Fall 11-15-2019

Vision-based Autonomous Tracking of a Non-cooperative Mobile Robot by a Low-cost Quadrotor Vehicle

Cheikhna Ahmed Tidiane Sy
The University of New Mexico

Cheikhna Ahmed Tidiane Sy

Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Rafael Fierro , Chairperson

Balu Santhanam

Fernando Moreu

Christopher D.Petersen

Vision-based Autonomous Tracking of a Non-cooperative Mobile Robot by a Low-cost Quadrotor Vehicle

by

Cheikhna Ahmed Tidiane Sy

THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Electrical Engineering

The University of New Mexico

Albuquerque, New Mexico

December, 2019

Dedication

I want to thank God Almighty for guiding and for giving me the strength when and where I needed it.

I dedicate this work to my fantastic mother Sawadogo Ouado Fatimata and my father Arona Sy for their continued encouragement, support, advice, and prayers.

To my two amazing brothers Yves François Lompo and Lamine Sy who consistently motivated me to give the best of myself.

To my adorable sister Sawadogo Aïcha who always finds the right words to make me smile.

Acknowledgements

I would like to thank my advisor, Professor Rafael Fierro, for giving me this great opportunity to perform my research in the MARHES Lab. I especially want to express my sincere gratitude for his guidance, suggestions, patience and encouragement, without which this work would not have been accomplished.

I would also like to thank my "Probability Theory and Stochastic Process" professor, Dr. Balu Santhanam, for his great insight regarding estimation and filtering theory.

I am extremely grateful to Dr. Reiner Martens for all his supports, advise and encouragements.

I cannot thank enough my friends Miko Serna, Aziz Hmaidi, Errin Madem, Mustapha Al Maashadani, Firas Ayoub, Adeoluwa Akerele, Sayf BenFraj, Ghada Zribi, Yomi Tadfor, Nziki Tadfor, Anthony Riviera, Orlando Garayburu and Wenjing Shi for their various types of help and support.

At last, words fall short of expressing my thanks and gratitude to my parents, brothers and my sister for being extremely supportive.

I would have never been able to do this without you. Thank you for being there when I needed you.

Vision-based Autonomous Tracking of a Non-cooperative Mobile Robot by a Low-cost Quadrotor vehicle

by

Cheikhna Ahmed Tidiane Sy

M.S., Electrical Engineering, University of New Mexico, 2019

Abstract

The goal of this thesis is the detection and tracking of a ground vehicle, in particular a car-like robot, by a quadrotor. The first challenge to address in any pursuit or tracking scenario is the detection and unique identification of the target. From this first challenge, comes the need to precisely localize the target in a coordinate system that is common to the tracking and tracked vehicles. In most real-life scenarios, the tracked vehicle does not directly communicate information such as its position to the tracking one. From this fact, arises a non-cooperative constraint problem. The autonomous tracking aspect of the mission requires, for both the aerial and ground vehicles, robust pose estimation during the mission. The primary and crucial functions to achieve autonomous behaviors are control and navigation. The principal-agent being the quadrotor, this thesis explains in detail the derivation and analysis of the equations of motion that govern its natural behavior along with the control methods that permit to achieve desired performances. The analysis of these equations reveals a naturally unstable system, subject to non-linearities. Therefore, we explored three different control methods capable of guaranteeing stability while mitigating non-linearities. The first two control methods operate in the linear region and consist of the intuitive Proportional Integrate Derivative controller (PID). The second linear control strategy is represented by an optimal controller that is the Linear Quadratic Regulator controller (LQR). The last and final control method is a nonlinear controller designed from the Sliding Mode Control Theory. In addition to the in-depth analysis, we provide assets and limitations of each control method. In order to achieve the tracking mission, we address the detection and localization problems using respectively visual servoing and frame transform techniques. The pose estimation challenge for the aerial robot is cleared up using Kalman Filtering estimation methods that are also explored in depth. The same estimation method is used to mitigate the ground vehicle's real-time pose estimation and tracking problem. Analysis results are illustrated using Matlab. A simulation and a real implementation using the Robot Operating System are used to support the obtained results.

TABLE OF CONTENTS

List of Figures

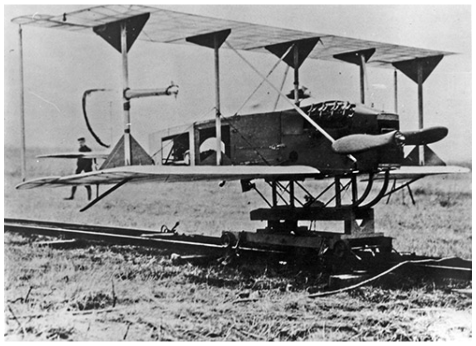
List of Tables

Chapter 1

Introduction

1.1 Background

Unmanned Aerial Vehicles (UAVs), commonly called drones, are aerial vehicles that can take off and maneuver without pilots. The primary purpose of the construction of UAVs was for war. The Austrian army first developed and used them in 1849 in the form of air balloons filled with explosives to attack Venice. UAVs took a different appearance during the first world war. The Hewitt Sperry Automated Airplane, shown in Figure 1.1(a), is considered the ancestor of present day torpedoes. It was originally developed by Elmer Sperry, an American inventor and entrepreneur, to serve as a flying bomb. The American military later automated their pursuit aircraft, the standard E-1, to serve as a drone. During WWII, technology made tremendous advancement in the domains of electronics and communication. For example, airplanes used radar technology to triangulate their position by measuring their distance from several radar signals. The LORAN (long-range navigation) system that used radar navigation was the precursor to today's satellite-based GPS technology. These technological advancements from WWII to nowadays, along with technologies such as the GPS, permitted UAVs to become more reliable and to adapt to several types of operations. Quadrotors were among the first vertical take-off and landing vehicles (VTOLs). The first quadrotor, The Omnichen 2 as shown in Figure 1.1(b), had four rotors and eight propellers all driven by one motor. It was developed in 1920 by Etienne Omnichen to solve the problem of vertical flights encountered by helicopter vehicles. In 1956 Dr. George de Bothezat and Ivan Jerome created one of the first quadrotors to use varying thrust of the four propellers to control roll, pitch, and yaw motions. After the 1960's, the creation of a new breed of quadrotors that are cheap, agile, lightweight and smart have been made possible by developments such as the use of brushless motors and the invention of advanced and efficient programming languages, like c and python. Advancements in radio transmitters and receivers and both the miniaturization and optimization of processors and controllers further contributed to the success of these new quadrotors. Autonomous comes from the Greek word 'Autonomos' that means "Having its own laws". In the realm of robotics it expresses the ability of robots to execute and achieve specific tasks with a minimum intervention of humans. For a robot to achieve autonomy, it must have a minimum awareness regarding its surrounding environment. Cameras are one of the most used sensors in robotics via computer vision applications to provide vision based sensing. Computer vision appeared in 1960's and has become a specialized branch of Artificial Intelligence, focused on enabling machines to visually perceive the world and respond to it. It is a process of taking in visual information and analyzing and processing that information in the purpose to correctly identify objects contained within said information. Figure 1.2(b) shows an example of accurate features detection. Thanks to the advances in the field of Computer Vision, and the significant increases in available computing power, machines can now "see" thousands of images, and process them far more rapidly and accurately than a human could ever do. Figure 1.2(a) illustrates an example of hardware setup up for computer vision purposes.


