outputs two separate parameters in the form of a depth map, and a relative pose of the closest visible landing pad. This is performed with an encoder q_{RGBD}, which is an 8-layer ResNet, i.e. a Neural Network with connections that allow skipping layers, and it is used to obtain the Latent representation Z. Then, two decoders are used to divide Z into the two previously mentioned outputs. The data used to train this network is generated by AirSim.

A Markov Chain is a stochastic process with discrete state space which satisfies the Markov property, i.e. the evolution of the process only depends on the present state. By adding transition probabilities, rewards associated to each state action pair, and a cost/performance function, a Markov Decision Process (MDP) is obtained. A Partially Observable MDP (POMDP) is a generalization of the MDP where the agent cannot directly observe the underlying state. In Backman et al. [4] this is applied by treating the user's landing goal as a hidden state that is only known by the user and must be inferred by the agent. To obtain the closest optimal policy, as was done before in Polvara et al. [32], an actor-critic method with double Q-learning is applied.

3.6 Hardware

In Table 3.1, the Hardware used in each research analysed is shown, namely the UAV and its parts.

Table 3.1: Hardware

Ref.	Drone	Flight controller	Camera	Extra Feature	Distance/ Range Sensor
[22]	Hummingbird	X	Axis 207MW wifi camera	-	-
[19]	DJI F450	ARM Cortex-M4	down looking camera	wide-angle lens	Sonar
[3]	DJI F550 hexacopter	PixHawk	MatrixVision mvBlueFOX	Magnetic legs, SuperFisheye lens	TeraRanger Rangefinder
[16]	Qball-X4	HiQ and Gumstix	-	24 OptiTrack cameras	-
[2]	Parrot AR Drone 2.0	-	integrated camera	“Bluefox” USB camera	-
[11]	DJI F450		2 MatrixVision mvBlueFOX	-	TeraRanger One
[24]	DJI S500	FCC	Cameleon3 USB camera	Fish-eye	-
[33]	Parrot Bebop	X	RGB	Pressure sensor (UGV)	
[40]	M100 UAV	X	ZENMUSE X3	-	Barometer
[32]	Parrot AR Drone 2.0	X	down looking camera	-	-
[10]	PX4 IRIS	Pixhawk	X	-	-
[28]	DJI Phantom 4	X	RGB camera	Gimbal	-
[25]	Tarot T810 hexacopter	DJI NAZA-M V2	Logitech C920 HD	Gimbal	Astech LDS-30A
[8]	DJI Matrice 100	DJI Guidance V2	(1) MatrixVision mvBlueFOX, (2) Zenmuse X3	Sunex DSL224D lens(1), Gimbal(2)	-
[9]	Bebop 2	X	RGB camera	-	-
[4]	X	X	RGB-D	-	-

3.7 Conclusion

To help visualize and divide the analysis of the papers in question, a table was created. This table contains key methods that were implemented by the different approaches and divides the papers according to the years they were published.

Table 3.2: State of art Conclusion

Years	Artificial Landmarks	Adaptive/Fixed Threshold	Neural Networks	Deep Q-learning	PID Controllers
2008	[22]	[22]	-	-	[22]
2013	[19]	[19]	-	-	[19]
2016	[24]	[24]	-	-	-
2017	[2] [11] [8]	[11]	-	-	[2] [8](PD)
2018	[32] [28] [25]	[28]	[5] [32] [28] [25]	[32] [25]	[32]
2019	[3] [10]	[3]	[33] [10]	[33]	-
2020	[9] [4]	-	[9]	[4]	[4]

Chapter 4

Implementation

The goal of this project is to achieve a fully autonomous UAV that is able to identify, locate, and land on a moving UGV.

In this chapter, a description of the developed approach is presented. Section 4.1 starts with the characteristics of the simulated world built in Gazebo and its elements, and then specifies the hardware used, such as the UAV and the UGV, and some of their components. Section 4.2 starts by describing the reference frames that will be analysed for our approach, then the forces applied on the drone are analysed, and the consequent system of equations that describe the motion of the UAV. Finally, the implemented controller and the inner autopilot are thoroughly outlined. Section 4.3 contains a detailed explanation of the vision detection algorithm and the transforms necessary for this implementation. In Section 4.4 the used ROS topics, communication protocol, and then state machine package and its usability are presented. Finally, Section 4.5 contains an thorough explanation of the files that contain the implementation and the purpose of each one.

4.1 System Setup

In this Section, the UAV and UGV provided for this research will be described along with their most relevant parts, as well as the simulated versions.

4.1.1 UAV

The quadrotor provided for this project is shown in Figure 4.1. The UAV's frame is DJI F450, which weights 1.2 kg with a payload capability up to 0.3 kg.

A flight controller board is required to allow basic flight capability, hence the PixHawk 2.1 Autopilot is used along with the Ardupilot, which is an open source autopilot system for copters. The open-source firmware and its well-documented interface allows us to connect it to a high-level onboard computer Jetson TX2. This computer is equipped with Ubuntu 16.04 and has all the ROS packaged necessary to run and document everything onboard. An off-board computer is also necessary during flights to interact,

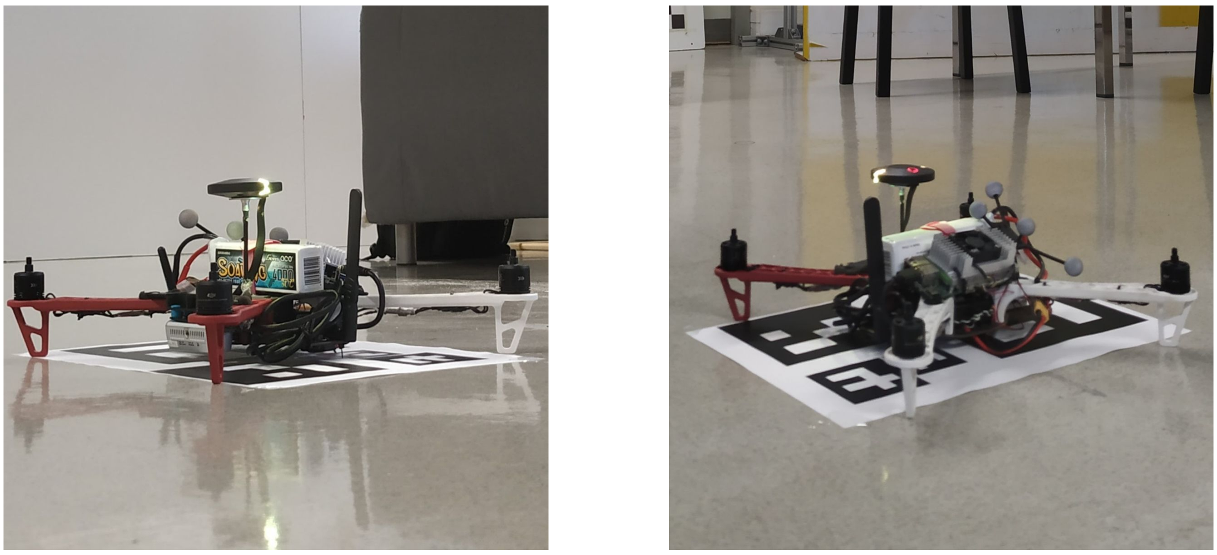

(a) Front view. (b) Back view.

Figure 4.1: Quadrotor.

through Secure Shell (SSH), a communication protocol between remote entities, with the companion computer TX2.

The PixHawk contains sensors such as gyroscopes, accelerometers, an atmospheric pressure sensor, a magnetometer, and GPS, and it produces a single position, velocity, and orientation estimates of the UAV in the world frame by their measurements.

The quadrotor is also equipped with an Intel Realsense depth Camera. This camera has an optimal range of up to 10m, a wide field of view, and a global shutter, meaning that all pixels of an image are exposed simultaneously, allowing it to capture fast moving objects, making it ideal for robotic navigation and object recognition.

In Figure 4.1(a), the compass, battery and camera of the drone are visible, whereas in Figure 4.1(b) the computer and its ventilation is present, note that the rotors are not on these images.

4.1.2 UGV

The UGV provided for this research is Jackal as shown in Figure 4.2. This is a small, fast, entry-level field robotics research platform. The Jackal is fully integrated, compact and waterproof. It has an integrated PC with an Nvidia Jetson running Ubuntu 16.04 LTS and ROS Kinetic installed. Also contains a Velodyne, GPS and IMU. A platform was installed on top of the robot, as can be seen in the figure, to serve as a landing pad for the drone.

4.1.3 Simulation

Testing an incomplete algorithm directly in a robot is time consuming and can lead to harming the robot. Therefore it is recommended to first perform all the necessary testing in simulation, and only after implement it in a real scenario.

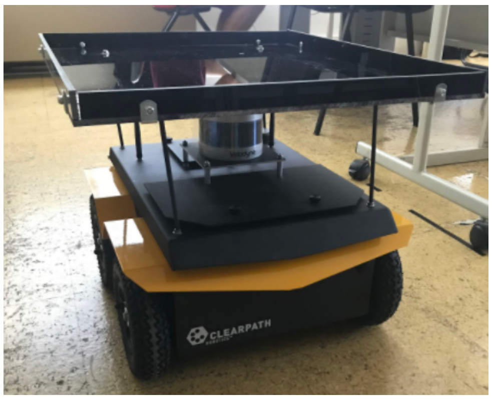
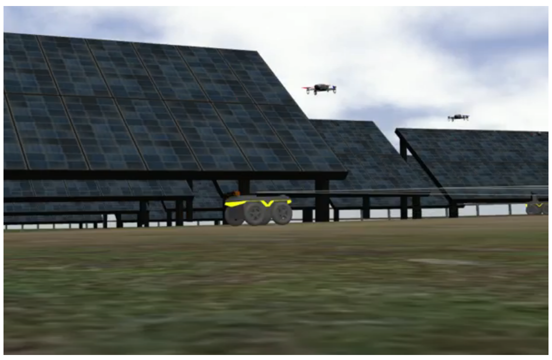


Figure 4.2: UGV - Jackal.

Gazebo is a well-known official robotics 3D simulator that allows to accurately simulate a variety of robots in a complex realistic environment. However, it currently does not support Ardupilot, autopilot integrated in the real UAV. The SITL(software in the loop) simulator, allows building the autopilot with a C++ compiler, simulating the behavior of Ardupilot without any special hardware. Therefore, through an added plugin, Gazebo is used as an external simulator for Ardupilot. The most common copter used to simulate this autopilot in Gazebo is IRIS.

For the results in simulation to be reliable, it is of high importance that the simulated world is as close to the real experiment as possible. Thus the chosen drone was the IRIS Copter, this drone contains the same extras as the real drone, namely the camera and the autopilot, and even though the frame is different, the two frames are relatively the same size and represent just the physical structure, therefore do not alter the viability of the simulation.

The simulated environment that will be used on this project can be observed in Figure 4.3. Figure (a) demonstrates the simulation with the solar panel farm, Figure (b) contains the Iris Copter and the Jackal without the solar panel farm, to improve the efficiency of the simulator.


(a) Solar Panel Farm.

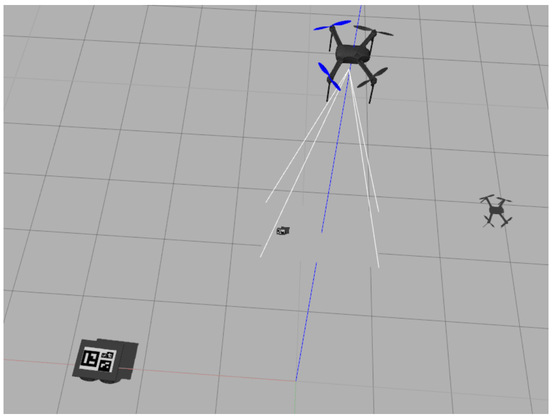

(b) Iris.

Figure 4.3: Gazebo simulation environment.

4.2 Quadrotor Controller

4.2.1 Quadrotor Controller Design

A Cascaded Controller is a common choice when it comes to UAV controllers due to its effectiveness against disturbances [12]. By inserting an inner-loop, the undesirable effects of the disturbances are rejected before entering the outer-loop, therefore the inner-loop needs to have a faster response, run at a higher frequency, than the outer-loop, so that it can cope with the disturbances.

Three independent proportional integral (PI) controllers, on the outer-loop controller, were used to control the three basic movements that define the positioning of the quadrotor.

Figure 4.4 illustrates the general structure of the cascaded controller. The distance measured by the camera from the ArUcO markers is fed to the position controller, the target angles are then computed for each axis according to the appropriate PI constants for each specific case, which will be further outlined in Section 5.2.

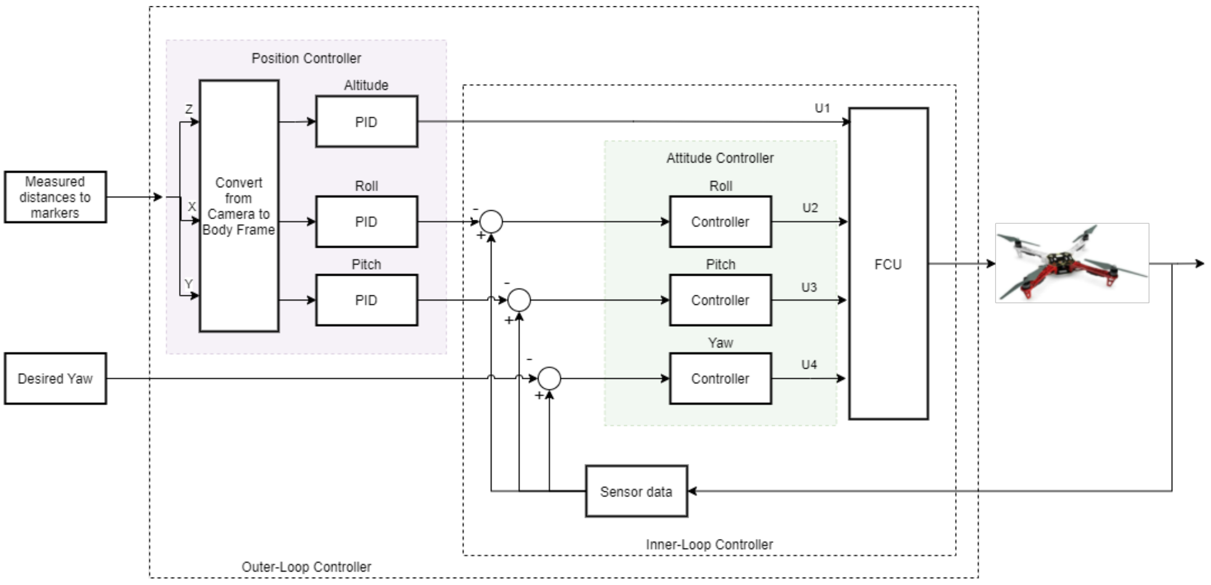


Figure 4.4: Inner and Outer-Loop UAV Controller.

As mentioned before, Roll, Pitch and Yaw are the drone's rotation angles about its X, Y and Z axis respectively. Since a UAV is symmetrical about its Z axis, has no ”front” or ”back” and the camera is pointing downward, it is not necessary to change the UAV's yaw in order to lead it to its target. Additionally, for this project in particular, any landing orientation is acceptable, therefore only the Roll and Pitch angles require controlling.

When a certain distance is measured, the values are automatically converted to the camera frame. However, this frame is different from the body frame, as could be seen in Figure 2.1. Hence it is necessary to perform a transform from the Camera Frame to the Body Frame, this transformation will be further explained in Section 4.3.

These two controls as well as the movement regarding the Altitude of the UAV constitute the Position Controller, whose output is considered as velocity, which is connected as input to the Attitude loop, generating the control commands for the motors.

The Autopilot's part of the controller, the Inner-Loop Controller, in Figure 4.5 is made of an angle controller, which takes in the target angle and the actual angle, and converts the error into a desired rotation rate. Then, a rate controller follows and a PID converts the previous error into a high level motor request, as represented in Equation 2.16. These high level motor commands are then converted into individual motor outputs by the Flight Control Unit (FCU), a small computer that connects the various parts of the multicopter together and is responsible for running the algorithms and calculations that keep the multicopter flying correctly.

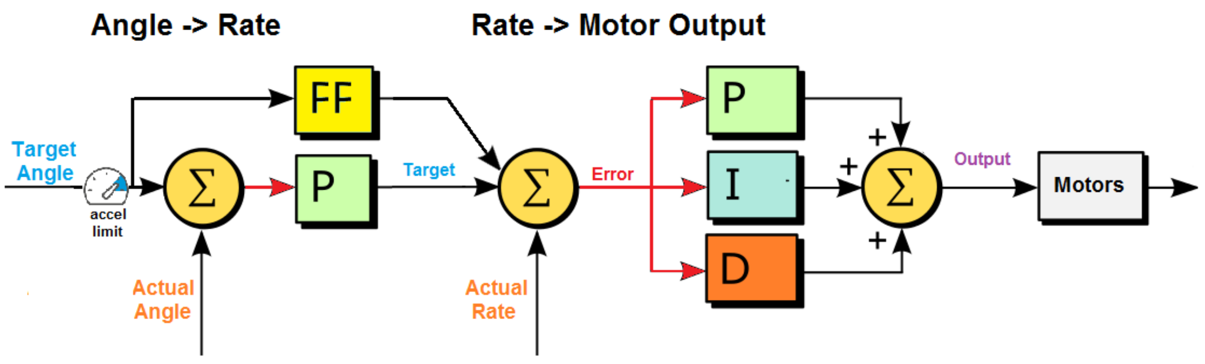


Figure 4.5: Ardupilot Inner-loop UAV Controller [38].