(a) Hewitt Sperry Automated Airplane

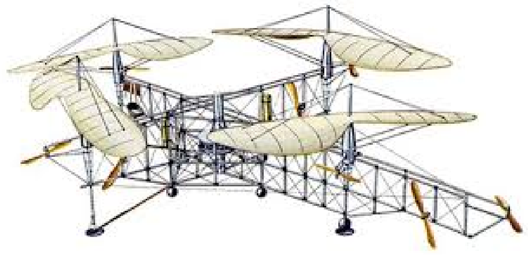
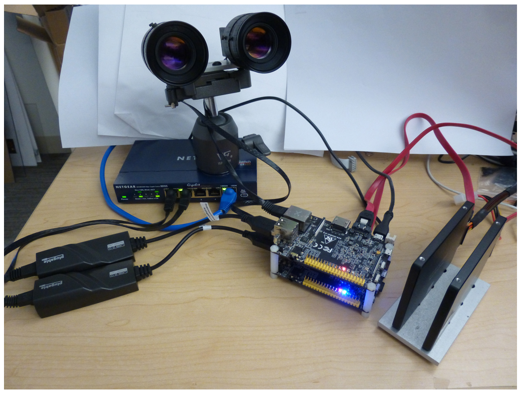
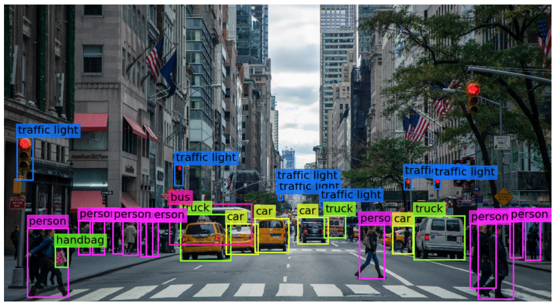

(b) Omnichen 2

Figure 1.1: Early UAVs.


(a) Drive computer vision camera on single-board computer



(b) Features detection using Computer Vision

Figure 1.2: Computer Vision and Applications.

1.2 Motivation

Security is one of the realms where drones can be highly useful. For instance, they can be used to assess dangers by finding gaps and vulnerabilities that would not normally be seen. They can also allow the remote surveillance and control of a defined area. Police pursuit by law enforcement agencies in the United States is controversial due to a high number of injuries and deaths. From 1994 to 2002, according to [1], there were 2,654 crashes involving 3,965 vehicles that lead to 3,146 fatalities among which 1,088 involved people that were not in the fleeing vehicle. The high speed of the vehicles, combined with their numbers, considerably increases the risk of deadly crashes and collateral damage. The risk of crashes due to the number of vehicles involved in these pursuits could be mitigated by an automated aerial vehicle used to detect and follow the fleeing one while continuously updating the escaping target position to police officers who would be able to efficiently plan a trap. Another example where automated surveillance could be an asset are open areas with a high density of buildings such as campuses. The 2018 annual clery report of the University of New Mexico [2] reveals 15 cases of aggravated assaults for 2015, 12 for 2016 and 19 for 2017; 88 cases of motor vehicle theft for 2015, 174 for 2016 and 222 for 2017. In these cases, an automated security system could be implemented that would generate an alert, remotely guide an UAV to the crime scene, and detect and track the suspect. This would provide real time situational awareness to campus law enforcers which would improve their choices in reactivity and planning. One of the best choices of UAVs for this type of mission is without doubt the quadrotor. The quadrotor helicopter, or quadcopter, uses four equally spaced rotors spinning at high speed to push air downwards in order to create a thrust force to maintain and navigate the vehicle in the air. Quadrotor motion can occur in six different directions, meaning that it has six degrees of freedom (three translations and three rotations). One of the main assets of quadrotors is their mechanical simplicity. Traditional helicopters use a main rotor and a tail rotor to maneuver in the air. Displacement in any direction is achieved via a complex cyclic-pitch mechanism; the goal of which is to adjust the pitch of the main motor blades as they spin around, which vectors its thrust. Quadrotors are mechanically simpler; they use four motors connected to four fixed-pitch propellers to produce motion in six possible directions. On a small scale, their mechanical simplicity makes them more versatile and suitable than helicopters. Our work is directly related to surveillance and addresses the problems of localization, pose estimation and real time tracking using a low cost quadrotor associated with control and estimations techniques.

1.3 Problem Statement

The main application of this thesis is related to surveillance with the quadrotor as the main agent. The challenge consists of being able to detect a defined target and estimate both its actual and future positions, allowing us to track it with visual data as the only source of direct information. The problem can be presented in the following steps:

1. **Target detection:** The main challenge in this step is the identification of the ground-vehicle in an outdoor 3-D environment using the quadrotor's on-board camera.

2. **Target relative pose estimation:** This second challenge is expressed by the need to extract the position and orientation of the mobile robot relative to the quadrotor.

3. **Target tracking:** This third and last challenge is for the quadrotor vehicle to be able to follow the detected target considering its natural motion while taking into account the fact that partial occlusion and even loss of detection can occur during the quadrotor's motion. In fact, any displacement of the quadrotor in the X-Y plane necessitates a preliminary rotation about its X or Y axis. The down-facing camera being attached to the quadrotor frame, any rotation of the frame leads to a rotation of the camera which can result in a lost detection.

The three above steps are further explored and reveal other challenges. Target detection and relative pose estimation utilize computer vision algorithms to process images in order to extract information such as position and orientation. It requires lot of computational power, which in most cases is available on heavy computers, yet not suitable for on-board processing. Target tracking requires a great pose estimation system to provide in real time an accurate position of the quadrotor. It also needs a good estimation of the tracked object information such as position, velocity and acceleration; as well as an efficient control method to both stabilize the quadrotor and perform the pursuit. The first two problems are addressed utilizing the concept of machine vision by using a model based tracker to detect and obtain the mobile vehicle pose relative the on-board camera. A companion computer is used to allow on-board processing. The approach to solve the third and last problem consists of first using frame transformation properties to obtain the pose of the target relative to the camera into the frame in which the position of the quadrotor is defined; and secondly, continuously control the quadrotor to this desired position. The constraints of partial occlusion and loss of tracking is mitigated by considering small pitch and roll angles and by using estimation theories to predict the future pose of the mobile robot based on its actual one and its velocity. The second chapter of this thesis presents a literature review that illustrates the challenges related to objects tracking using quadrotors in both indoor and outdoor environments. It includes the state of art and particular solutions used to solve specific problems. The third chapter is dedicated to the quadrotor system modeling. Chapter four provides a deep analysis of the main-agent (Quadrotor) system along with some control methods and states estimation strategies in both the linear and nonlinear realm. Chapter five presents visual servo control theories used for the target detection and tracking, formulates the problem and uses all these explored techniques to achieve our target tracking mission. It also includes a simulation using a 3D graphic simulator and a physical implementation, along with their respective results and limitations.

Chapter 2

Literature Review

In the past years, much research has been conducted to allow autonomous UAVs to track known targets. The challenges are to be able to control and stabilize the aerial vehicle, acquire the target position and efficiently lead the aerial vehicle to a desired relative position vis-à-vis to the designated target. The control challenge especially for quadrotor vehicles has been mitigated by the efficient design of control strategies [3], [4], [5], [6], [7], [8] and the quick improvement of autopilots [9], [10] [11]. The problem of target tracking is directly related to computer vision algorithms and has been addressed via techniques such as visual servo control [12], [13]. Vision based control techniques are computationally expensive due to the large amount of data relative to images and videos that have to be processed. [14], implemented a method capable of successfully tracking a moving target using compressed H.264 video streamed to a ground station by a WiFi camera attached to a quadrotor. In the same work, the ground station computer is in charge of executing the image processing algorithm to estimate the object position. The same ground station also computes and sends back orientation commands to the quadrotor via WiFi. With the development of tiny computers such as the raspberry pi, it is now possible to obtain additional computational power on the quadrotor. [15] developed an approach to track a static object with an on-board image system coupled with an Inertial Measurement Unit (IMU) without any external localization sensor or GPS. [16] presents a method to track a moving ball using a single down-facing camera and an IMU. The article presents two ingenious techniques to obtain the pose of a sphere relative to the camera knowing its radius to solve the problem of partial occlusion. It also includes a planning algorithm considering the dynamics of the quadrotor, the actuators limitations and the field of view constraints. Our work is dedicated to outdoor autonomous flight and all processing is done on-board using a raspberry Pi as companion computer. We solve the problem of localization by using an Extended Kalman Filter (EKF) that fuses the data of the Global Positioning system (GPS) and the Inertial Measurement Unit (IMU). The problem of tracking is addressed using visual servoing techniques coupled with pose transformation concepts along with another Extended Kalman Filter to predict the motion of the ground vehicle in the case of loss of tracking.

Chapter 3

Quadrotor System Modeling

The quadrotor helicopter uses four equally spaced rotors spinning at high speed to push air downwards to create thrust and maintain the vehicle in the air. Quadrotors motion can occur in six different directions implying that it has six degrees of freedom (three translations and three rotations). One of the main assets of quadrotors is their versatility. They can be controlled via the angular speed of their four independent motors. Quadrotors are classified as under-actuated vehicles because they use their four motors to produce motion in six possible directions. This fact represents a challenge, and at the same time, an exciting problem for control engineers. Modeling is the first process in the quest to understand and control any system. This chapter is dedicated to the quadrotor vehicle modeling [17] [18].

3.1 Coordinate System and Pose Representation

A coordinate system, also called frame, is a method used to localize a point or an object on the earth. The pose of an object represents its position and orientation relative to a fixed reference also called origin.

3.1.1 Coordinate System

The first step in the modeling process is defining the coordinate systems in which the vehicle will be operating. In navigation, guidance, and control of an aircraft or rotorcraft, there are several coordinate systems used in design and analysis. They are used to describe the position and orientation of our vehicle. We will be working with two main coordinate frames (Figure 3.1) that are:

- Inertial Frame:

  The inertial frame is an earth-fixed set of axis used as a static reference. It can be seen as the home location of the vehicle. We will use the common aeronautical inertial frame described as follows: the x-axis pointing to the north, the y-axis pointing to the east and the z-axis pointing down(NED).

- Body Frame:

  The body frame is the coordinate system that is aligned with the body of the quadrotor. The x-axis is the one indicating the nose of the quadrotor and used to characterize the forward and backward movements of the vehicle, while the y-axis is the one that is used to characterize the lateral movements (left, right movements of the vehicle). We are using the NED convention, therefore the z-axis of the body frame is defined to point downward meaning that altitude above the ground is negative.

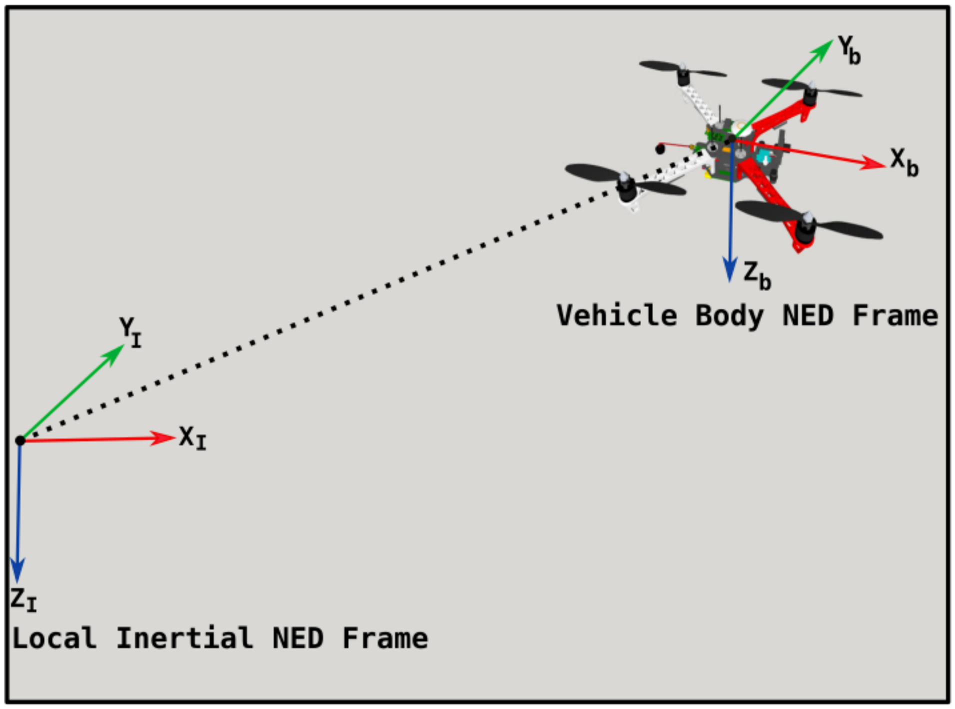


Figure 3.1: Quadrotor Body Frame and Inertial Frame.

3.1.1.1 Theoretical aspects of Position and Orientation

An important starting point in any robotics and computer vision system modeling is the representation of positions and orientations of objects in the real world [19]. A coordinate frame is a set of orthogonal axes which intersect at an origin. Position and orientation of an object's coordinate frame is known as its pose and can be illustrated as a set of coordinate axes. A system can be made of multiple elements each having a coordinate frame. In this case, it is always possible to obtain the pose of one object related to another one. This is called relative pose. The absolute pose is the position of an element with respect to the inertial coordinate.

In the following, we denote by:

ξ: The relative pose of a frame with respect to a reference frame.

${}^{A}\xi_{B}$:The relative pose describing the frame B with respect to the frame A

$\oplus$: Operator used to compose poses with the following rules.

- 0 is the neutral element representing the zero relative pose:

  $\xi \oplus 0 = \xi$

  $\xi \ominus 0 = \xi$

  $\xi \ominus \xi = 0$

  $\ominus\xi \oplus \xi = 0$

- The inverse of a pose exists

  $\ominus {}^{A}\xi_{B} = {}^{B}\xi_{A}$

- Composition:

  ${}^{A}\xi_{B} \oplus {}^{B}\xi_{C} = {}^{A}\xi_{C}$

- The operation is not commutative:

  $\xi_A \oplus \xi_B \neq \xi_B \oplus \xi_A$ except in the case where $\xi_A \oplus \xi_B = 0$

1. Point Description

   Using a reference coordinate frame it is possible to represent the displacement of a point with respect to this reference frame, with any points being described by a coordinate vector. This is illustrated in the Figure 3.2(a).

2. Object Description

   In this case, it is important to assume that the object is rigid so that each point has a fixed relative position to the frame describing the object position and orientation. This case is illustrated by the Figure 3.2(b) and represents a set of points within an object. The object being composed of multiple points, it is more suitable to describe the position and orientation of the object using its coordinate frame notably B than to describe each of the points within it.

3. Coordinate frame Composition:

   Let's consider the Point M in the Figure 3.2(C) it can be described with respect to any of the three frames $\{A\}$, $\{B\}$ and $\{C\}$, using respectively the vectors, $\vec{A_M}$, $\vec{B_M}$ and $\vec{C_M}$. Using poses, vectors transformation and frames composition, it is always possible to express the motion of any point in any particular coordinate frame:

   - Transformation of vector:

     $$\vec{A_M} = {}^A\xi_B.\vec{B_M}$$

   - Composition of frame plus Transformation of vector:

     $$\vec{A_M} = ({}^A\xi_B \oplus^B \xi_C).\vec{C_M}$$

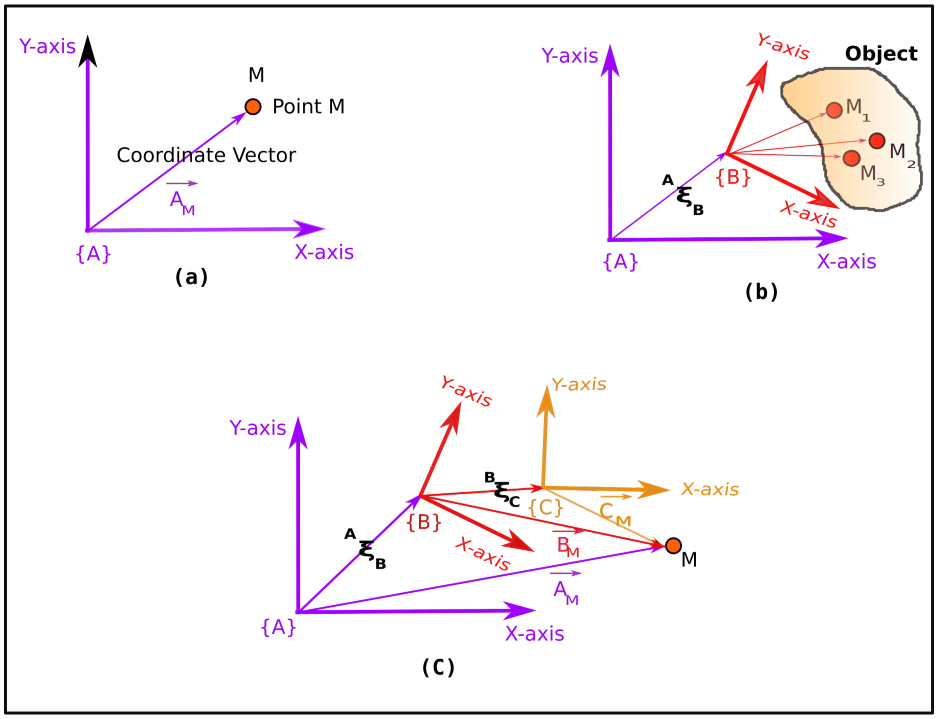


Figure 3.2: Frames representation.