Both the Inner and the Outer Loop controllers need the information regarding the actual current state of the UAV in order to measure the errors and perform the necessary correction, this is the Sensor Data. This data comes from rate gyroscopes, accelerometer, compass, GPS, airspeed and barometric pressure measures; and by using an Extended Kalman Filter (EKF), the vehicle's position, velocity and angular orientation are estimated.

Using an EKF to fuse measurements from several sensors is very important since it allows the filtering of measurements that would lead to errors and therefore make the vehicle more prone to faults. Ardupilot's current best option for sensor fusion is EKF3, which provides more stable and accurate filtering than its previous version, EKF2. This is due to this message containing, for each sensor the difference between the value predicted using the IMU data before corrections are applied, and the value measured by the sensor. Thus EKF3 is the sensor fusing method used in this implementation to feed the curent state of the UAV to the controller.

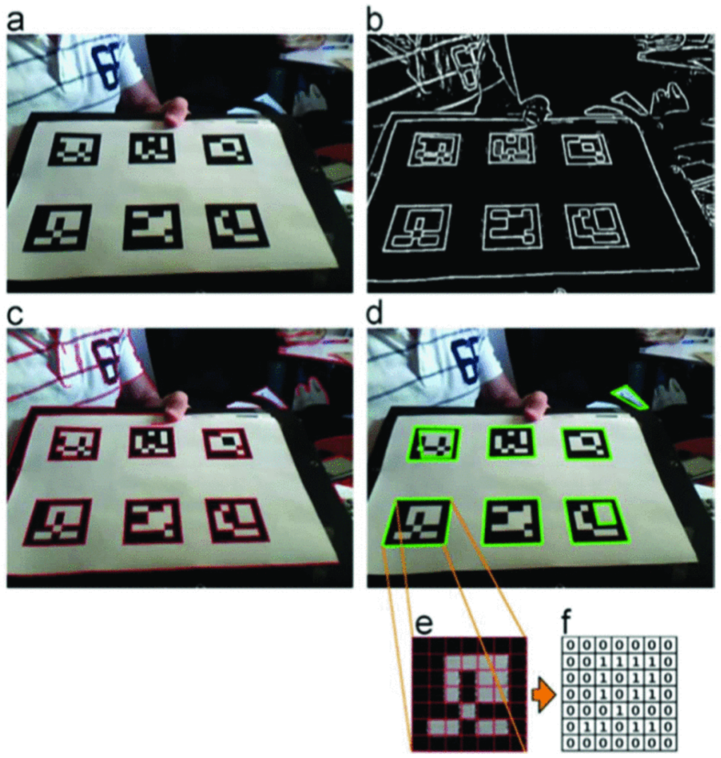
4.3 Vision-based UGV detection

After analysing the variety of artificial landmarks referred in 3.1, fiducial landmarks were chosen to detect the UGV due to their uniqueness and fast detection. After reviewing the comparison done by [21] analysed in Section 2.2 and having in consideration that the ArUcO library is the most recent out of the ARToolkit library variations, the ArUcO library was chosen for the implementation presented in this thesis.

Originally, the ArUcO library was created by [14] to provide a 3D estimation of the position of the camera towards each single marker. This library is based on OpenCV [34] [15]. The square markers comprise a wide black border with an inner binary matrix that can go from 4x4 up to 7x7, and each one having up to 1000 different markers. In this implementation, the 4x4 bit dictionary with up to 1000 markers was used.

The process of detecting the marker is mainly divided in detecting marker candidates, and in codification. First, the original image, Figure 4.6(a) is converted to grayscale, and then to binary by using an adaptive threshold, Figure 4.6(b). Then, a contour extraction is used in order to excluded non square shapes, and keep only marker candidates, Figure 4.6(c) and (d).

In order to decodify the marker, first a perspective transform is used to align the image, Figure 4.6(e), then a threshold is calculated, this time using Otsu's algorithm [30] to obtain a more accurate separation between the white and black sections. Finally, a bit by bit analysis is performed to obtain the binary code of the marker, Figure 4.6(f), and compare it to the specified dictionary and get its id.



0	0	0	0	0	0	0
0	0	1	1	1	1	0
0	0	1	0	1	1	0
0	0	1	0	1	1	0
0	0	0	1	0	0	0
0	1	1	0	1	1	0
0	0	0	0	0	0	0

Figure 4.6: . Image process for automatic marker detection [14]. (a) Original image. (b) Result of applying local thresholding. (c) Contour detection. (d) Polygonal approximation and removal of irrelevant contours. (e) Example of marker after perspective transformation. (f) Bit assignment for each cell.

After obtaining the id of the marker, the solvePnP function of OpenCV is used for every corner of the marker to find the rotation and translation vectors, this function is known as the n point projection. Consequently, after solving equation (4.1) the 3D position is determined from the 2D image.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{13} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_G \\ Y_G \\ Z_G \\ 1 \end{bmatrix} \tag{4.1}$$

Where s is the desired scale factor; the first vector describes the coordinates of the pixel point projected in the image plane; the second matrix are the camera matrix intrinsic parameters, which is obtained in the camera calibration procedure from the *camera_info* topic; then r_{ij} and t_i are each element of the rigid body rotation and translation matrix, respectively, between the camera and marker plane; and X_G, Y_G and Z_G are the coordinates of the 3D points in the world coordinate system.

The ROS library has a package called fiducials [39] that implements the specified algorithm above. For this package to function correctly, the following information is provided, the camera topics, such as the *info* topic containing the intrinsic and extrinsic matrices of the camera, provided by the Realsense ROS package [20] ; the image obtained by the camera, both raw and compressed, to perform the detection; the dictionary that the markers in use belong to; and the size of the side of the markers to be detected.

After various attempts, the pattern in Figure 4.7 was chosen to be used both in simulation and for the real robot. The reasons for the choice of this specific pattern were the following:

- A large marker that can be seen from up to 20m height
- A very small marker that can be detected from just a few centimeters
- Intermediate sized markers for an accurate detection from all heights
- A white border around the markers for the algorithm's corner detection
- A marker on every side of the pattern so the UGV is equally detectable from all angles

The pattern presented in Figure 4.7 has four markers, each with different, randomly chosen, IDs and sizes. The lateral size of the markers, from smallest to biggest is 0.0599m, 0.0840m, 0.1470m and 0.2069m; and the IDs are 55, 168, 227 and 946, respectively.

Transforms

When dealing with a robot, it is crucial to keep track of all the coordinate frames within the robot in order to guarantee its correct functioning. In ROS, the transforms [36] between every coordinate frame form a tree, where every node corresponds to a coordinate frame, and every frame has one parent and an unlimited number of children. This allows the user to transform points, vectors, etc., between any two coordinate frames at any time.

Initially, the implemented algorithm did not consider the arrangement of the markers in a pattern, therefore if only one marker was being seen at a time, then it would move towards the center of that marker, instead of the center of the pattern. However, it was concluded that, knowing the arrangement of the markers in the pattern was a better solution, since by knowing the distances of each marker to the middle of the pattern, the center of the UGV would always be the landing target. This avoids errors, such as landing on a border and risking the UAV tilting off.

To perform this translation from each marker to the center of the pattern, a static transform is published by creating a translated child to every marker node with the translation matrices for each marker.

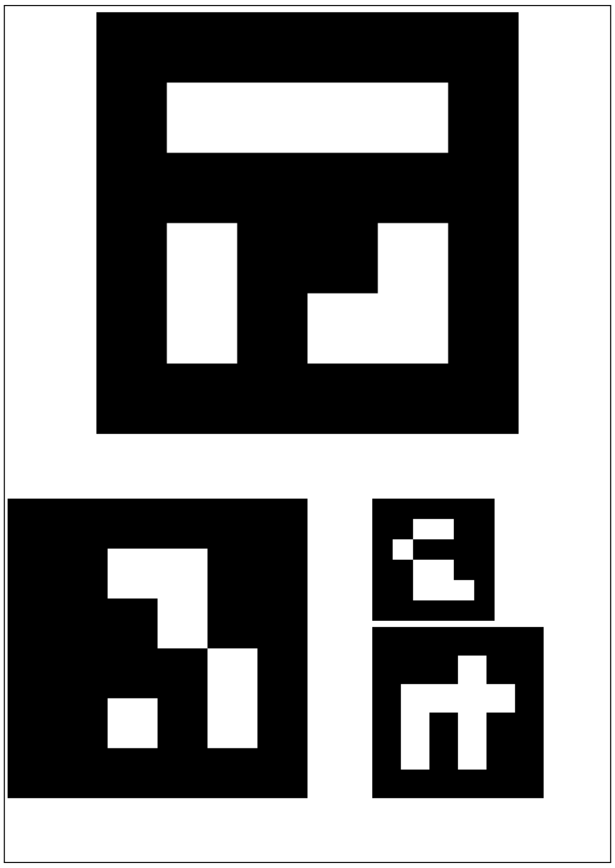


Figure 4.7: Marker pattern.

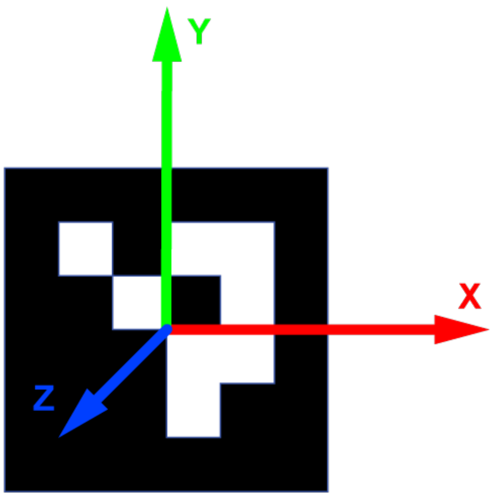


Figure 4.8: Aruco Marker axis.

These translations represent the distance from the center of each marker to the center of the pattern. It must be taken into consideration that the axis of each marker is oriented as East-North-Up (ENU), meaning that the x-axis is positive to the right, the y-axis is positive to the north, and the z-axis is positive to the front of the marker. In Figure 4.8 these axis can be better visualized, the orientation of the axis is relevant here to perform the translation from each marker to the center, where every marker is oriented as the one represented in Figure 4.8. The translations performed for each marker are the following:

$$T_{55} = \begin{bmatrix} -0.06175 \\ +0.06175 \\ 0 \end{bmatrix} T_{168} = \begin{bmatrix} -0.0738 \\ +0.1367 \\ 0 \end{bmatrix} T_{227} = \begin{bmatrix} 0.0735 \\ +0.1053 \\ 0 \end{bmatrix} T_{946} = \begin{bmatrix} 0 \\ -0.10345 \\ 0 \end{bmatrix} \tag{4.2}$$

Then, since the marker's pose is with relation to the camera frame, another static transform needs to be added to transform it to the body frame. Both in simulation as well as in with the real drone, the camera is flipped (has a π rotation in the Z axis), and is in the front of the drone, instead of in the center. The following transformation to correct the measurements is based on equation 2.1, where the necessary transform is the rotation of the axis with the translation to the center of the drone. As can be seen in Figure 2.1, in order to obtain the transform between the camera frame to the body frame, it is necessary to first rotate 90 degrees about the Y axis, and then -90 degrees about the Z axis. After these rotations are done, the Y axis is flipped, thus its symmetric is calculated by multiplying it with negative one, as well as the symmetric of the Z axis, since Z positive is upwards, its symmetric is needed to descend:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_B = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} R_y\left(\frac{\pi}{2}\right) R_z\left(-\frac{\pi}{2}\right) \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_C + \begin{bmatrix} -0.07 \\ 0 \\ 0 \end{bmatrix}. \tag{4.3}$$

Finally, after all the necessary transformations have been added, the transform frames between the Body Frame, the Camera Frame and the Target Frame, where in this case the Target Frame is the Fiducial's Frame, is represented as in Figure 4.9. The frames highlighted with red are automatically added by the realsense camera package, the ones in yellow are automatically added by the aruco detection package, and the blue are manually added.

Where $base\_link$ is the Body Frame, $camera\_link$ is the Camera Frame and their connection is the transformation in equation 4.3. Each fiducial transform is represented as $fiducial\_XX$, where *XX* is the id of the marker, and this transform is automatically published by the ArUcO detection algorithm along with its transform to the Camera Frame. The translation represented in equation 4.2 for each fiducial to the center of the pattern, is represented as $fiducial\_T\_XX$. Thus, by performing the transform from each $fiducial\_T\_XX$ to the $base\_link$, the pose of the markers relative to the Body Frame is obtained.

4.4 Implementation in ROS

Robot Operating System (ROS) is an open-source, meta-operating system for robots. The main mechanism used by ROS to communicate is by sending and receiving messages. In order to send these messages, nodes are used. These are software processes that perform a computation or task, but with the capability to register with the ROS Master node, and therefore communicate with other nodes in the system. These nodes can be scattered across multiple platforms, for example, between the robot's computer and a main computer, as long as connected to the same ROS Master node. To send messages,

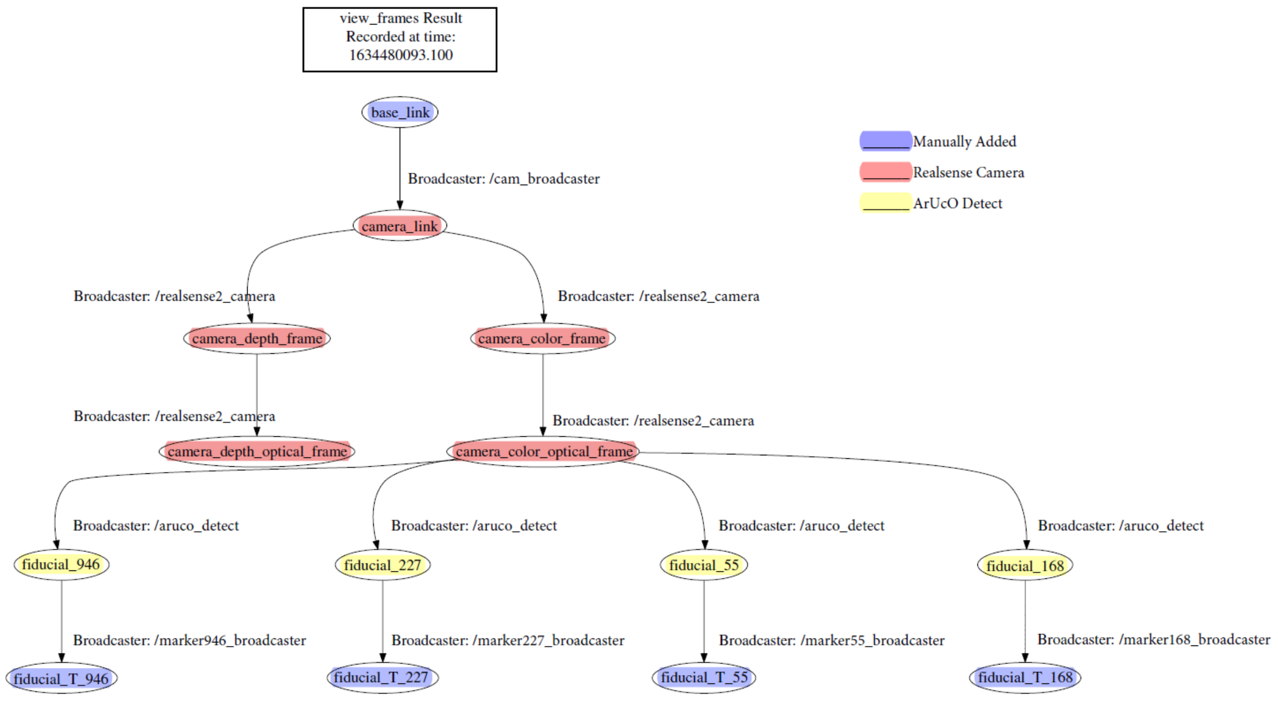

view_frames Result
Recorded at time:
1634480093.100
base_link
Broadcaster: /cam_broadcaster
camera_link
Broadcaster: /realsense2_camera
Broadcaster: /realsense2_camera
camera_depth_frame
camera_color_frame
Broadcaster: /realsense2_camera
Broadcaster: /realsense2_camera
camera_depth_optical_frame
camera_color_optical_frame
Broadcaster: /aruco_detect
Broadcaster: /aruco_detect
Broadcaster: /aruco_detect
Broadcaster: /aruco_detect
fiducial_946
fiducial_227
fiducial_55
fiducial_168
Broadcaster: /marker946_broadcaster
Broadcaster: /marker227_broadcaster
Broadcaster: /marker55_broadcaster
Broadcaster: /marker168_broadcaster
fiducial_T_946
fiducial_T_227
fiducial_T_55
fiducial_T_168
Manually Added
Realsense Camera
ArUcO Detect

Figure 4.9: Real Drone Tranforms.

a node needs to publish information. This information is called a topic. To receive information, the topic needs to be subscribed by the node, and transmitted messages on that topic will be received by the node. For example, in Figure 4.10, the node that will perform the Image recognition, needs to subscribe to the *Camera* topic to obtain the said image. Whereas the node UAV Controller, needs to publish to the topic that commands the linear and angular velocities of the UAV, to control it. Amongst other topics that are also necessary.