3.1.2 Poses Representation:

In the mathematical point of view, the poses (ξ) are considered as a group. In fact, it can be seen as an algebraic structure with an operator ($\oplus$) having a neutral element (the zero relative pose (0)), having an inverse denoted by ($\ominus$) and satisfying the concept of closure meaning that the result of the operation between two poses is also a pose. This group belongs to the special Euclidean group referred to SE(2) for two dimensions and SE(3) for three dimensions.

1. Representation in 2-dimensions:

   Many robots, such as mobile robots, evolve in the 2D realm. This environment can be pictured using a simple Cartesian system of coordinates with two orthogonal axis. The classic configuration is a x-axis representing the horizontal direction and a y-axis representing the vertical one intersecting each other at a point called the origin. In this configuration any point can be described by its x and y coordinates or as a bound vector. Distance is an important measurement to define displacement so it is more suitable to work in a normed vector space. The latter fact implies that each of our axis will have a unit vector. Let's define $\hat{x}$ the unit vector along the x-axis and $\hat{y}$ the one along the y-axis. Any point M can be defined as follows:

   $$M = x\hat{x} + y\hat{y} \tag{3.1}$$

   As mentioned above, we have seen that we can use relative poses composition (ξ) and an operator ($\oplus$) to express points in any frame. Figure 3.3(a) illustrates two main frames: $\{A\}$ and $\{B\}$. The brown frame $\{T\}$ is the frame $\{A\}$ translated by a vector $\vec{t}$. The frame $\{B\}$ is obtained by rotating the brown frame $\{T\}$ by an angle θ. This tell us that the representation of any pose can be achieved using translation and rotation.

   - Rotation

     Let's denote by $\hat{x}_T$ and $\hat{y}_T$ the unit vectors of the the frame T and by T_x and T_y the x and y coordinates of the point M related to frame T. The point M can be described in this frame by:

$$T_M = T_x\hat{x}_T + T_y\hat{y}_T = \begin{bmatrix}\hat{x}_T & \hat{y}_T\end{bmatrix}\begin{bmatrix}T_x \\ T_y\end{bmatrix} \tag{3.2}$$

The frame $\{B\}$ is a simple rotation by an angle of θ of frame $\{T\}$, therefore we can use basic trigonometric rules to express the unit vector of frame $\{B\}$ in function of the one of frame $\{T\}$ that give us the following expressions:

$$\hat{x}_B = \cos\theta\hat{x}_T + \sin\theta\hat{y}_T \tag{3.3}$$

$$\hat{y}_B = -\sin\theta\hat{x}_T + \cos\theta\hat{y}_T \tag{3.4}$$

In matrix form:

$$\begin{pmatrix}\hat{x}_B & \hat{y}_B\end{pmatrix} = \begin{bmatrix}\hat{x}_T & \hat{y}_T\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix} \tag{3.5}$$

Expressing the point M with respect to the frame $\{B\}$

$$B_M = B_x\hat{x}_B + B_y\hat{y}_B = \begin{bmatrix}\hat{x}_B & \hat{y}_B\end{bmatrix}\begin{bmatrix}B_x \\ B_y\end{bmatrix} \tag{3.6}$$

Considering equation(3.5) we obtain:

$$B_M = \begin{bmatrix}\hat{x}_T & \hat{y}_T\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}B_x \\ B_y\end{bmatrix} \tag{3.7}$$

Considering the right-hands sides of equations (3.7) and (3.2) we finally have:

$$\begin{bmatrix} T_x \\ T_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} B_x \\ B_y \end{bmatrix} = {}^{T}R_B \begin{bmatrix} B_x \\ B_y \end{bmatrix} \tag{3.8}$$

The matrix ${}^{T}R_B = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ permits to express any point in the frame $\{B\}$ into the frame $\{T\}$. It is called the rotation matrix. This matrix is orthogonal, meaning that the columns are orthogonal and each of them is a unit vector representing the unit vectors defining the frame {B} with respect to the frame {T}. Another interesting property related to this matrix is that its determinant is $+1$.
The latter fact means that the rotation matrix belongs to the special orthogonal group of dimension 2 ($R \in SO(2) \subset \mathbb{R}^{2\times 2}$). The same fact also tells us that the rotation preserves the length of the vectors being rotated $|B_M| = |T_M|$, $\forall\theta$. Another important property of this matrix is that the inverse of the rotation matrix is just its transpose($R^{-1} = R^{T}$).

$$\begin{bmatrix} B_x \\ B_y \end{bmatrix} = ({}^{T}R_B)^{-1} \begin{bmatrix} T_x \\ T_y \end{bmatrix} = ({}^{T}R_B)^{T} \begin{bmatrix} T_x \\ T_y \end{bmatrix} = ({}^{B}R_T) \begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{3.9}$$

- Translation:

  The translation is represented by the vector $\vec{t}$. The axis of the frames $\{A\}$ and $\{T\}$ are parallel, thus the position of the point M related to the frame $\{A\}$ can be obtained as follows:

$$\begin{bmatrix} A_x \\ A_y \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.10}$$

Considering the rotation motion described before we can rewrite the equation as follows:

$$\begin{bmatrix} A_x \\ A_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} B_x \\ B_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ 1 \end{bmatrix} \tag{3.11}$$

Using an homogeneous transformation, we have:

$$A_{\bar{M}} = \begin{bmatrix} A_x \\ A_y \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{A}R_B & t \\ 0_{1\times 2} & 1 \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{T}R_B & t \\ 0_{1\times 2} & 1 \end{bmatrix} B_{\bar{M}} \tag{3.12}$$

$$A_{\bar{M}} =^{A} T_B.B_{\bar{M}} \tag{3.13}$$

With:

t=(x,y): The translation from the frame $\{A\}$ to the frame $\{B\}$

${}^{A}R_B$=${}^{T}R_B$: The orientation.

$A_{\bar{M}}$: The homogeneous form of the vector A_M.

$B_{\bar{M}}$: The homogeneous form of the vector B_M.

${}^{A}T_B \in SE(2) \subset \mathbb{R}^{2\times 2}$: The homogeneous transformation representing the relative pose between the frame $\{A\}$ and $\{B\}$ and belonging to the special Euclidean group of dimension 2.

$$A\xi_B(x,y,\theta) \sim \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

2. Representation in 3-dimensions:

   Any point in this configuration can be represented by is three coordinates (x,y,z) or by a bound vector as follows:

$$M = x\hat{x} + y\hat{y} + z\hat{z} \tag{3.15}$$

   With:

   $\hat{x}$, $\hat{y}$, $\hat{z}$ the respective unit vectors of the x-axis, y-axis and z-axis.
   Similarly to the 2D case, any pose is represented by a translation and a rotation, the difference being the additional z-coordinate, that is orthogonal to the already orthogonal two coordinates axis (x-axis and y-axis).
   Rotation in 3D environment is more delicate than the 2D case. There are several methods to represent objects rotation in 3D such as, Euler and Cardan angles, unit quaternions, dual quaternions [2].We will work with Euler's representation, for it is more intuitive. The upcoming subsection related to the quadrotor kinematics will present in detail this 3D rotation.