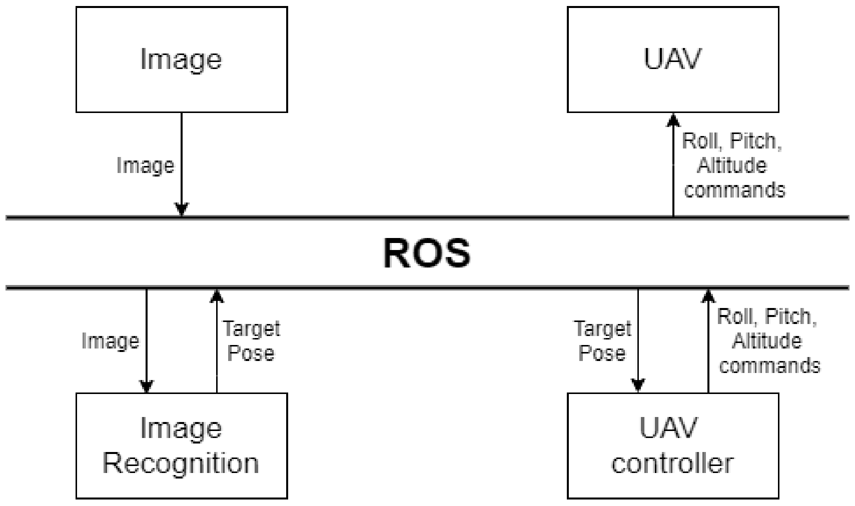


Figure 4.10: Communication using ROS.

4.4.1 Mavros

The communication between the ROS Node *UAV controller* and the autopilot is done through MAVROS [27]. The purpose of MAVROS is to handle messages between ROS and Ardupilot by converting ROS Topics into MAVlink messages. MAVlink is messaging protocol suitable for micro aerial vehicles due to being lightweight and low cost. This way, the autopilot can send the messages to the drone without having to make any changes to its functioning.

The ROS Topic used to send the velocity commands to the UAV is cmd_vel. This is a message of type geometry_msgs/Twist, which divides the velocity in linear and angular parts. Linear indicates the velocity for the x, y and z axis, and angular is the rotation in the same axis.

4.4.2 SMACH

SMACH (State MACHine) [6] is a ROS Python library [35] to build hierarchical state machines. SMACH is useful to execute high-level reasoning tasks where all states and transitions are explicitly defined, therefore should not be used for low-level states that require high efficiency.

This library provides two main interfaces, States and Containers. States are executed until a certain outcome is returned. Containers are a collection of one or more states. A Container can be interpreted as a State when nested in another Container, creating Hierarchical State Machines.

The types of States and Containers used in this implementation were the following:

- **Generic State** - A generic state with customized inputs, outputs, returns and execution;
- **CBState** - A state that executes a callback to a certain ROS Topic when it is active;
- **ServiceState** - A state that executes a ROS Service when it is active;
- **MonitorState** - A state that monitors a certain ROS Topic and terminates when 'invalid' is returned;
- **StateMachine Container** - A State Machine where each state is executed sequentially and only one state is being executed at a time;
- **Concurrence Container** - A Container that executes more than one state simultaneously. In the Concurrence outcome map, the outcomes are defined for when each child terminates, as well as for when the concurrence terminates.

SMACH also provides a visualization interface (SMACH viewer) that maps the structure of the running code directly onto a graph, allowing the user to visualize the created state machine as well as to identify errors in connections or execution.

In this implementation, the graph created as visualized by SMACH viewer is as in Figure 4.11.

SM_ROOT symbolizes the begining of the State Machine. In this case, the UAV starts with *GoHome*, a CBState that commands the UAV to a waypoint where the landing target is expected to be found, which is its initial position, or home. Once this is finished, a MonitorState *WaitAtHome* is initialized to monitor the marker's topic. This is achieved by having a ”marker message is empty” condition, when this

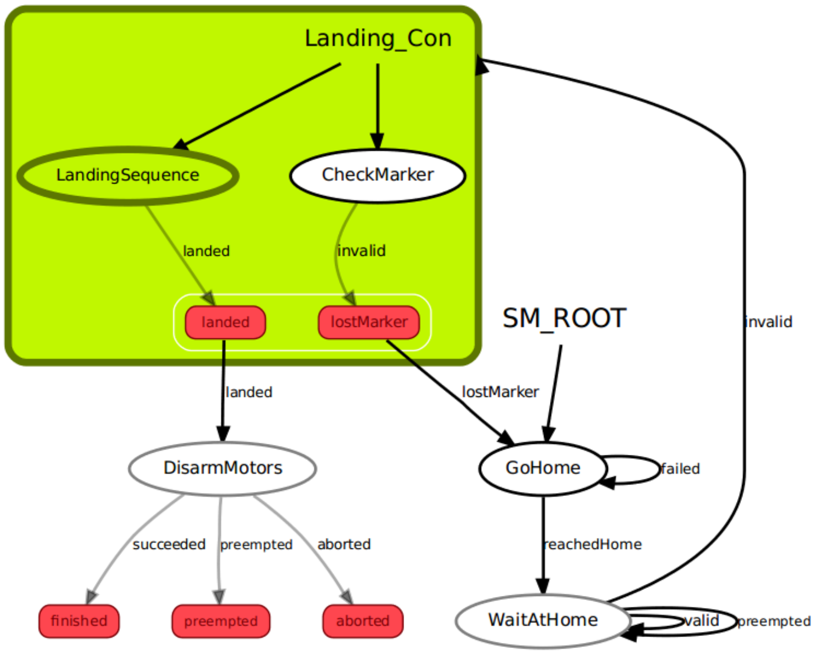


Figure 4.11: Implemented State Machine by SMACH Viewer.

condition is false, this state is terminated by returning 'invalid', thus starting the Concurrence Container *Landing_Con*.

Landing_Con has two concurrent states. *Landing Sequence* is a Generic State that starts the Publisher class in *follow_UGV.py*, and terminates when the class itself returns 'landed'. This class has its own inner low-level state machine to perform the following and landing. Since it requires high efficiency it is not implemented directly in SMACH, but instead in a separate class. While the landing sequence is occurring, a MonitorState *CheckMarker* is monitoring the marker's topic and, if for a certain number of consecutive iterations no marker reading is received, it returns 'invalid' on termination. This *CheckMarker* state is necessary because when the UGV is moving, the UAV might lose track of it at first and, as was mentioned before, the attitude controller will keep working with the latest measurements taken, and with these measurements it will only take the quadrotor a few seconds to reach its target, as will be seen in Chapter 5. However, it is necessary to have this concurrent state so that if something goes wrong, the UAV stops the controller and goes back Home.

In this Concurrent Container, only one state needs to terminate for the entire container to terminate, meaning that, if *CheckMarker* returns 'invalid' before *Landing Sequence* returns 'landed', the overall container returns 'lostMarker' and goes back to the initial state. Whereas if *Landing Sequence* terminates first, the container returns 'landed'.

Upon landing, the ServiceState *Disarming Motors* evoques the arming/disarming service in mavros, disarming the motors and consequently shutting down the State Machine.

4.5 Implemented Code

The implemented solution is divided in 3 main python files.

fsm.py

This is the main file therefore contains the high level state machine thoroughly explained in 4.4.2. As was said, when a marker is tracked, it starts performing the landing sequence, therefore moving on to the file *follow_UGV.py*, and only exiting it either when it stops tracking the marker, or when the file returns 'landed'.

follow_UGV.py

For each iteration of the markers topic, all the measurements for the currently observed markers are stored. Then, as was explained in Section 4.3, the translation of each marker to the center of the pattern has been published, therefore it is necessary to compute the transformation of the translated markers $fiducials\_T\_XX$ to the Body Frame $base\_link$ to obtain the pose of the center of the Target Frame with relation to the Body Frame. By performing this translation and consequent transformation, it is guaranteed that the UAV will always have as its target the center of the Landing Pad, independently of which markers can be seen.

Afterwards, an average of the previously corrected measurements is performed since the controller can only take one error as its input. While the real height of the UAV is higher than the top of the UGV, these errors will be sent to the file PID.py and then the output of the controller is published to the velocity command of the UAV.

PID.py

Three independent PI controllers were implemented for the movement on each axis, therefore, this class starts by calculating the commands for X and Y independently. When computing these commands, it is taken into account whether the UGV is moving or not, this is information is taken considering that both robots can communicate. This is due to needing higher PID gains to reach and match the UGV's velocity when it is moving. However, higher constants would lead to an increase in oscillation when the UGV is stationary. Thus these constants are lower when the UGV is not moving, and higher when it is. Further explanation regarding the functionality of each constant and the tuning method is on Section 5.2.

From the moment a marker is seen up until the mission is completed or aborted, the X and Y controllers are always attempting to follow the UGV, however, the Z controller will only attempt to descend if a marker is currently being tracked. This is a safety measure implemented to prevent the UAV from performing a wrong landing attempt, therefore, when the UGV is tracked, the Z controller is activated and starts descending.

Two conditions regarding the limitations of the camera's field of view need to be taken into account:

- UAV is bellow 20cm above the markers (measured value) - it will not be able to detect even the smallest marker since it is too close to perform the detection;
- UAV is below 60cm above the markers (measured value) - the overall pattern exceeds the field of view of the camera, hence not all markers are detectable.

Thus, having the previous statements in consideration, two safety measurements were implemented to avoid the maximum amount of errors possible when descending. First, if the UAV is perfectly aligned on top of the UGV, within a 10cm threshold to the center, and has descended up to 70cm above the markers, then an aggressive descent will be performed, by providing a fixed velocity of 1.2 m/s. This is a necessary measurement since otherwise it would stop descending below the 20cm due to not seeing the markers, as well as to avoid the ground effect when the drone is very close to the ground. Secondly, if the drone goes bellow the 70cm height above the UGV and the distance to the center if above the 10cm aligned threshold, the descent is stopped to allow the UAV to properly align with its target. To sum up, the two applied measures are the following (measures in distance to the target):

1. UAV within 10cm horizontal and 70 cm vertical - Aggressive descent of 1.2m/s
2. UAV over 10 cm horizontal and within 70cm vertical - Disable Z controller and enable only attitude controller.

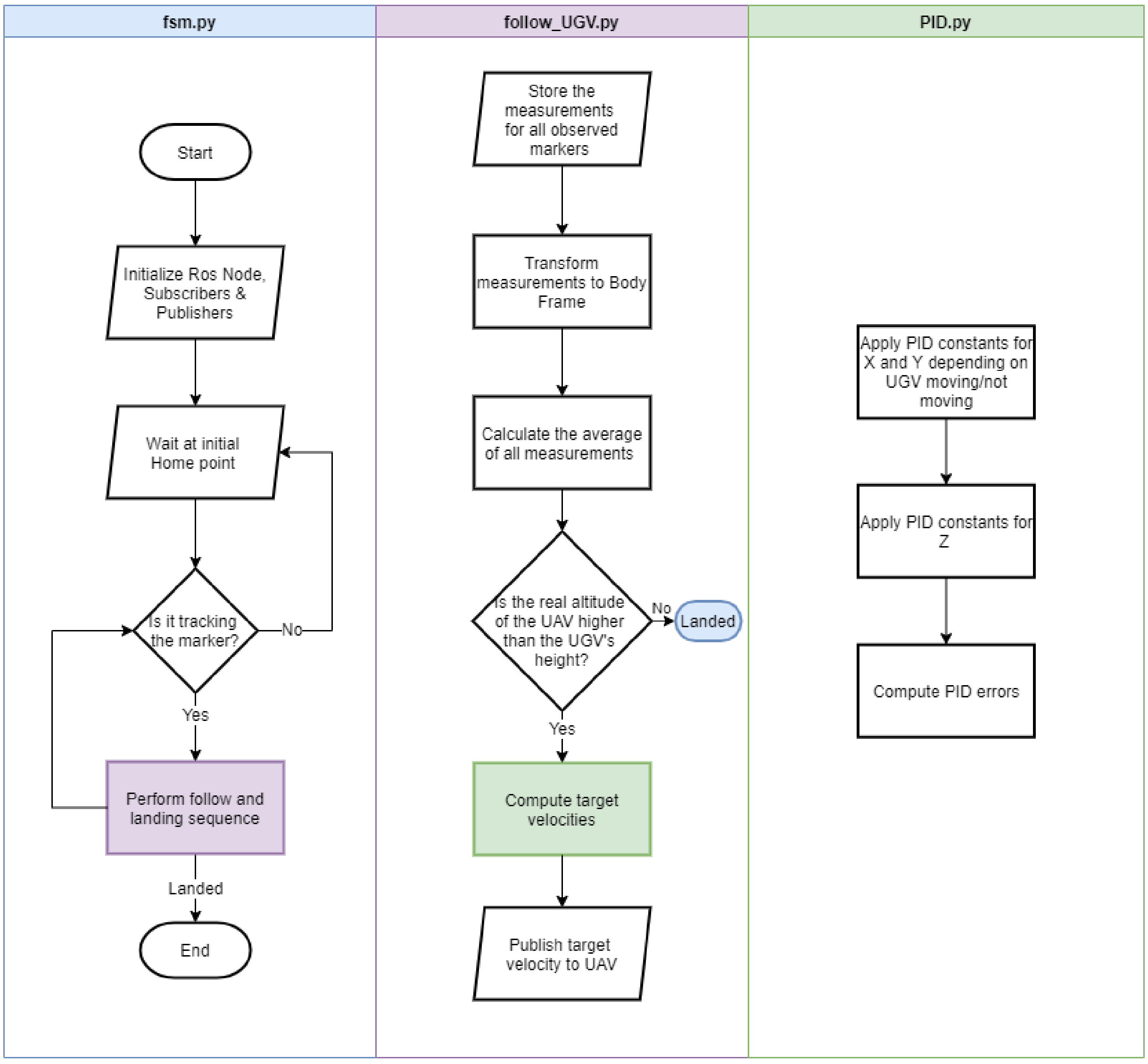

fsm.py
follow_UGV.py
PID.py
Start
Initialize Ros Node, Subscribers & Publishers
Wait at initial Home point
Is it tracking the marker?
No
Yes
Perform follow and landing sequence
Landed
End
Store the measurements for all observed markers
Transform measurements to Body Frame
Calculate the average of all measurements
Is the real altitude of the UAV higher than the UGV's height?
No
Landed
Yes
Compute target velocities
Publish target velocity to UAV
Apply PID constants for X and Y depending on UGV moving/not moving
Apply PID constants for Z
Compute PID errors

Figure 4.12: Code Flowchart.

Chapter 5

Simulation Results

In this Chapter, all the results performed in the simulation from the implemented solution will be presented. From tests to guarantee the correct functioning of the experiment in Section 5.1; to the necessary tuning of the controllers to ensure the best possible performance in Section 5.2; and to the performed tests in simulation in Section 5.3.

For all the tests in simulation the following steps are performed to set-up the simulator:

- Launch the simulator - This includes launching Gazebo with the environment described in Section 4.1, the Transforms, the ArUcO detection algorithm and Mavros;
- Launch the ArduCopter console - A console to view the state of the drone, as well as its battery and other modes;
- Change the Mode to Guided - Mode that allows the drone to be manually controlled, further explained in Chapter 6;
- Takeoff to 6m height;
- Position the UGV depending on the experiment in place;
- Run the implemented code.

These steps and the necessary files and packages to perform them are thoroughly explained in the GitHubs available on Chapter A.

5.1 Detection Range

In order to perform a successful landing, it is necessary to make precise estimates of the UGV's pose with relation to the UAV. As was shown in section 4.3, ArUcO markers are a reliable choice when measuring a 3D distance estimate, thus some tests were performed to guarantee and analyse how precise these estimates are.

To provide some context, the measurements from detected ArUcO markers are passed through an array. An unlimited amount of markers can be detected at this same time, as long as they fit the camera's field of view. The measures are then sent through the array and are distinguished by the ID and time frame at which they were detected. At each iteration, all the measurements done on that iteration are sent through the array.

Firstly, an analysis on the measured distance in the z axis was done in order to conclude the best starting height for this experiment. In order to do this, the drone was set to takeoff to 0.5m height, with the UAV placed directly underneath it. Then the UAV would be manually commanded to go up 1m or 0.5m at a time, until the markers stopped being detected. The markers, as was referred in Section 4.3, are organized from smaller to bigger in size and respectively in ID, hence the smaller marker has ID 55 and the bigger had ID 946.

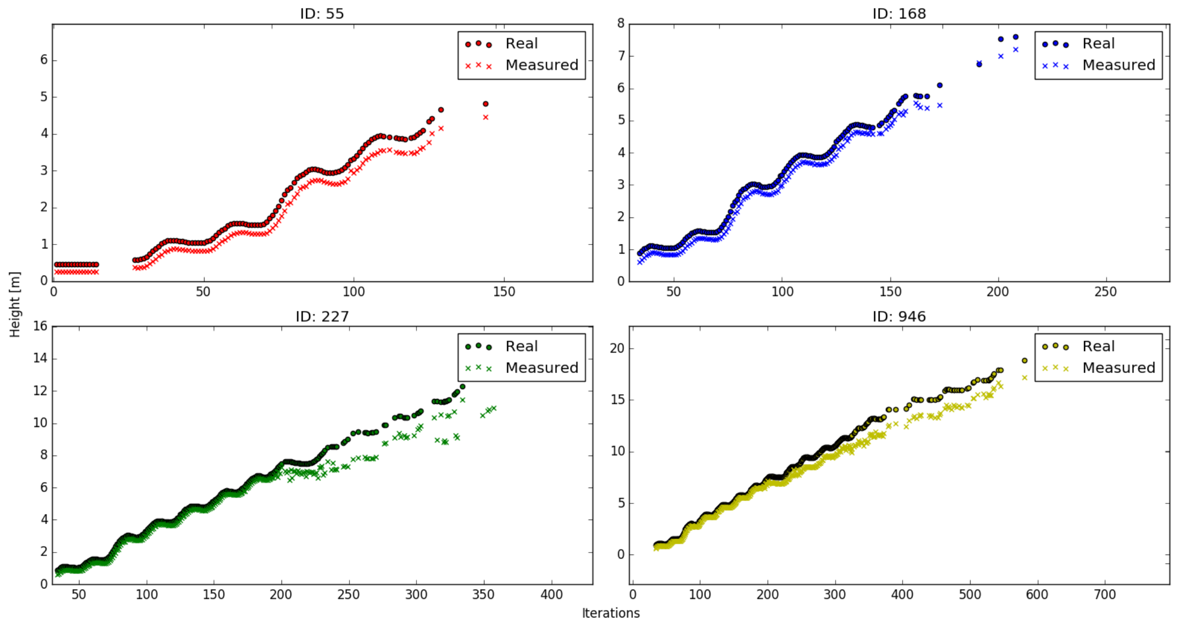


Figure 5.1: Minimum and maximum detectable height.

Figure 5.1 shows in the Y axis of the plot is the height, or distance in Z in meters, and the X axis of the plot is the time frame, or iteration between each measurement. It can be seen that every marker has an offset of ±0.28 between the measured and real value, this difference is the height of the UGV, since the markers are placed on top of the ground robot. In the beginning of the experiment there was a gap in the detection of marker 55, this was due to the marker momentarily going out of the field of view of the drone's camera. It is clear that larger markers can be seen from higher altitudes, however, the accuracy of the measurements is increasingly lower. Marker 946 can be detected up until 20m altitude, but starts increasingly losing accuracy from 10m height, the same happens to marker 227 from 7m altitude. Thus it was considered best to start the state machine at a maximum of 7m above the ground. On the other hand, markers 55 and 168 stop being consistently detected at around 5 and 6 meters height, therefore, to increase accuracy while still maintaining a wide field of view in the X and Y axis, it was defined that the takeoff altitude for the UAV would be 6m, which means that the state machine in Section 4.4.2, before commencing the Landing Sequence, sets the altitude of the UAV to 6m height, and all the experiments in the remainder of this Chapter already have this implemented.

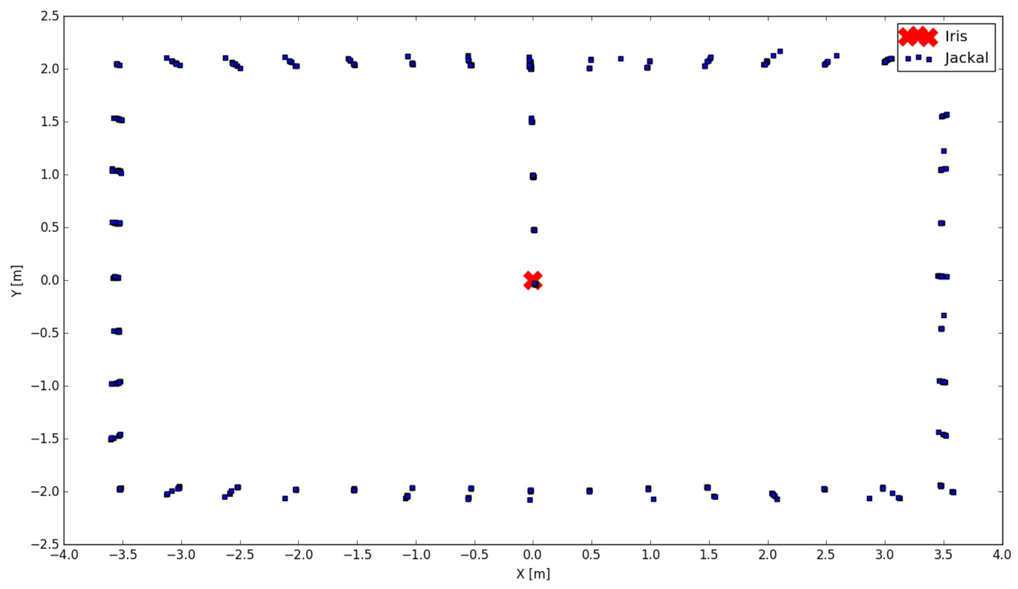


Figure 5.2: Camera field of view at 6m height.

After defining the optimal height from which to start the Landing Sequence, it is important to verify the maximum field of view of the camera from that height. As can be seen from Figure 5.2 and from Figure 4.3(b), the camera has a funneled field of view with a rectangle shape. Figure 5.2 shows the range of visibility of the camera when the UAV has 0 Yaw. To perform this experiment, both the UAV and UGV were place at the same X and Y coordinates, with the UAV at 6m altitude. The red cross in Figure 5.2 represents the position of the drone in (X,Y) = (0,0), and the blue squares are the measurements performed by the aruco detection algorithm estimating the UGV's relative pose. Firstly, both robots were at (X,Y) = (0,0), then, with the UAV always in the same position, the UGV was moved 0.5m at a time, first forward in Y, until the UGV left the line of sight of the UAV. The UGV would then go back to the last point where it could be detected, and moved manually on the opposite direction, still 0.5m at a time. This process was repeated for front, backwards, left and right until all the edges of the field of view of the camera were defined.

From this it can be concluded that, at 6 meters height, the limitations of the field of view of the camera of the UAV are $X \in [-3.5, 3.5]$ and $Y \in [-2.0, 2.0]$. This is important in order to know when the UGV starts being detected at the desired height.

5.2 PID Tuning

When tuning a PID controller, there are some basic notions that need to be taken into account first. The error that is received by the controller is the difference between the current state of the UAV, the Process Variable (PV), and the state of its target, the SetPoint (SP). The Gain, also called proportional gain, determines how much change the output will make due to a change in error, however, too much gain can result in an unstable oscillatory system. The Reset, or integral gain, determines how the output changes over time with the error, this determines how fast the output should move. The derivative action corrects the rate of change in the error, this way it acts as a counteract to the rapid changes in the error, however, when the error is noisy and contains small, random, rapid changes, it is a bad idea to use the derivative control. Since derivative extrapolates the current slope of the error, it is very affected by noise.

To start the tuning of the PID, first the input of the controller was taken into account. When the input error changes quickly, a large gain isn't recommended since it will lead to a drastic response to every change, hence a small gain is chosen to start with and then gradually increase it. The general tuning steps used for the different scenarios were the following:

- Set integral and derivative terms to zero and increase the proportional gain until the output of the control loop becomes faster without making the system visibly unstable, i.e. to the point where the error does not stabilize and tend to zero;
- When the proportional response is fast enough, gradually increase the integral term so that it reduces the error without overshooting;
- Once P and I have been set to a desired values with minimal error, the derivative gain is increased to improve the reaction to the error while reducing overshoot.

Since the UAV needed to be more reactive when the UGV is moving rather than when it is stationary, the tuning of the horizontal movement of the UAV was independent for each scenario.

5.2.1 Stationary UGV

To perform this tuning, the simulation was setup for each value tested, where the UGV would start at coordinates $(X, Y) = (2, 3)$, chosen due to being the edge of the field of view of the camera when the UAV is at position $(X, Y, Z) = (0, 0, 6)$, as was previously shown in Section 5.1. The UAV would then start the state machine, and once it detects the UGV, the Landing Sequence is initiated.

To tune the X and Y controllers when the UGV is stationary many tests with different values were performed, the values in Table 5.1 were considered most relevant to better visualize the optimal tuning solution.

First, the proportional gain was gradually increased until it started creating too much overshooting and leading to failure. As can be observed in Figure 5.3(a), the first two responses tested were effective but too slow; between the last two values, the full tuning of the PID was tested for both 0.3 and 0.4, however, the latter proved to be a bit more oscillatory and less reliable when it comes to consistency, hence the K_P was set to 0.3. Then, the integral gain is increased following the same school of thought, as shown in Table 5.1, a finer tuning was required, since a large integral gain on a slow changing target can lead to higher oscillations. The K_I value that optimally decreased the oscillations, as can be visualized in Figure 5.3(b), was 0.002. Finally it is necessary to tune the derivative gain in order to achieve optimal response speed, while having minimal overshooting; in Figure 5.3(c), the constant

Table 5.1: Values tested for the X and Y tuning when the UGV is stationary.

	K_P	K_I	K_D
Tuning P	0.1	0	0
	0.2	0	0
	0.3	0	0
	0.4	0	0
Tuning I	0.3	0.001	0
	0.3	0.002	0
	0.3	0.003	0
	0.3	0.004	0
	0.3	0.006	0
Tuning D	0.3	0.002	0.005
	0.3	0.002	0.01
	0.3	0.002	0.02

chosen as derivative gain for this scenario was 0.01 since it visibly anticipated certain changes, making the overshooting less sharp, while still providing a fast response,

5.2.2 Moving UGV

To perform this tuning, the simulation was setup for each value tested, where the UGV would start at coordinates $(X, Y) = (-5, 0)$ and move in a linear trajectory with a velocity of $1m/s$. The UAV would be at its initially position with the state machine enabled with the "WaitAtHome" state, and transition to the Landing Sequence when the UGV passed underneath it.

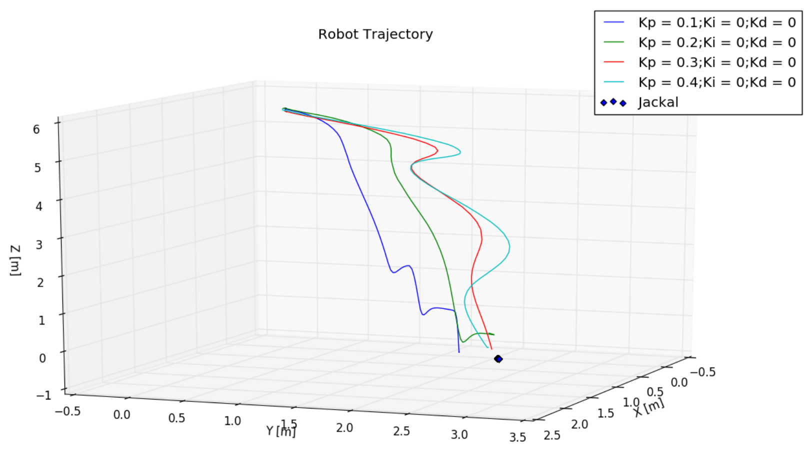
When tuning the X and Y controllers for the scenario where the UGV is moving, the initial tuning of the proportional gain response was skipped. This is due to the UAV not being able to attain the velocity of the UGV only through a proportional gain, therefore the K_P constants were chosen having in consideration the previously tuned scenario. When the UGV is stationary, the value 0.3 was chosen due to its fast response and reliability; with a moving UGV, the response however needs to be faster, so that the UAV won't lose track of the UGV. The following tuning was tested for a K_P of 0.3 and 0.4, the former proved to not be reactive enough to the fast changes a moving vehicle presents, thus the proportional gain chosen for this PID was 0.4.

The integral gain is considered the most important part of the PID when the UGV is moving, since the integral term increases its action with the accumulated error and the time for which this error has persisted. This means that if the applied force hasn't been enough to reduce the error to zero, this force will be increased as time passes, enabling the UAV to achieve the same velocity as the UGV.

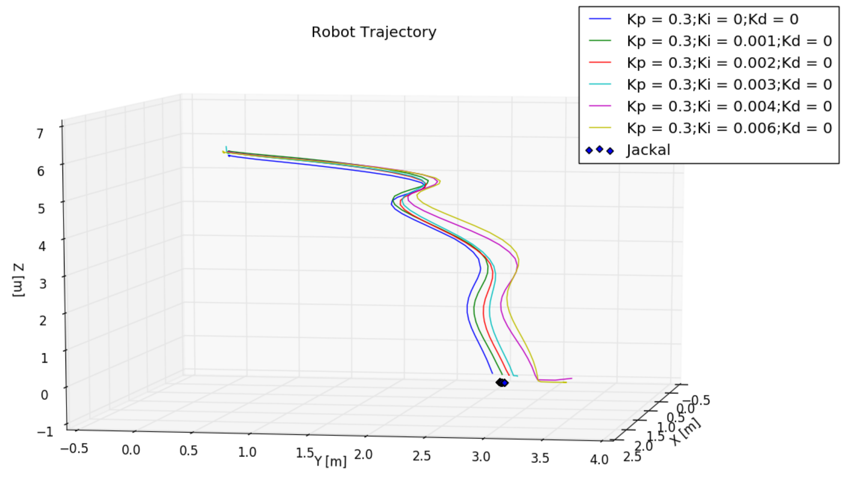
The tested gains for the tuning of this PID are shown in Table 5.2:

As can be seen in Figure 5.4(a), the first value wasn't fluid in its response thus being slower, whereas the last value tested was too oscillatory in the axis where the UGV isn't moving therefore being unreliable. The remainder of the tuning was performed for both K_I values of 0.04 and 0.05, both values were reliable and non oscillatory, however 0.05 proved to be much more unstable when it comes to consistency and different trajectories than the one tuned with, hence the optimal integral gain was concluded to be 0.04.

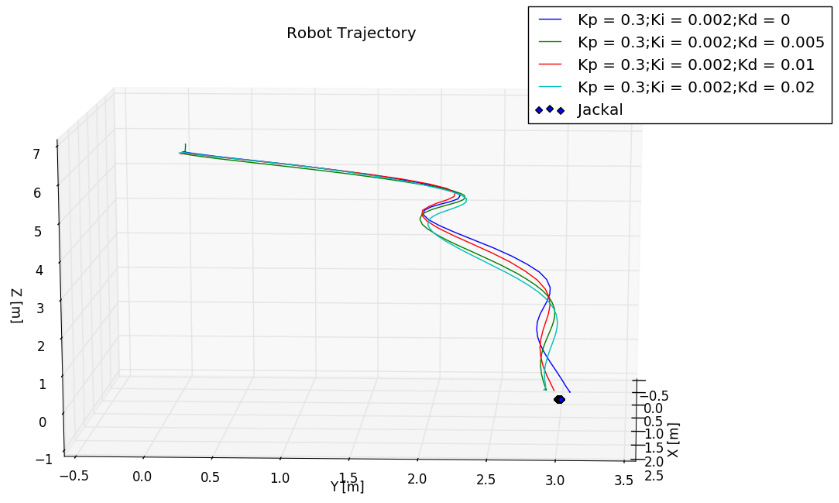
The purpose of the derivative gain is again to reduce the overshooting, its importance can be seen



(a) Tuning K_P.



(b) Tuning K_I.



(c) Tuning K_D.

Figure 5.3: PID Tuning with UGV stationary.

Table 5.2: Values tested for the X and Y tuning when the UGV is moving.

	K_P	K_I	K_D
Tuning I	0.4	0.03	0
	0.4	0.04	0
	0.4	0.05	0
	0.4	0.06	0
Tuning D	0.4	0.04	0
	0.4	0.04	0.05
	0.4	0.04	0.1

in Figure 5.4(b) since without the derivative part, dark blue line, there was a clear overshooting that required correction. Very large K_D values proved to be inefficient since they would make the response too fast or too slow; thus the optimal value for this gain was achieved at 0.005.