   Euler's Rotation Theorem:

   Any rotation can be considered as a sequence of rotations about different coordinate axes. Euler angles use a combination of three successive rotations about axes with no two consecutive rotations about the same axis [20]. It is also called the three-angles representation and can be divided into two models.

- Eulerian:

This type allows repetition of rotations, yet not successive about one of the axis. It results in 6 possibles sequences that are:

XYX,XZX,YXY,YZY,ZXZ,ZXY.

- Cardanian:

  This sequence doesn't allow any repetition of rotations about any of the three axes and is described as followed:

  XYZ,XZY,YXZ,YZX,ZXY,ZYX.

Let's define the orthogonal matrix ${}^{A}R_{B}$ which rotates a vector with respect to frame $\{B\}$ to a vector with respect to frame $\{A\}$.The relative pose is obtained like in the 2D case by combining rotation and translation.

$$A_{\bar{M}} = \begin{bmatrix} A_x \\ A_y \\ A_z \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{A}R_{B} & t \\ 0_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{T}R_{B} & t \\ 0_{1\times 3} & 1 \end{bmatrix} B_{\bar{M}} \tag{3.16}$$

$$A_{\bar{M}} =^{A} T_{B}.B_{\bar{M}} \tag{3.17}$$

with:

$t = (x, y)$: The translation from the frame $\{A\}$ to the frame $\{B\}$

${}^{A}R_{B}$=${}^{T}R_{B}$: The orientation.

$A_{\bar{M}}$: The homogeneous form of the vector A_{M}.

$B_{\bar{M}}$: The homogeneous form of the vector B_{M}.

${}^{A}T_{B} \in SE(3) \subset \mathbb{R}^{4\times 4}$: The homogeneous transformation representing the relative pose between the frame $\{A\}$ and $\{B\}$ and belonging to the special Euclidean group of dimension 2.

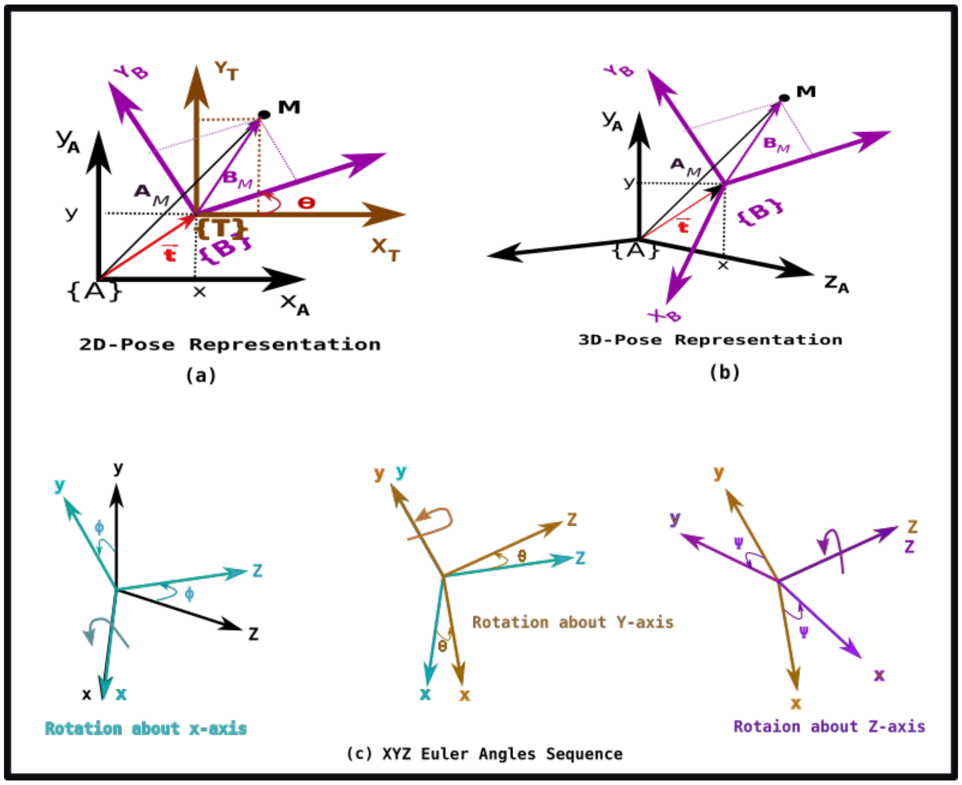


Figure 3.3: 2D and 3D Representations.

As stated above, the quadrotor vehicle consists of four independent brushless motors each having a rotor with a fixed pitch. In order to properly balance the vehicle, the motors must be arranged in a specific way. The most common configurations are the “+” and “X” configurations (Figure 3.4). where motors 1 and 2 rotate counterclockwise while motors 3 and 4 rotate clockwise. This configuration guarantees the elimination of the rotating moment produced by the torque of the motors; avoiding the vehicle to yaw when all the motors spin at the same speed, it also results in the reaction torque from the pairs of adjacent motors to be exactly opposed if they are all spinning at the same speed. We assume the “Plus” configuration for the following subsections.

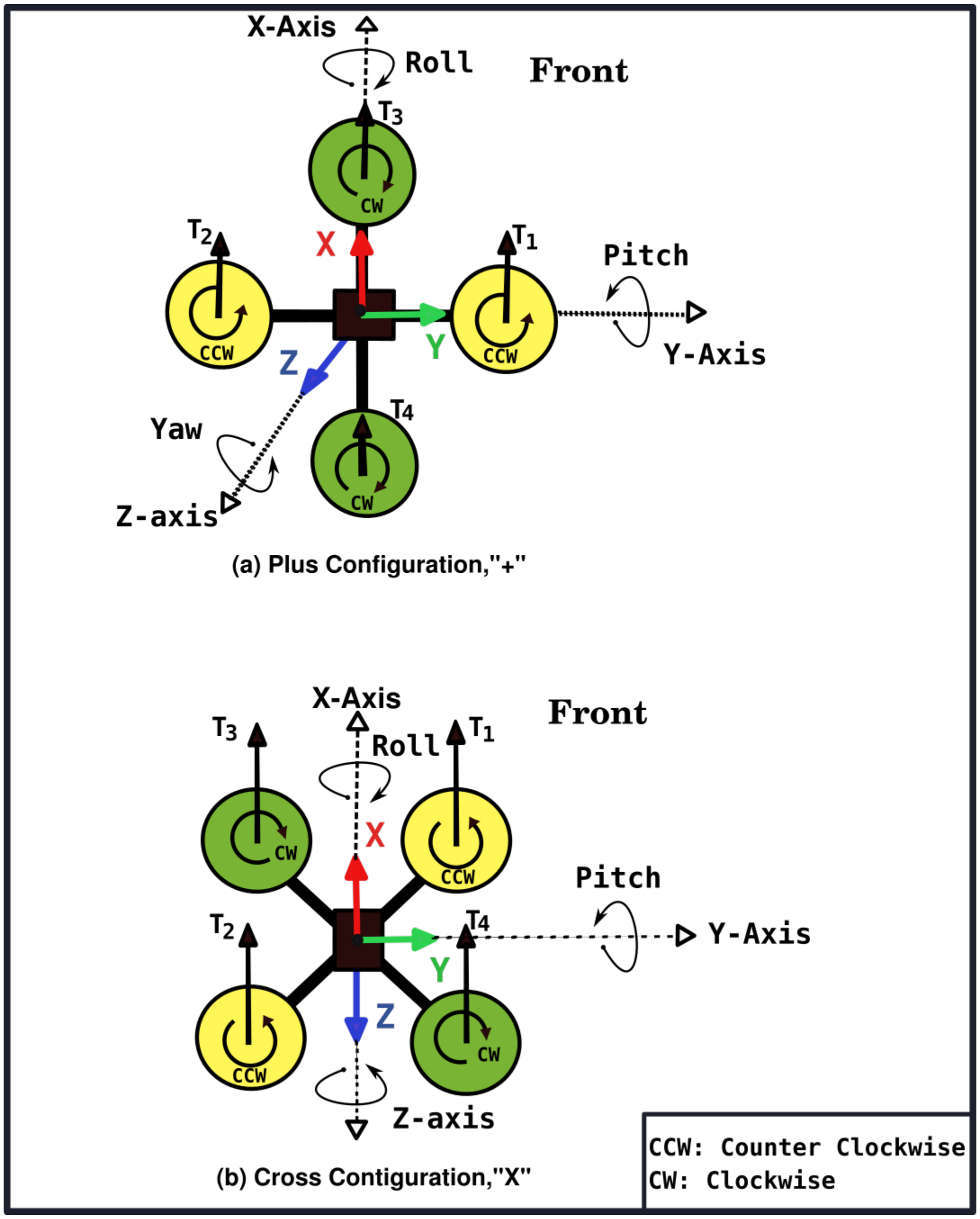


Figure 3.4: Quadrotor plus (+) and cross (X) Configurations.