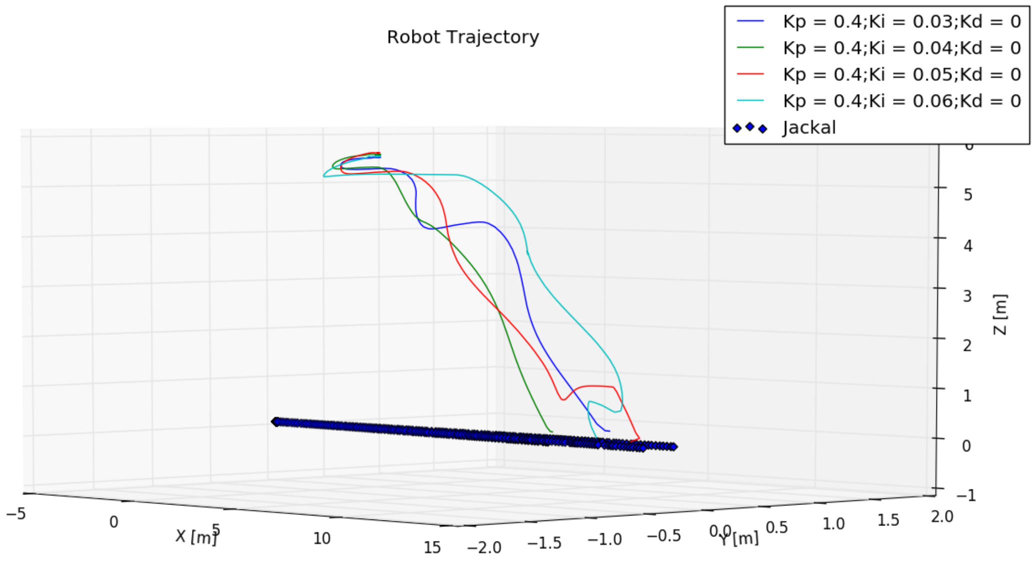
5.2.3 Descent

The controller for the altitude of the UAV needs to behave a little differently than the attitude controllers, the latter has the purpose of quickly reducing the horizontal error between the drone and its target without oscillating when reaching the target; whereas the former has some conditions that, if not fulfilled, would lead to a false landing attempt:

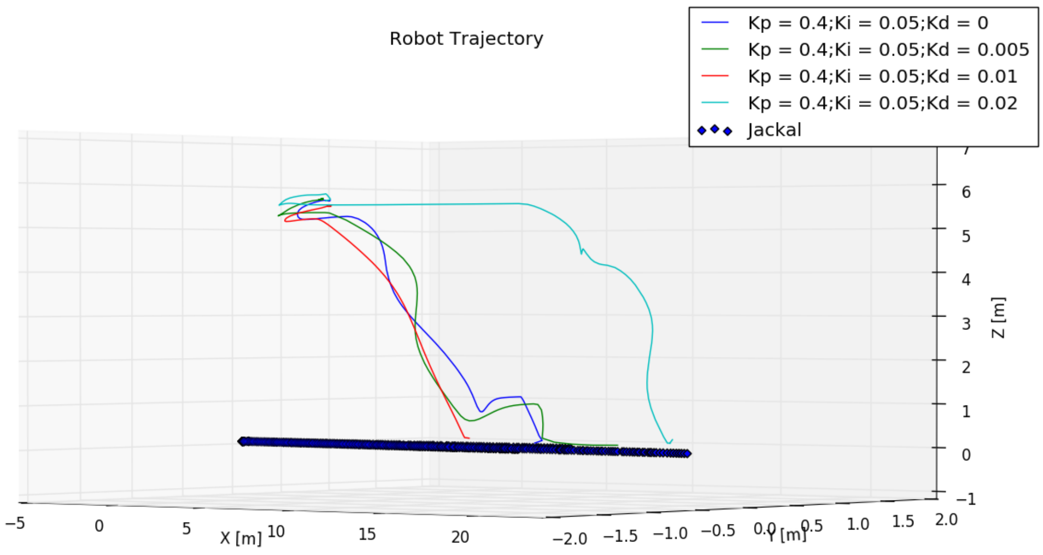
- The drone should only attempt to land if it is currently seeing a marker, this will avoid unnecessary descents when the attitude controller has not yet managed to match the velocity of the UGV;
- The descent should be slow at first to avoid aggressive movements that would lead to a very firm and inconstant descent;
- The descent should be faster when close up to the target to allow a fluid descent even when the the height is very small.

Having the above mentioned conditions in consideration, it can be clearly concluded that a simple proportional controller would not meet the requirements since it would provide an unnecessarily fast descent at high altitudes, and a very slow descent at low altitudes. On the other hand, an Integral controller would be able to bring the error to zero, with a slow reaction at start, and a much faster response at small heights. Hence, as can be seen in Table 5.4, for the altitude controller it was considered crucial to begin the tuning with the integral gain.

It is very important that the altitude controller provides a smooth descent, to avoid descending at a too fast rate and consequently having to stop descending due to losing the marker. The main goal is to provide a linear trajectory from the point it starts descending until it lands, to achieve this, several K_I values were tested. Larger gains would lead to the aforementioned scenario, while smaller gains had a slow response. The integral gain that proved to be effective while having a fast response was 0.008. Note that even though the integral part is crucial for this landing, its values are still very small to avoid a very aggressive manoeuvre.



(a) Tuning K_I.



(b) Tuning K_D.

Figure 5.4: PID Tuning with UGV moving.

The proportional gain provides an immediate response to a certain input. This part of the controller is mostly relevant in the beginning of the descent to provide a small ”boost” before the integral controller gradually increases its output. Having this in consideration, it can be seen in Figure 5.5(b) that a very large constant leads to an aggressive response at a large height, and consequently to a delay in the necessary time to start tracking the UGV. Thus, the proportional constant of 0.001 was chosen due to providing a slightly faster response.

As was mentioned before, there can be different sets of PIDs for different purposes, in a scenario

Table 5.3: Values tested for the Z tuning.

	K_P	K_I	K_D
Tuning I	0	0.005	0
	0	0.006	0
	0	0.007	0
	0	0.008	0
	0	0.009	0
	0	0.01	0
	0	0.012	0
Tuning P	0	0.008	0
	0.1	0.008	0
	0.05	0.008	0
	0.01	0.008	0
	0.005	0.008	0
	0.001	0.008	0
	0.0005	0.008	0
Tuning D	0.001	0.008	0
	0.001	0.008	0.01
	0.001	0.008	0.005

with fast changing values and noise, a derivative controller is not advised. Therefore, since the descent controller needs to act in a fast precise way, it was opted to exclude the derivative part of the descent controller. Nevertheless, some derivative constants were tested along with the previously defined proportional and integral controller, it was concluded that, no matter how small the constant, the derivative part would only worsen the overall response of the controller.

5.2.4 Consistency

To guarantee that the implemented solution is reliable after being properly tuned, it is necessary to test if the output is consistent, therefore always behaves in the same way. The best way to show this is by providing the exact same scenario, and verifying if the same output is obtained. The consistency of this implementation is tested both for when the UGV is stationary, as well as when it is moving at 1 m/s velocity.

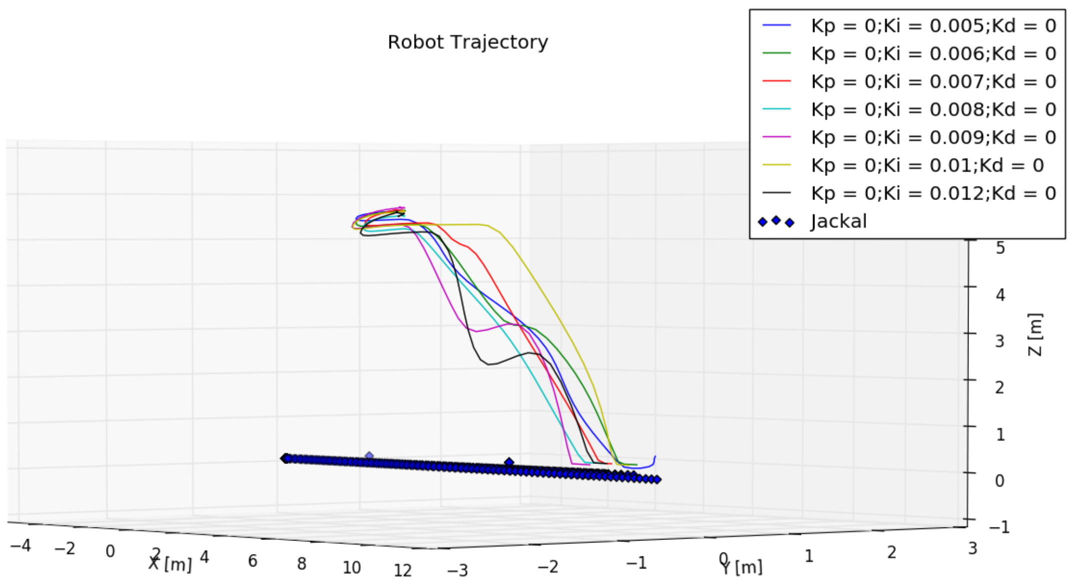
Stationary

The consistency of the implemented work was tested when the UGV is stationary at the position (x,y)=(2,3) since, as was mentioned in Section 5.2, it is the limit of the field of view of the camera. The simulation was run 10 times in a row for this exact scenario and the outputs were plot together to be compared, and the outcome was Figure 5.6.

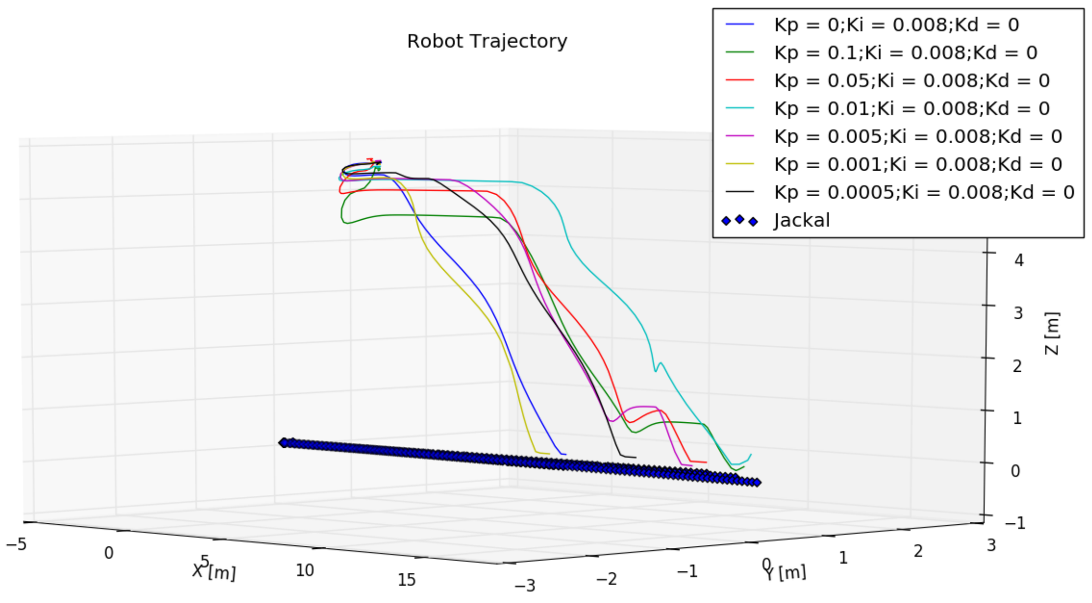
Afterwards, to further test this consistency, the UGV was placed in 9 different positions within the field of view of the drone when at its initial point (x,y)=(0,0) with 6 meters height. The UGV positions were the following:

The results from landing on a stationary target in different positions can be seen in Figure 5.7.

It can be clearly concluded from both Figures that the variations in the outputs were minimal and



(a) Tuning K_I.



(b) Tuning K_P.

Figure 5.5: PID Tuning for UAV descent.

the behaviour of the controller proved to be nearly identical on each iteration, independently on the positioning of the target.

Moving

When analysing the consistency for the UGV moving at a 1m/s velocity, the UGV is initially positioned behind the UAV at (x,y)=(-5,0), and when it starts moving it enters the field of view of the UAV which then

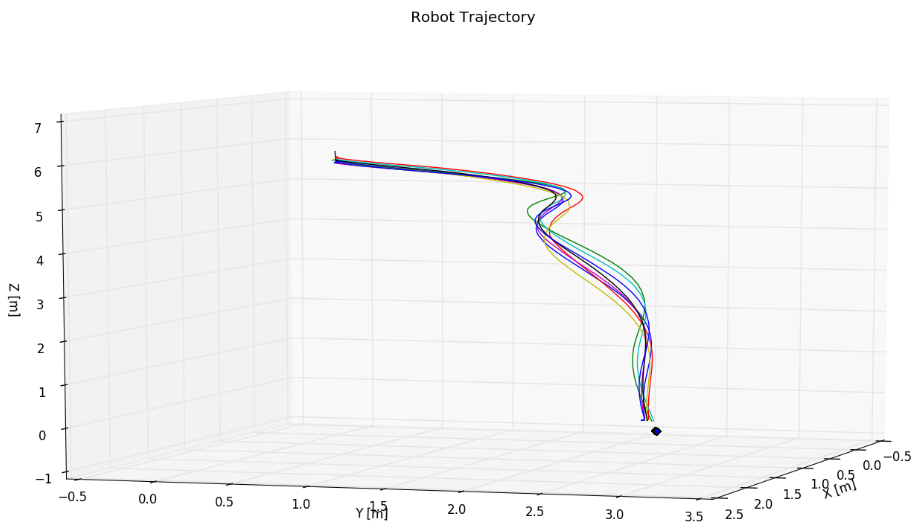


Figure 5.6: Consistency test when the UGV is stationary on the same position.

Table 5.4: UGV positions tested for stationary landing consistency.

X [m]	Y [m]
2	3
-2	3
-2	-3
2	-3
1	1
-1	-1
2	0
0	-2
0	0

attempts to track and land on it. The simulation was again run 10 times in a row for the above mentioned scenario and the outputs are represented in Figure 5.8.

Figure 5.8(a) shows a view of the lateral motion of the UAV. It can be viewed that eighth out of the ten outcomes were very similar, following a nearly identical trajectory, diverging by only a few centimeters. The remaining two experiments had an overshoot when attempting to land, which led to needing to reattempt the landing. This difference only resulted in a delay of around 5 seconds in the landing time. Thus it is concluded that all outcomes were in line with what was anticipated.

Figure 5.8(b) shows a frontal view of the trajectory, this image shows the oscillation of the controller on the axis opposite to the direction of the movement. It can be observed that this oscillation is minimal and it deviates at most 20cm from its target .

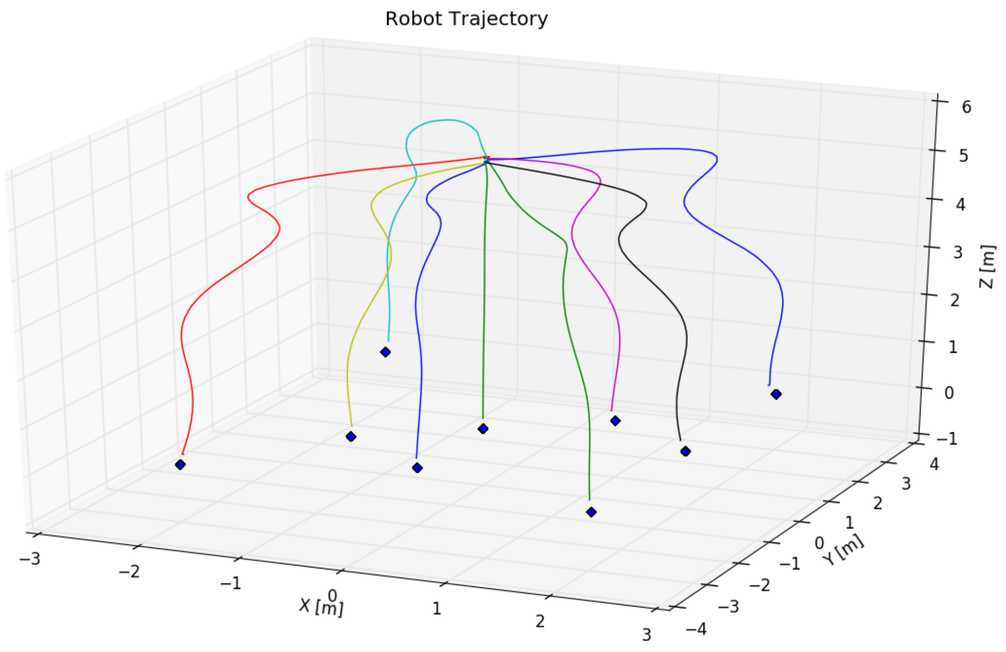


Figure 5.7: Consistency test when the UGV is stationary on different positions..

5.3 Simulation Experiments

After guaranteeing that the implemented controller functions correctly at the standard tuned velocity and trajectory, it is necessary to check its versatility by experimenting other velocities and trajectories.

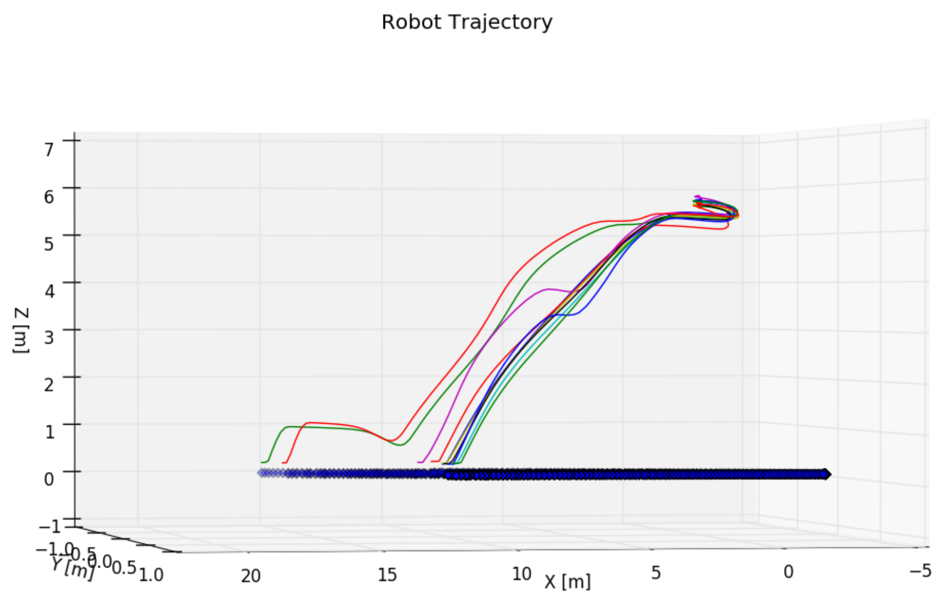
Linear Trajectory

In the first place, the same trajectory from the tuning was tested, where the UGV starts at (x,y) = (-5,0) and moves only in the x axis, with an increased velocity of 2m/s. The outcome of this experiment is represented in Figure 5.9. The average time to perform the landing at 1m/s velocity was between 10 and 15 seconds, whereas when the velocity is doubled, this time increases to 20 seconds.

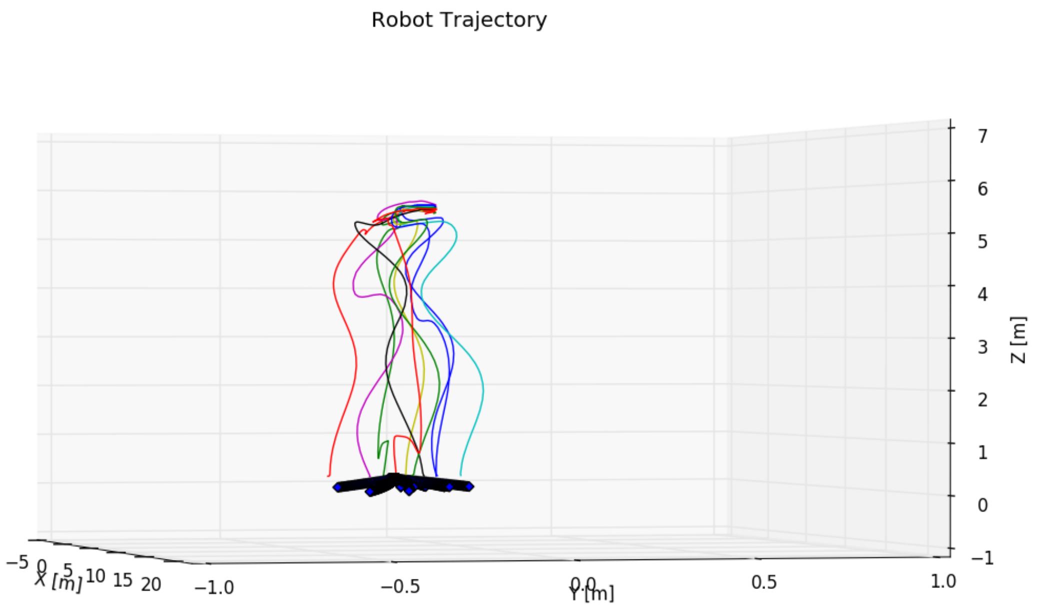
It can be observed that the oscillation in the direction opposed to the movement becomes considerable for larger velocities. This is due to the integral part of the controller gradually increasing its output to reduce the error. Even though these oscillations increase, they are still rather small in perspective, since they reach no more than values of 50cm.

Diagonal Trajectory

In all the previous tests, the UGV would reach the drone from one side, meaning that the UAV would only need to apply a large velocity in either X or Y in order to follow it. Since in this implementation it was considered that the UAV does not need to change its yaw in order to land, if the UGV approaches the UAV from a diagonal, then both X and Y will require high velocities to track the target. The result of this experiment in Figure 5.10, where the UGV was initially at (x,y)=(-4,-4) with a 45º angle about its z axis.



(a) Side view.



(b) Front view.

Figure 5.8: Consistency test when the UGV is moving.

This experiment, opposing to the previous ones, doesn't show considerable oscillation since both axis require high velocities. The overall landing proved to be smooth and has an average duration of the same 10 seconds as the previously tested experiments for the linear trajectory at 1 m/s.

As was performed for the linear tuned trajectory, a higher velocity is also tested for this scenario. The outcome of the diagonal trajectory at 2m/s velocity is as in Figure 5.11.

As can be recalled from the previously tested trajectory at 2m/s velocity, the time interval to perform

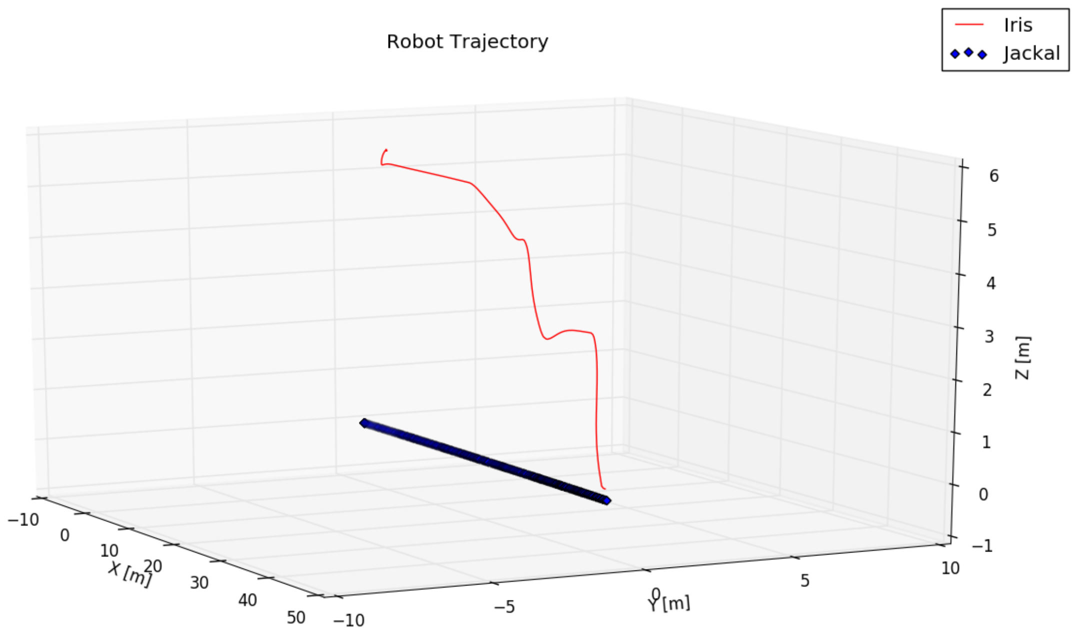


Figure 5.9: Linear Trajectory at 2m/s velocity.

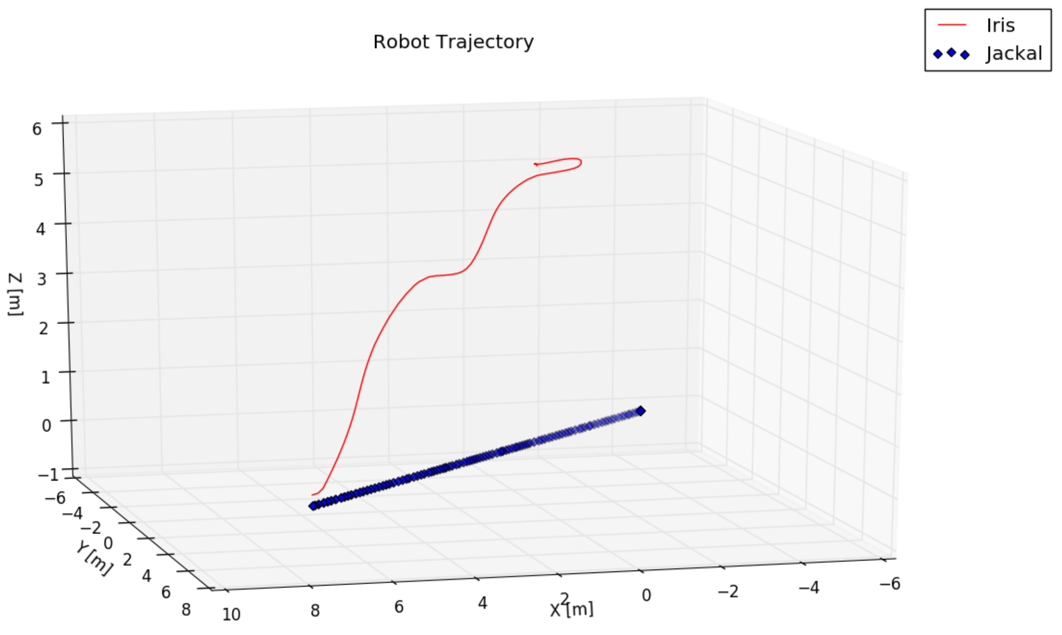


Figure 5.10: Diagonal Trajectory at 1m/s velocity.

the landing was more or less 20 seconds. For this experiment, the landing took 19 seconds to be completed.

Regarding the performed trajectory of the UAV, it can be observed that near the end and overshoot occurred and it was necessary to realign the attitude before continuing the descent. Overshooting is more prone to happen for higher velocities since the integrator takes longer to increase until it reaches the velocity of the UGV; and when the drone reaches the UGV, its velocity is higher than its target, and

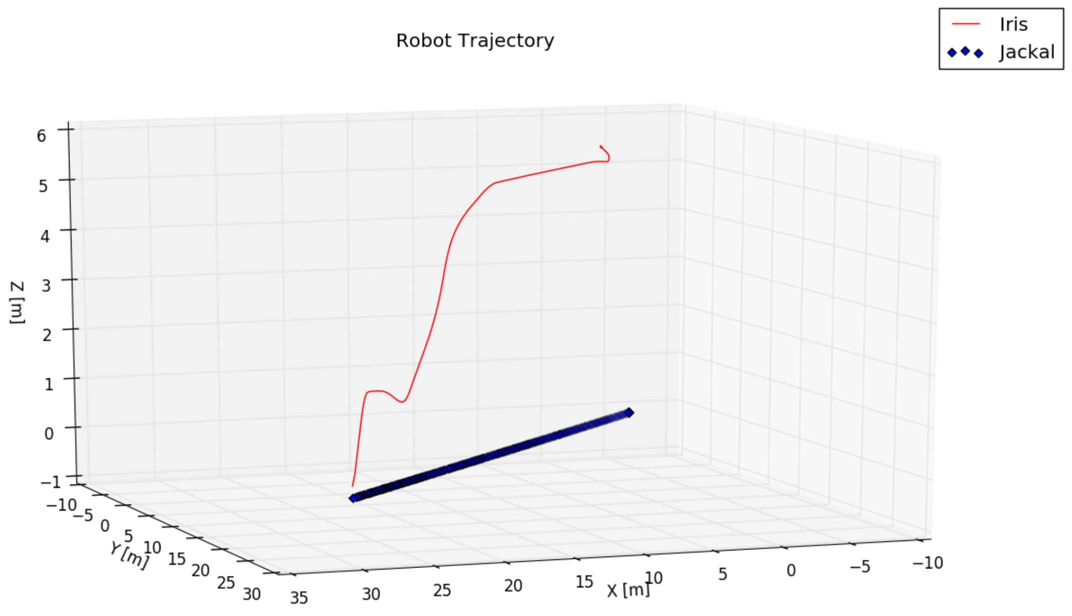


Figure 5.11: Diagonal Trajectory at 2m/s velocity.

the correction causes an overshoot. Overall the landing was performed at the expected time and without any significant oscillations, showing that, in simulation, the controller was efficient, by performing the landings in a very short time period, and precise, for accurately landing on its target.

Chapter 6

Real Drone Results

In this Chapter, all the results tested on the real drone will be presented. After concluding the experiments in simulation and guaranteeing that everything worked correctly, it was time to move on to implementing the solution on the real drone. First it was necessary to install the required packages on the drone's computer; then all the frames in the drone were analysed to guarantee correct transforms between the markers and the frame of the drone. To test this last step, the Motion Capture System available on the 8th Floor of ISR was used and its analysis is in Section 6.1; then the final step was to move on to the flight tests with the real drone overviewed in Section 6.3.

For all the tests in the real drone, the following steps are performed to set-up the drone:

- Connect all the needed terminals to the drone through SSH;
- Run the launch file - This file includes the launch files for the Mavros node, the realsense node, the aruco detection algorithm, and the necessary transforms;

If it is a flight test:

- Manually takeoff the drone
- Change the Mode to Guided - Mode that allows the drone to be manually controlled, further explained in Section 6.3;
- Run the implemented code.

These steps and the necessary files and packages to perform them are thoroughly explained in the GitHubs available on Chapter A.

6.1 Motion Capture System

A Motion Capture System, also known as Mo-cap, is an environment composed of at least two cameras, that when calibrated provide a the 3D position and orientation of a certain object. This is achieved by attaching passive markers to the UAV, coated with retroreflective material, which is detected by the cameras since it reflects the light in the lenses.

Before moving on to testing the implemented controller on the drone, it was first necessary to guarantee that the distances measured to the marker and transforms were correct, thus the Mo-cap system was used to compare the real and measured distance of the drone to the markers. First, a previously known object was used to calibrate all the cameras; then the drone with the passive markers was placed on top of the ArUcO marker pattern, which was previously taped to the floor. Then, the position of the ArUcOs is set as the origin of the world, so to obtain the real pose of the UAV with relation to the center. Finally, while having an external computer recording the logs for both the Mo-cap system as well as the ArUcO detection program, different experiments were performed to verify the accuracy of the measurements.

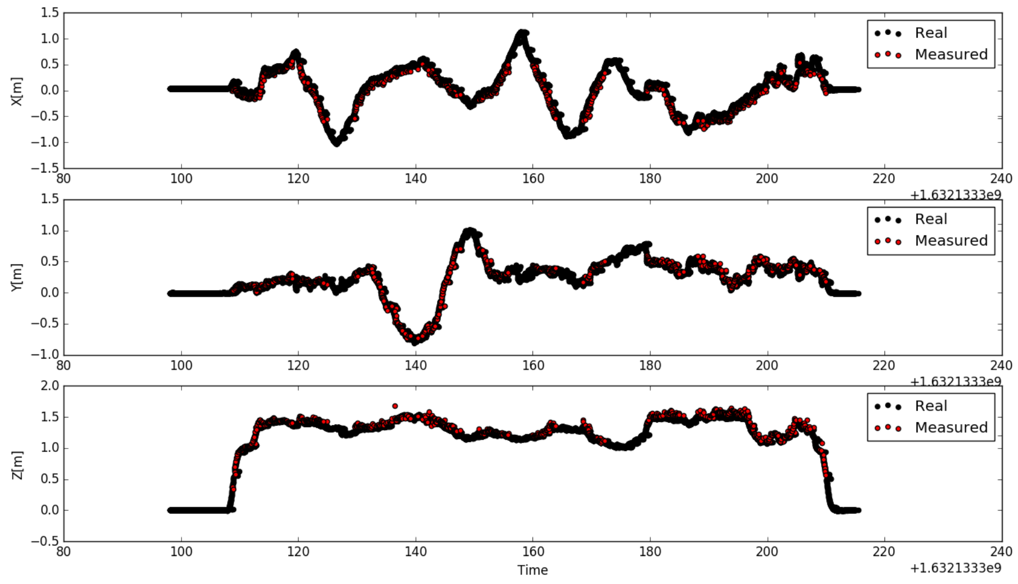
On the first try-out, the drone was moved manually in X and Y at a fixed altitude, without tilting it around its axis. The quadrotor would first be moved to the left until the markers left the field of view of the camera, then back to the center, and the same would be performed for the other sides one at a time, front, right and back. Then, at 180 seconds, a full rotation of yaw in the drone was performed.

Figure 6.1 is plotting both the values measured by the drone for each individual marker, and by the Mocap system over time. By analysing Figure 6.1(a), it can be concluded that, under the aforementioned circumstances, the distance measures performed by the vision detection algorithm were nearly perfect and therefore reliable. Figure 6.1(b) shows the relative angles for all markers and the Mocap system. It can be clearly seen here that the drone moves until the markers go out of bounds of the camera, since the values for the markers are intermittent. The values measured for the Yaw are very consistent and evidently show the full rotation in Yaw previously mentioned at 180 seconds, where the values descend to minus Pi and switch to Pi going then back to the initial value. The Roll and Pitch values on the other hand, even though they are similar to the mocap values, they show a clear offset, mainly on the smallest and largest markers, 55 and 946. This can be due to being far away enough so that the smallest marker's measurements become inaccurate, or close enough that the largest markers measurements are still not as accurate. But overall the relative angles are only fully reliable for the Yaw.