3.2 Kinematics

Kinematics is the branch of mechanics concerned on the motion of points, objects, and groups of objects without considering the causes that create those motions (forces involved). This part aims to illustrate the motion of the vehicle in the body and inertial frame. We are especially interested on the position, velocity, attitude and angular velocity of the quadrotor. The derivation of the kinematics model will be divided in two parts: the translational kinematics whose goal is to provide a means to transform variables between the inertial and body frames and the rotational kinematics that provides a tool to relate the quadrotor angular rates to the Euler angles.

3.2.1 Translational Kinematics

Let's define the following vectors:

$x = (x, y, z)^T$: The position of the vehicle.
$\dot{x} = (\dot{x}, \dot{y}, \dot{z})^T$: The velocity of the vehicle.
$\theta = (\phi, \theta, \psi)^T$: Roll (rotation about the x-axis), pitch (rotation about the y-axis) and yaw (rotation about the z-axis).
$\dot{\theta} = (\dot{\phi}, \dot{\theta}, \dot{\psi})^T$: The roll, pitch and yaw respective derivatives.
$w = [w_1, w_2, w_3]^T$ The angular velocity vector.

Position state variables are expressed in the inertial frame while velocity ones are in the body frame. Because of this fact, we need a tool to convert these variables in either of the frames. This is done using the rotation matrix introduced in the previous section. The rotation matrix is obtained as follows:

- The orthogonal matrix for rotation of ϕ about the x-axis:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.18}$$

- The orthogonal matrix for rotation of θ about the y-axis:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.19}$$

- The orthogonal matrix for rotation of ψ about the z-axis:

$$R_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

Let's define two frames that are in addition to the inertial and body frame: the vehicle-1 and vehicle-2 frames used to represent the vehicle after the first and second rotations. We are using the Eulerian XYZ sequence.

- Roll rotation leading to Vehicle-1 frame:

  Considering that we start in the inertial frame, this rotation produces a new frame with the y-axis and z-axis, rotated by a roll angle ϕ. ${}^{I}R_{v_1}(\phi) = R_x(\phi)$ defined above.

- Roll and Pitch rotations leading to Vehicle-2 frame:

  This configuration is the result of the rotation about the y-axis of the vehicle-1 frame. The rotation is obtained using the matrix ${}^{v_1}R_{v_2}(\theta) = R_y(\theta)$ defined above. The rotation from the inertial frame to the vehicle-2 frame is obtained as follows:

$${}^{I}R_{v_2}(\phi,\theta) =^{I} R_{v_1}(\phi)^{v_1}R_{v_2}(\theta)$$

$${}^{I}R_{v2}(\phi,\theta) = \begin{bmatrix} cos(\theta) & 0 & -\sin(\theta) \\ \sin(\phi)\sin(\theta) & \cos(\phi) & \cos(\theta)\sin(\phi) \\ cos(\phi)\sin\theta & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{3.21}$$

- Roll,Pitch and Yaw rotations leading to body frame:

  This frame is obtained by executing a rotation about the z-axis (yaw) from the intermediate vehicle-2 frame. The rotation is obtained via the rotation matrix $R_z(\psi) =^{v2} R_{_B}(\psi)$. The total rotations leading from the inertial frame to the body frame are given by:

$$^I R_B(\phi,\theta,\psi) =^I R_{v_1}(\phi)^{v_1} R_{v_2}(\theta)^{v_2} R_{_B}(\psi)$$

$$\begin{bmatrix} \cos\psi\cos\theta & \cos\theta\sin\psi & -\sin\theta \\ -\cos\phi\sin\psi + \cos\psi\sin\phi sin\theta & \cos\phi\cos\psi + \sin\phi\sin\psi\sin\theta & \cos\theta\sin\phi \\ \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta & \cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{3.22}$$

The transformation from the body frame to the inertia frame is just the inverse of the matrix R and is obtained as stated previously by just applying a transpose operation.

$$^B R_{_I}(\phi,\theta,\psi) = (^I R_B(\phi,\theta,\psi))^T$$

$$^B R_I(\phi,\theta,\psi) = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ cos\theta\sin\psi & \sin\phi\sin\psi\sin(\theta) + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix} = R \tag{3.23}$$

3.2.2 Rotational Kinematics

The rotational kinematic aims to relate the angular rates (expressed in the body frame) and the time derivative of the Euler angles (expressed in the vehicle-1 and vehicle-2 frames). This is done using the three rotation matrix $R_x(\phi), R_y(\theta), R_z(\psi)$, with the assumption that the time derivative of each Euler rate is small.

$$w = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = R_x(\phi)R_y(\theta)\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R_x(\phi)\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{3.24}$$

$$w = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.25}$$

Respectively :

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}^{-1} w \tag{3.26}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos\theta} & \frac{\cos(\phi)}{\cos\theta}) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.27}$$

3.3 Dynamics

In opposite to the previous part, dynamics is closely related to the forces involved in the motion of a point or an object. This is the gateway to understand and properly model our system.

3.3.1 Motors

The quadrotor vehicle is equipped with 4 motors whose goal is to convert the electrical power acquired from the battery to produce torque and thrust forces used to control and navigate the platform. This type of motor has the advantage of working in DC, to be controlled electronically via a closed loop controller (electrical speed controller) that provides pulses of current to the motor windings. Another important advantage of this type of motor is the high power to weight ratio and the ability to work with high speed.

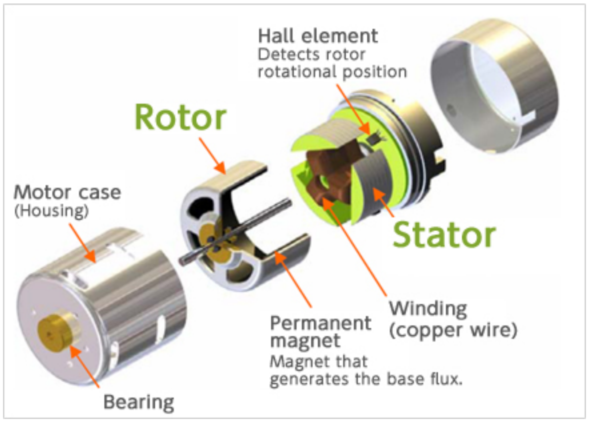


Figure 3.5: Brushless Motor Structure.

3.3.1.1 Torque and Power

The torque, or moment of force, is the cross product of the position vector and the force vector $\vec{\tau} = \vec{r} \times \vec{F}$. With $\vec{F}$ the force vector and $\vec{r}$ the vector from the origin of the defined coordinate system to the point where the force is applied. It represents a twist (rate of change of angular momentum) to an object and can be by analogy seen as the equivalent of linear force (push or pull).

The torque produces by every motor is given by:

$$\tau = K_t(I - I_0) \tag{3.28}$$

With

- I_0 : The current with no load on the motor.
- I : The input current
- K_t : The torque proportionally constant.

In order to obtain the motor's power consumption, we need to define the voltage drop across the motor. This voltage can be found using the following expression:

$$V = IR_m + K_v w \tag{3.29}$$

With

- I : The input current
- K_v : A proportionality constant related to the back-EMF generated by RPM
- w : The angular velocity of the motor
- R_m : The motor Resistance

The power consumed by a motor is given by $P = VI$. Using the torque and voltage equations(3.28 and 3.29) we get:

$$P = \frac{(\tau + k_t I_0)(k_t I_0 R_m + \tau R_m + K_t K_v w)}{K_t^2}. \tag{3.30}$$

Assuming a very small resistance ($R_m \approx 0$) and $K_t I_0 << \tau$ the power equation can be reduced to:

$$P = \frac{(\tau w) K_v}{K_t} \tag{3.31}$$

3.3.2 Torques and Forces

Flying robots are actuated by forces. The consequence of the previous statement is that their motion is highly linked to the forces and torques involved in their model. Here we derive the different forces and torques involved in the motion of the quadrotor and thus the different equations of motion.

3.3.2.1 Forces

The thrust force is the one responsible to push our quadrotor in the air. This force is directly related to the power produced by each motor.

$$P = T v_h \tag{3.32}$$

With :

- T : The thrust.
- v_h : The air velocity.

Assuming that v_h is the air velocity when hovering (low air speed) and that air around the quadrotor is stationary relative to the vehicle, we can express v_h as a function of thrust:

$$v_h = \sqrt{\frac{T}{2\rho A_r}} \tag{3.33}$$

With:

- T : The thrust.
- ρ : The density of the surrounding air.
- A_r : The area swept out by the rotor.

Using the previous expressions of the power (equations (2.28) and (2.29)), we get the following expression:

$$P = \frac{(\tau w)K_v}{K_t} = \frac{(Tw)K_v K_\tau}{K_t} = \frac{T^{\frac{3}{2}}}{\sqrt{2\rho A_r}} \tag{3.34}$$

With:

- K_τ : A proportionality constant related to the blade configuration and parameters. It comes from the previous definition of the torque $\vec{\tau} = \vec{r} \times \vec{F}$ expressing the fact that the torque is proportional to the thrust by the ratio K_τ ($\tau = K_\tau T$).

  From the previous equations it is easy to determine the trust that is given by:

$$T = Kw^2 \tag{3.35}$$

With

$$K = \Big(\frac{K_v K_\tau \sqrt{2\rho A_r}}{K_t}\Big)^2 \tag{3.36}$$

The constant $K > 0$ is called the thrust or lift coefficient and depends on air density, the cube of the rotor blade radius, the number of blades and the chord length of the blade. Notice that the thrust force is applied along the z-direction. The total thrust produced by the 4 motors and applied to our vehicle in the body frame is therefore:

$$T_B = \sum_{i=1}^{4} T_i = K \begin{bmatrix} 0 \\ 0 \\ \sum {w_i}^2 \end{bmatrix} \tag{3.37}$$

Another important force acting on our vehicle is the friction. For purpose of simplicity, it is assumed that it is just proportional to the linear velocity in each direction. The drag force applied to our vehicle in the global frame is given by:

$$F_D = \begin{bmatrix} -K_{dx}\dot{x} \\ -K_{dy}\dot{y} \\ -K_{dz}\dot{z} \end{bmatrix} = \begin{bmatrix} K_{dx} & 0 & 0 \\ 0 & K_{dx} & 0 \\ 0 & 0 & K_{dz} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{3.38}$$

With $K_{dx},\ K_{dy},\ K_{dz}$ some constants depending on the same factors as K.

3.3.2.2 Torques

The torque, permits the spinning of the propellers attached to the motors with the z-component directly involved in the creation of the thrust. The torque must be able to provide instantaneous angular acceleration to overcome the frictional drag force. The fictional force is given by:

$$F_D = \frac{1}{2}\rho C_D A_p v^2 \tag{3.39}$$

With:

- ρ : The surrounding fluid density.
- C_D: A dimensionless constant.
- A_p: The propellers cross-section.

This force permits to evaluate the torque due to drag. Since $\tau = r \times F$ we get the following equation:

$$\tau_D = \frac{1}{2}R\rho C_D A_p v^2 = \frac{1}{2}R\rho C_D A_p (wR)^2 = bw^2 \tag{3.40}$$

With:

- w: The angular velocity.
- R: The propeller radius.
- b: The drag coefficient that depends one the same factor as K.

For each motor, the total torque can be expressed as follows:

$$\tau_z = bw^2 + I_M \dot{w} \tag{3.41}$$

With:

- $\dot{w}$: The angular acceleration of the propeller.

- I_M: The moment of Inertia about the z-axis

Assuming Hovering state, where propellers will be maintaining a almost constant thrust (No acceleration), $\dot{w} \approx 0$ equation (3.41) can be simplified as follows:

$$\tau_z = (-1)^{i+1} b w_i^2 \tag{3.42}$$

With:

- $(-1)^{i+1}$ is positive for propellers spinning clockwise and negative for propellers spinning counterclockwise.

The yawing torque or total torque about the z-axis is given by:

$$\tau_\psi = b(w_4^2 - w_2^2 + w_3^2 - w_1^2) \tag{3.43}$$

The rolling torque, torque about the vehicle x-axis can be obtained via:

$$\tau_\phi = dT_3 - dT_4 = dK(w_1^2 - w_2^2) \tag{3.44}$$

The pitching torque, torque about the vehicle y-axis can be obtained via:

$$\tau_\theta = dT_1 - dT_2 = dK(w_3^2 - w_4^2) \tag{3.45}$$

With:

- d: The distance from the the motor to the center of mass.

- w_i: The different rotor speeds.

The torques in the body frame can all be summarize as follows:

$$\tau_B = \begin{bmatrix} dK(w_1^2 - w_2^2) \\ dK(w_3^2 - w_4^2) \\ b(w_4^2 - w_2^2 + w_3^2 - w_1^2) \end{bmatrix} \tag{3.46}$$

3.4 Equation of Motions

3.4.1 Translational Equations of Motion

The translational dynamics of the vehicle are better expressed in world coordinates (inertial frame) and are given by the Newton's second law. Any acceleration in this frame is due to three components which are gravity, linear friction and thrust. Equation (3.37) expresses the thrust force in the body frame, yet we are operating in the inertial frame. Therefore we need to express this latter in the inertial frame. This is done by using the rotation matrix R derived in equation (3.23). We obtain the following linear motion.

$$m\ddot{X} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + RT_B + F_D \tag{3.47}$$

With :

- $\ddot{X}$: The state vector representing the acceleration is the 3 directions (x, y, z).
- g: The acceleration due to gravity.
- R: Rotation matrix from the Body to the inertial frame.
- T_B: Thrust force in the body frame.
- F_D: Drag force.

3.4.2 Rotational Equations of Motion

The rotational acceleration of the quadrotor is given by the Euler's second equation of motion associated with the gyroscopic torque vector. The gyroscopic torque vector represents a rotation around unwanted axis, perpendicular to the axis of the propeller and the axis around which the wanted rotation is being achieved. The rotational equations of motion are expressed in the body frame, for we need to define any rotation in relation with the center of the quadrotor.

$$\tau = I\dot{w} + w \times (Iw) + \tau_g \tag{3.48}$$

The rotational acceleration is therefore given by:

$$I\dot{w} = -w \times (Iw) + \tau - \tau_g \tag{3.49}$$

where,

- I: 3×3 inertia matrix of the vehicle (considering a symmetric frame).

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.50}$$

- w=$\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$: The angular velocity vector.

- $\tau = (\tau_x, \tau_y, \tau_z)$: The torque applied to the air frame.