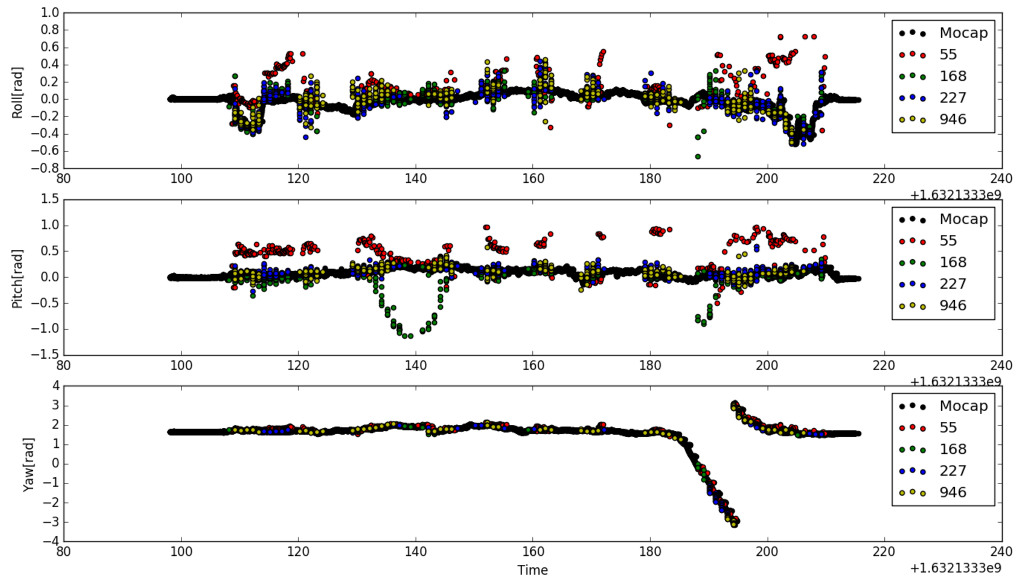
On the second try-out, the drone was moved manually in a circle in X and Y around the markers at a fixed altitude, while maintaining it aggressively tilted to test the accuracy of the results for high Roll and Pitch angles, and amidst the experiment at 110 seconds, the drone was rotated while tilted.

From Figure 6.2(a), it can be concluded that the distances are equally well measured, even with extreme Roll and Pitch angles, meaning angles higher than the ones attained by a normal quadrotor flight. In Figure 6.2(b), both the Roll and Pitch angles were accurately measured for all markers with the exception of some outliers. It can be seen that both Roll and Pitch were tested for a range of [-1, 1] radians, values that are hardly attainable in a flight run. Thus it can be concluded that, when flying the drone, the Roll and Pitch angles will not disturb the measurements. When analysing the Yaw the same can be concluded, since the 360º turn while keeping both Roll and Pitch with high angles did not alter the reliability of the measured distances.

When comparing Figure 6.2 with the previous Figure 6.1, it can be seen that, at 70 seconds, when initially increasing the altitude, the Roll and Pitch angles also show an offset, that gradually goes away



(a) Distance.

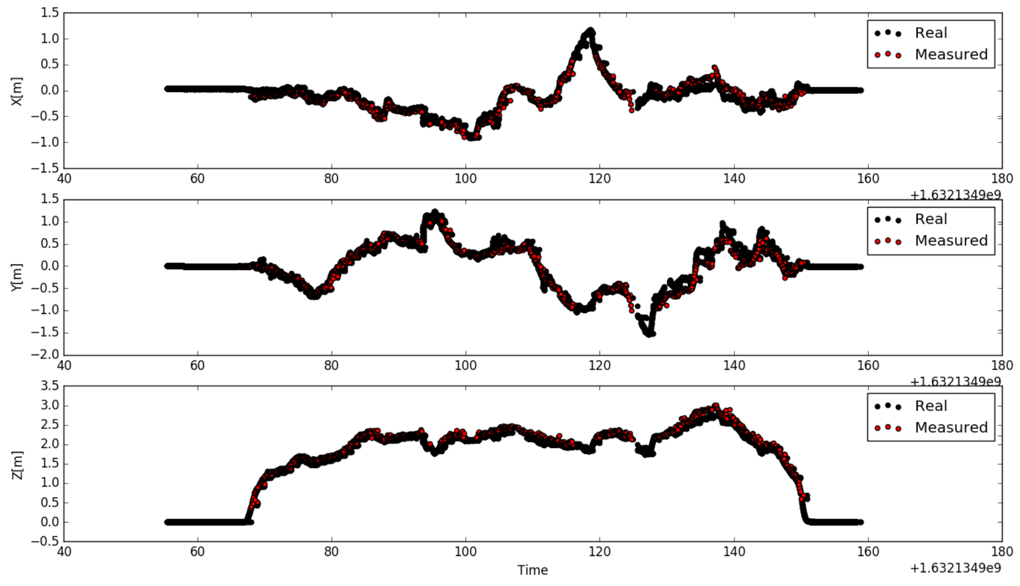


(b) Angles.

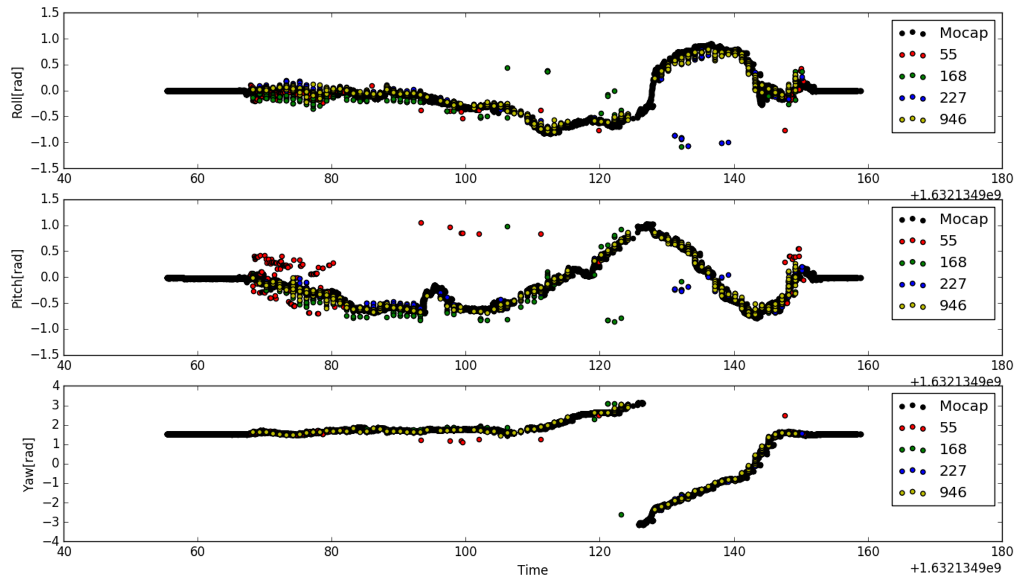
Figure 6.1: MoCap System Experiment 1.

when the altitude is increased.

The Mocap system allowed to reach the conclusion that the distance measurements were very reliable and thus it is safe to begin the flight tests for the controller.



(a) Distance.



(b) Angles.

Figure 6.2: MoCap System Experiment 2.

6.2 Detection Range

As was previously analysed for the simulation, it is required to test the limitations in the detection of the markers. It is particularly important for the landing to verify the descent height at which the markers stop being detected. To perform this test all the launch files were run on the drone, and then manually, while aligned on top of the marker pattern, the drone was lifted until all the markers were being correctly measured, and put back down.

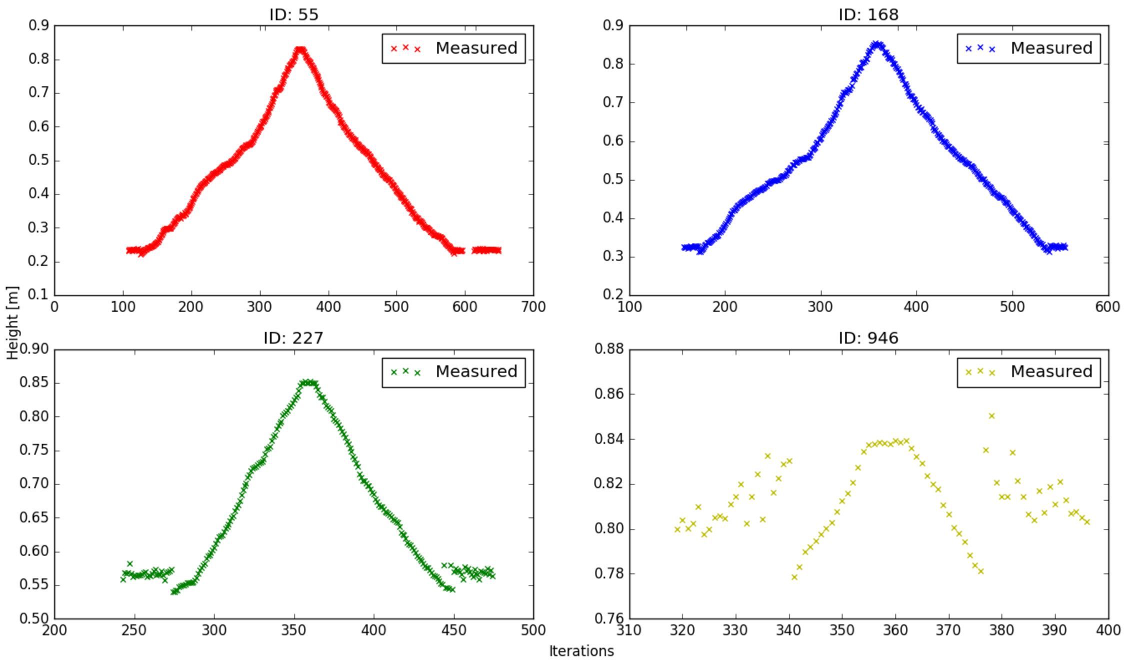


Figure 6.3: Minimum detectable height real experiment.

From Figure 6.3 it can be seen that the largest marker, ID 946, only starts being accurately detected at 0.78cm height, the next biggest with ID 227 at 0.53cm, ID 168 at 0.32cm and the smallest marker with ID 55 at 0.23cm. This shows the importance of performing the Aggressive Descent, as was mentioned in Section 4.5, at 70cm above the markers. The 70cm were defined in simulation as the distance in Z at which the camera could still detect all the markers, thus being the optimal safest landing choice, to avoid losing the markers. Therefore it is proven that the implemented state machine is coherent as well for the real drone and it is safe to move on to testing it in real flight tests.

6.3 Flight Experiments

6.3.1 Tuning

The simulation in gazebo was set up with the closest possible features to the real drone to allow the experiments to be realistic. However, the tuning of the controller may vary between the simulation and a real environment, for reasons that go from hardware to weather issues. Thus, the first flight test performed was with only the attitude controller enabled, while experimenting different values of the proportional constant. The overall procedure of initiating and setting up the drone was as explained in the beginning of this Chapter, for the tuning the implemented code would be run with the altitude controller disabled, to verify the response of the attitude controller to different proportional gains.

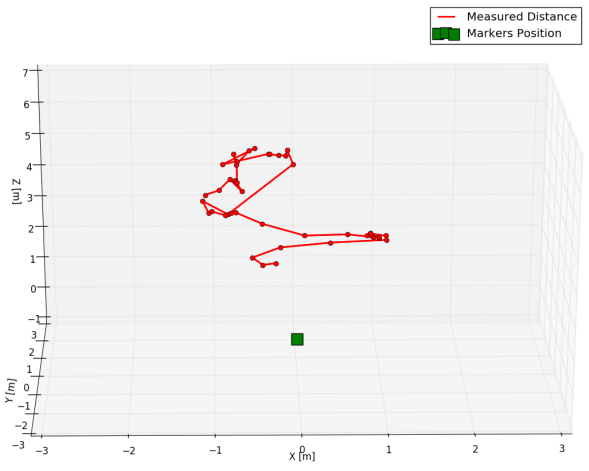
Since the tuning had already been performed for the simulation, there was a guideline for which values were best to test. The tested values are as in the Table 6.1:

The first experiment was with K_P of 0.3, which was the tuned value for the simulation, however this value was too small and the UAV would react too slowly, as can be seen in Figure 6.4(a), where at

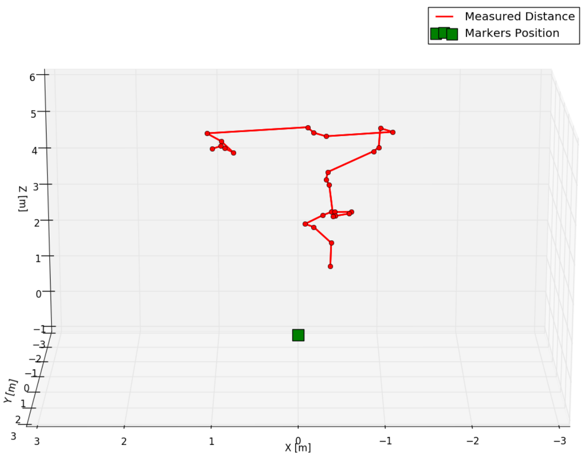
Table 6.1: Values tested for the X and Y tuning for the Real Experiment.

	K_P
Tuning P	0.3
	0.35
	0.4
	0.5

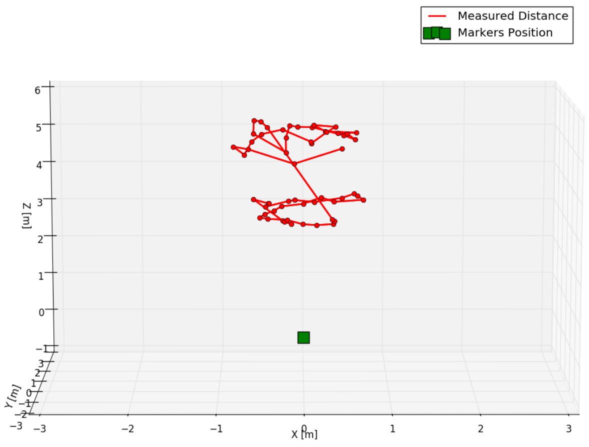
first the movement was very slow and when the altitude was manually decreased so changes could be visualized.



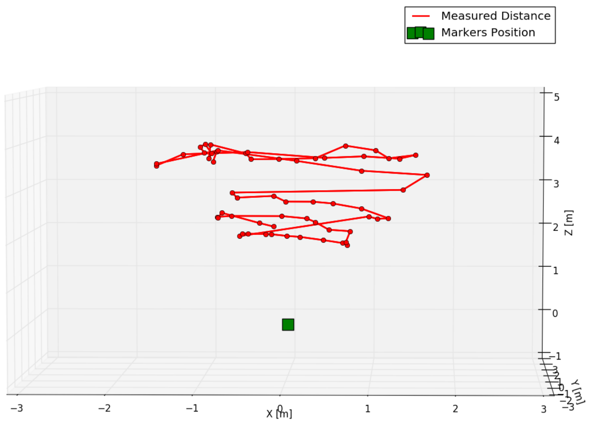
(a) K_P = 0.3.



(b) K_P = 0.35.



(c) K_P = 0.4.



(d) K_P = 0.5.

Figure 6.4: Real Drone tuning.

Afterwards a test was run with the proportional gain as 0.4. Initially, at 3 meters height, it can be seen in Figure 6.4(c) that the drone performed a steady oscillation around its target, thus the altitude was manually increased and it can be seen that the behaviour remained the same.

Even though this was a good result, 0.5 and 0.35 were still tested. Figure 6.4(d) shows that 0.5 was too aggressive, overshooting up to 2m past its target. Whereas 0.35, as in Figure 6.4(b), still showed a very slow and not reactive enough response.

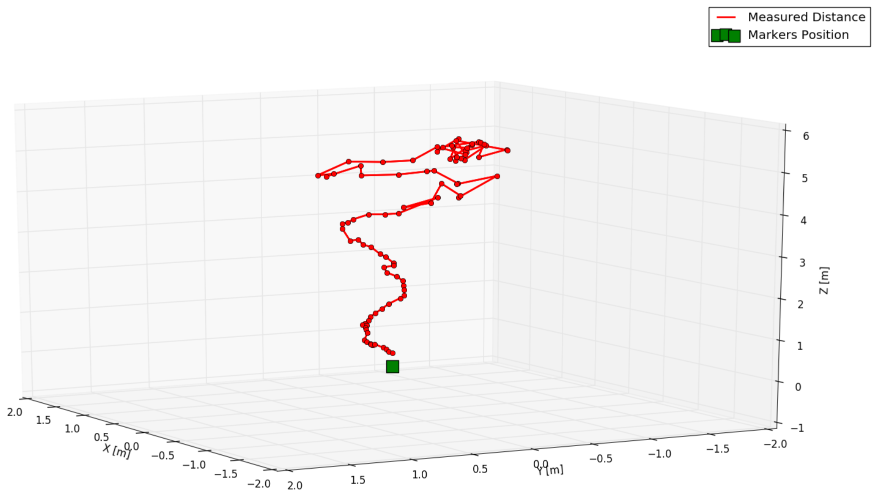
Thus it is defined that the proportional gain set for the real drone is of 0.4 and all the remaining experiments have this implemented.

6.3.2 Landing Experiments

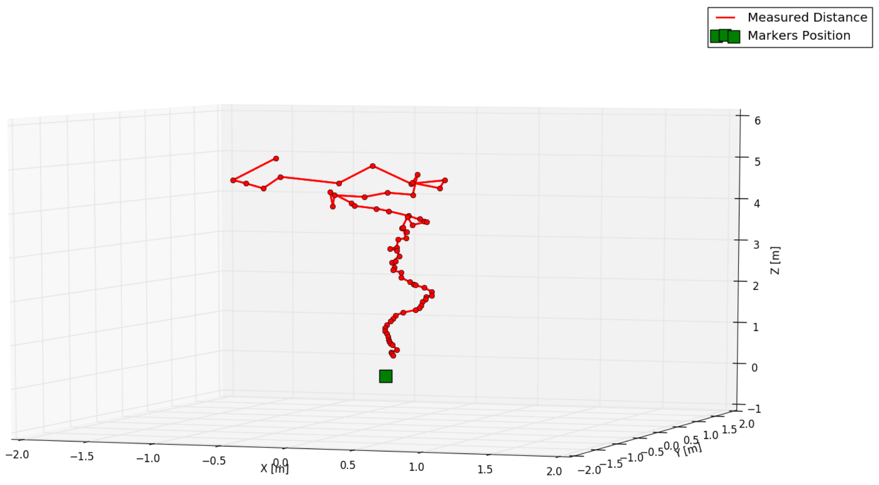
To perform the landing experiments, the drone was manually taken off from the markers, after reaching the optimal height of around 6 meters the mode on Mavros was switched to ”GUIDED”. Mavros has several modes with different purposes, while the drone is being manually commanded, it does not respond to other inputs, Guided mode allows the drone to be controlled internally through a running code. When the mode is switched, manual commands stop being accepted and the drone is entirely controlled by the implemented code. The following plots represent the marker position estimates from the point when the mode was switched to Guided, hence when the controller started commanding the drone.

Figure 6.5 plots three very successful landings, where the drone can be seen initially hovering, then, when the code is run, it initiates the Landing Sequence. It moves towards its target while slowly descending and gradually decreases the oscillations in attitude until it lands. As can be seen by the green square representing the position of the markers, the landing was always successful, demonstrating the precision previously proven in simulation.

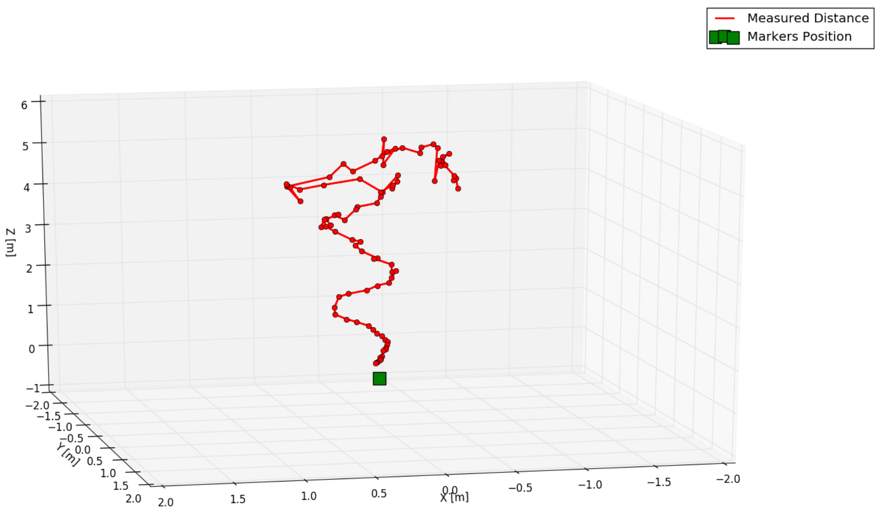
These experiments were three concurrent experiments taken in order to prove consistency by obtaining the same result on different runs.



(a) Test 1.



(b) Test 2.



(c) Test 3.

Figure 6.5: Real Drone Flights.

Afterwards, an experiment was done by starting the controller while further away from the target as well as with a different yaw than from the previous experiments.

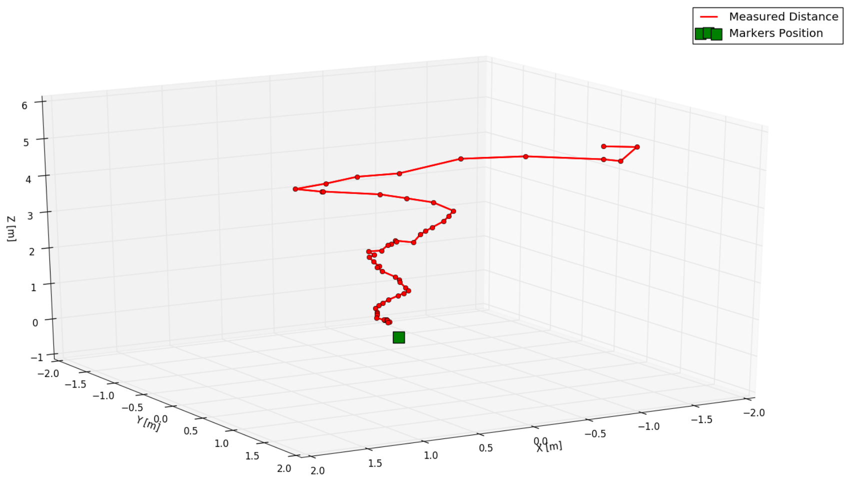


Figure 6.6: Real Drone Flights.

Since the drone was further away from its target, the first time the drone passed by the target led to a bigger overshoot past the markers than the previous tests, however, it can be observed from Figure 6.6 that starting further away from the markers and rotating the drone did not change the end result and the landing was still performed very fast and accurately.

While performing several experiments in a row, in one test it was possible to observe what happens when the markers are lost and it is necessary to realign in X and Y before reattempting the landing.

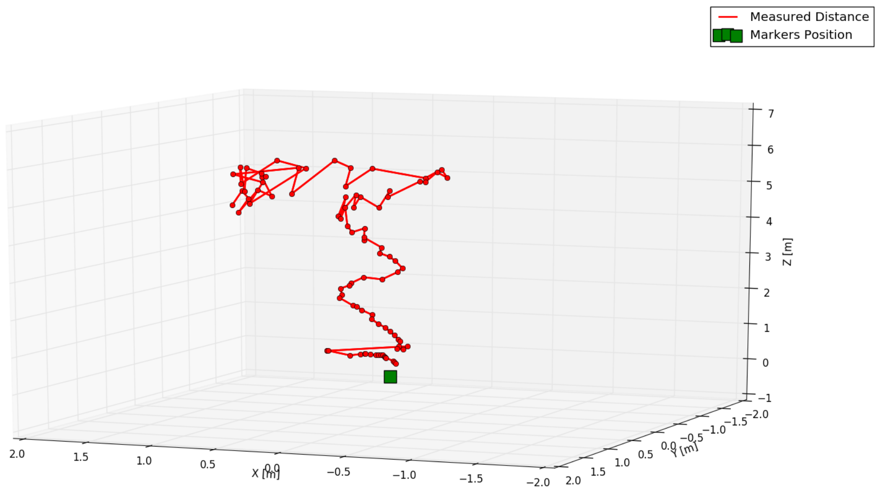


Figure 6.7: Real Drone Flights.

Figure 6.7 shows this experiment where the UAV was descending and it reached a point where from that height and distance it could not see a marker, therefore it immediately stopped, attempted to realign from the previous stored measures and then proceeded to land.

From these experiments it can be concluded that the implemented solution performed well in a real world situation proving to be viable and dependable when it comes to accurately and safely landing the UAV.

Chapter 7

Conclusions

This thesis presented the development of a cooperative system constituted by an Unmanned Aerial Vehicle and Ground Robot that, with the aid of vision, leads the quadrotor to fully autonomously detect and land on a possibly moving ground robot. To achieve this, a State Machine was developed to perform the high level decision tasks of the UAV. Then a cascaded controller was built, containing in its outer loop three independent controllers for the attitude and altitude, and in its inner loop the angle and rate controllers of the autopilot. The quadrotor for this project contained a Pixhawk 2.1 Autopilot with the Ardupilot open-source autopilot system, which allowed this work to perform successful higher level tasks without compromising the stability of the UAV. The vision-based detection of the target was performed with the aid of fiducial markers, more specifically ArUcO marker, setup in a pattern with differently sized markers that allowed the target to be detectable from up to 20m height, as well as up close. The behaviour of the quadrotor under the proposed control is observed in a simulator, Gazebo 7, and afterwards in the real drone, where the simulator was built with the same features as the provided drone. It was clear that the vision-based detection algorithm proved to be a very reliable option, both in simulation as well as with the real drone experiments, by providing accurate relative pose estimates under any circumstance. The implemented controller, after proper manual tuning, confirmed its validity to accurately land on a moving or stationary target, at a range of velocities, and from different trajectories of movement.

Thus, this thesis achieved, as was mentioned, successful landing on a stationary and moving target with a simple controller, proving that for less complex scenarios a more complicated approach is not necessary. Tested and proved the versatility of ArUcO markers and achieved a pattern constituted by different sized markers that demonstrated to be reliable at a wide variety of heights. Accomplished a system that is computationally light enough to allow to be run onboard with very fast reaction time. And fulfilled the purpose with the aid of an open source autopilot which allows it to be implemented on other systems besides the one tested.

7.1 Future Work

As of for future work, it would be advised to perform further tuning to the controller on the real drone, as well as test it for a moving target.

Afterwards, a linear controller can be biased to the scenario to which it was tuned, therefore it should be considered the implementation of a non-linear controller, for example a Model Predictive Control, since it would provide a controller more robust and less susceptible to change.

Furthermore, the detection of the UGV could ideally be performed without the aid of any artificial markers, and instead by detecting the UGV itself, either by using the RGBd camera to detect and land on moving targets, or by the use of machine learning algorithms such as Neural Networks or Deep Learning.

Bibliography

[1] K.-E. Åarzén. A simple event-based pid controller. *IFAC Proceedings Volumes*, 32(2):8687–8692, 1999.

[2] O. Araar, N. Aouf, and I. Vitanov. Vision based autonomous landing of multirotor uav on moving platform. *Journal of Intelligent & Robotic Systems*, 85(2):369–384, 2017.

[3] T. Baca, P. Stepan, V. Spurny, D. Hert, R. Penicka, M. Saska, J. Thomas, G. Loianno, and V. Kumar. Autonomous landing on a moving vehicle with an unmanned aerial vehicle. *Journal of Field Robotics*, 36(5):874–891, 2019.

[4] K. Backman, D. Kulić, and H. Chung. Learning to assist drone landings. *arXiv preprint arXiv:2011.13146*, 2020.

[5] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke. Training deep neural networks for visual servoing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[6] J. Bohren and S. Cousins. The smach high-level executive [ros news]. *IEEE Robotics & Automation Magazine*, 17(4):18–20, 2010.

[7] W. Bolton. *Instrumentation and control systems*. Newnes, 2021.

[8] A. Borowczyk, D.-T. Nguyen, A. P.-V. Nguyen, D. Q. Nguyen, D. Saussié, and J. Le Ny. Autonomous landing of a quadcopter on a high-speed ground vehicle. *Journal of Guidance, Control, and Dynamics*, 40(9):2378–2385, 2017.

[9] A. Cabrera-Ponce and J. Martinez-Carranza. Onboard cnn-based processing for target detection and autonomous landing for mavs. In *Mexican Conference on Pattern Recognition*, pages 195–208. Springer, 2020.

[10] J. P. C. de Souza, A. L. M. Marcato, E. P. de Aguiar, M. A. Jucá, and A. M. Teixeira. Autonomous landing of uav based on artificial neural network supervised by fuzzy logic. *Journal of Control, Automation and Electrical Systems*, 30(4):522–531, 2019.

[11] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza. Vision-based autonomous quadrotor landing on a moving platform. In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 200–207. IEEE, 2017.

[12] H. C. T. E. Fernando, A. T. A. De Silva, M. D. C. De Zoysa, K. A. D. C. Dilshan, and S. R. Munasinghe. Modelling, simulation and implementation of a quadrotor uav. In *2013 IEEE 8th International Conference on Industrial and Information Systems*, pages 207–212, 2013. doi: 10.1109/ICIInfS.2013.6731982.

[13] D. W. Gage. Ugv history 101: A brief history of unmanned ground vehicle (ugv) development efforts. Technical report, NAVAL COMMAND CONTROL AND OCEAN SURVEILLANCE CENTER RDT AND E DIV SAN DIEGO CA, 1995.

[14] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[15] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 10 2015. doi: 10.1016/j.patcog.2015.09.023.

[16] K. A. Ghamry, Y. Dong, M. A. Kamel, and Y. Zhang. Real-time autonomous take-off, tracking and landing of uav on a moving ugv platform. In *2016 24th Mediterranean conference on control and automation (MED)*, pages 1236–1241. IEEE, 2016.

[17] J. Ghommam and M. Saad. Autonomous landing of a quadrotor on a moving platform. *IEEE Transactions on Aerospace and Electronic Systems*, 53(3):1504–1519, 2017.

[18] J. E. Hopcroft, R. Motwani, and J. D. Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[19] C. Hui, C. Yousheng, L. Xiaokun, and W. W. Shing. Autonomous takeoff, tracking and landing of a uav on a moving ugv using onboard monocular vision. In *Proceedings of the 32nd Chinese Control Conference*, pages 5895–5901. IEEE, 2013.

[20] IntelRealSense. realsense-ros. `https://github.com/IntelRealSense/realsense-ros`, 2021.

[21] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios. Experimental comparison of fiducial markers for pose estimation. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 781–789. IEEE, 2020.

[22] S. Lange, N. Sünderhauf, and P. Protzel. Autonomous landing for a multirotor uav using vision. In *International conference on simulation, modeling, and programming for autonomous robots (SIMPAR 2008)*, pages 482–491, 2008.

[23] M. A. Laughton and M. G. Say. *Electrical engineer's reference book*. Elsevier, 2013.

[24] H. Lee, S. Jung, and D. H. Shim. Vision-based uav landing on the moving vehicle. In *2016 International conference on unmanned aircraft systems (ICUAS)*, pages 1–7. IEEE, 2016.

[25] S. Lee, T. Shim, S. Kim, J. Park, K. Hong, and H. Bang. Vision-based autonomous landing of a multi-copter unmanned aerial vehicle using reinforcement learning. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 108–114. IEEE, 2018.

[26] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation magazine*, 19(3):20–32, 2012.

[27] mavlink. mavros. `https://github.com/mavlink/mavros`, 2021.

[28] P. H. Nguyen, M. Arsalan, J. H. Koo, R. A. Naqvi, N. Q. Truong, and K. R. Park. Lightdenseyolo: A fast and accurate marker tracker for autonomous uav landing by visible light camera sensor on drone. *Sensors*, 18(6):1703, 2018.

[29] A. O'dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific, 2009.

[30] N. Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[31] L. E. Parker. Heterogeneous multi-robot cooperation. Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1994.

[32] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi. Toward end-to-end control for uav autonomous landing via deep reinforcement learning. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 115–123. IEEE, 2018.

[33] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. De La Puente, and P. Campoy. A deep reinforcement learning strategy for uav autonomous landing on a moving platform. *Journal of Intelligent & Robotic Systems*, 93(1-2):351–366, 2019.

[34] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 06 2018. doi: 10.1016/j.imavis.2018.05.004.

[35] ROS. executive_smach. `https://github.com/ros/executive_smach/tree/indigo-devel`, 2017.

[36] ROS. geometry. `https://github.com/ros/geometry`, 2021.

[37] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *Ieee Access*, 7:48572–48634, 2019.

[38] A. D. Team. Ardupilot. `https://ardupilot.org/dev/index.html`, 2021.

[39] UbiquityRobotics. fiducials. `https://github.com/UbiquityRobotics/fiducials`, 2021.

[40] T. Yang, P. Li, H. Zhang, J. Li, and Z. Li. Monocular vision slam-based uav autonomous landing in emergencies and unknown environments. *Electronics*, 7(5):73, 2018.

Appendix A

Code Repositories

The repositories to build the simulator as well as run the implemented code are available on GitHub.

A.1 GitHub Repositories

Table A.1: Code Repositories.

Description	Link
Github containing the solution implemented in this thesis and instructions to test it	`https://github.com/Saphira7544/IsabelCastelo_MasterThesis`
Github containing the Gazebo simulator used throughout the entire thesis	`https://github.com/durable-ist/Multi_Robot_Simulation`