- τ_g : The gyroscopic effect torque.

$$\tau_g = w \times G_z \sum_{i=1}^{4} I_r w_i = \begin{bmatrix} \dot{\theta} \\ -\dot{\phi} \\ 0 \end{bmatrix} I_r \sum_{i=1}^{4} (-1)^{i+1} w_i = \begin{bmatrix} I_r\dot{\theta}(w_4 - w_2 + w_3 - w_1) \\ -I_r\dot{\phi}(w_4 - w_2 + w_3 - w_1) \\ 0 \end{bmatrix} \tag{3.51}$$

where,

- I_r: The rotor moment of inertia

Using equations (3.49 and 3.51) the body frame rotational equation of motion is given by:

$$\dot{w} = \begin{bmatrix} \tau_\phi {I_{xx}}^{-1} \\ \tau_\theta {I_{yy}}^{-1} \\ \tau_\psi {I_{zz}}^{-1} \end{bmatrix} - \begin{bmatrix} I_r\dot{\theta}(w_4 - w_2 + w_3 - w_1){I_{xx}}^{-1} \\ -I_r\dot{\phi}(w_4 - w_2 + w_3 - w_1){I_{yy}}^{-1} \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}} w_y w_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}} w_x w_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}} w_x w_y \end{bmatrix} \tag{3.52}$$

3.4.3 Equations of Motion

Putting everything together we obtain:

- Quadrotor velocities in the Inertial Frame:

$$\dot{X_I} = R\dot{X_B}$$

$$\begin{bmatrix} \dot{x_I} \\ \dot{y_I} \\ \dot{z_I} \end{bmatrix} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\phi\sin\theta - \cos\phi\sin\psi & \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \sin\psi\cos\theta & \sin\phi\sin\psi\sin\theta + \cos\phi\cos\psi & -\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{x_B} \\ \dot{y_B} \\ \dot{z_B} \end{bmatrix} \tag{3.53}$$

- Quadrotor Acceleration in the Inertial Frame:

$$\begin{bmatrix} \ddot{x_I} \\ \ddot{y_I} \\ \ddot{z_I} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}([\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi]F_{thrust\{B\}} - k_{dx}\dot{x}_I) \\ \frac{1}{m}([\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi]F_{thrust\{B\}} - k_{dy}\dot{y}_I) \\ \frac{1}{m}([\cos\phi\cos\theta]F_{thrust\{B\}} - k_{dz}\dot{z}_I) + g \end{bmatrix} \tag{3.54}$$

- Euler Angles to Angular rates relation:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.55}$$

- Angular Acceleration equation:

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{I_{xx}}[(I_{yy} - I_{zz})qr - I_r q(w_4 - w_2 + w_3 - w_1) + dk(w_1^2 - w_2^2)] \\ \frac{1}{I_{yy}}[(I_{zz} - I_{xx})pr + I_r p(w_4 - w_2 + w_3 - w_1) + dk(w_3^2 - w_4^2)] \\ \frac{1}{I_{zz}}[(I_{xx} - I_{yy})pq + b(w_4^2 - w_2^2 + w_3^2 - w_1^2)] \end{bmatrix} \tag{3.56}$$

With:

- x_i, y_i, z_i: The coordinates in the inertial frame.
- x_B, y_B, z_B: The coordinates in the body frame.
- $F_{thrust\{B\}} = U_1$: The total thrust of the four motors.
- $dk(w_1^2 - w_2^2) = U_2$: The difference of thrust between the motors along the Y-axis leading to a roll movement and a translation along the Y direction.
- $dk(w_3^2 - w_4^2) = U_3$: The difference of thrust between the motors along the X-axis leading to a pitch movement and a translation along the X-axis.

- $b(w_1^2 - w_2^2 + w_3^2 - w_4^2) = U_4$: The difference of torque between the clockwise and counterclockwise rotors that creates a yaw movement (result of a moment that rotates the quadrotor around the vertical z axis).
- p : roll rate along x-axis in the body frame
- q : pitch rate along y-axis in the body frame
- r : yaw rate along z-axis in the body frame

Chapter 4

Quadrotor System Analysis, Control and Estimation Methods

This part provides a deep understanding of the quadrotor vehicle system by proceeding to a linear and nonlinear analysis. We start by exploring two important concepts that are the existence and uniqueness of solutions. In the second section dedicated to the linear analysis, we first start by defining an equilibrium point, around which we linearize our system. We then analyse the following concepts: Bounded-Input Bounded-Output stability (BIBO stability), internal stability, controllability and observability. In the second part of the same section, we explore in detail and provide a comparison between the PID and LQR. The third and final part of the linear analysis section presents, describes and analyses the concept of states reconstruction via observers design. Two observers techniques are implemented. The Luenberger observer applied for deterministic cases and the Kalman filter that is classified as an optimal observer, used in the presence of Gaussian noise. The third and final section of this chapter addresses the analysis and control of the quadrotor vehicle in the nonlinear point of view. The first section uses sliding mode control method to design a robust controller capable of overcoming some effects not considered by the linear controllers and guarantees stability using Lyapunov stability criteria. The second and last part of this section is concerned with the extension of kalman filtering for the nonlinear case. It presents, analyses and implements an Extended Kalman filter (EKF) that will be used in the target tracking operation defined in chapter six.

4.1 Existence and Uniqueness of solutions:

The problem of existence aims to address the question about whether or not a solution exists and under which condition this solution exists. The uniqueness problem is concerned with whether or not an existing solution is unique.

- Existence

  Let's consider the ODE 3.55 from the equations of motion that relates the derivative of the attitude angles and their angular velocities. This ODE is continuous everywhere except at $\theta = 90$. This means that there exist at least a solution except at this point.

- Uniqueness

  The same ODE 3.55 is continuously differentiable everywhere except at $\theta = 90$. This means that the kinematics are locally Lipschitz at the very least. Solutions exist and are unique as long as θ does not pass through 90.

4.2 Linear Approach

Linear systems theory is a mature area that provides a great variety of tools to analyze and control linear systems. The equations of motion derived in the previous part show a system that is highly nonlinear (sinus and cosinus components). The first stage of this analysis consist on finding an equilibrium point, then around this point analyze concepts such as stability, controllability, observability and finally implement a controller strategy capable of controlling our vehicle to desired locations. In order to achieve this goal the following assumptions are made. The quadrotor movements are near hover state meaning that the changes in angles and positions are small. This assumption helps cancelling the angular and translational rates contributions to the accelerations and the velocities in the XYZ planes. The equations of motion become:

$$\begin{bmatrix} \ddot{x}_I \\ \ddot{y}_I \\ \ddot{z}_I \end{bmatrix} = \begin{bmatrix} \frac{1}{m}[\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi]U_1 \\ \frac{1}{m}[\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi]U_1 \\ \frac{1}{m}[\cos\phi\cos\theta]U_1 - g \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{U_2}{I_{xx}} \\ \frac{U_3}{I_{yy}} \\ \frac{U_4}{I_{zz}} \end{bmatrix} \tag{4.2}$$

The previous assumption can be pushed further to simplify the contribution of the pitch, roll and yaw angles. Considering small oscillations (roll and pitch angles around zero), we have $(\cos(x) = 1, \sin(x) = x)$. The equations of motion become:

$$\begin{bmatrix} \ddot{x}_I \\ \ddot{y}_I \\ \ddot{z}_I \\ \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{U_1}{m}[\cos(\psi)\theta + \sin(\psi)\phi] \\ \frac{U_1}{m}[\sin(\psi)\theta - \cos(\psi)\phi] \\ \frac{U_1}{m} - g \\ \frac{1}{I_{xx}}U_2 \\ \frac{1}{I_{yy}}U_3 \\ \frac{1}{I_{zz}}U_4 \end{bmatrix} \tag{4.3}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.4}$$

4.2.1 Equilibrium point

An equilibrium point is a point where all the rates are zero. Intuitively speaking our vehicle should hover. Let's define two vectors. The first one is the states vector containing the position, angular position, velocity and angular velocities. The second vector is the inputs vector representing the total thrust(U_1), the difference of thrusts (U_2) and (U_3) and the difference of torque(U_4). Note that because of the small angles assumption, the angular velocities are just the derivative of the respective angles (equation (4.4)).

$$X = [x, \dot{x}, y, \dot{y}, z, \dot{z}, \phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}]^T$$

.

$$U = [U_1 \, U_2 \, U_3 \, U_4]$$

From the previous equations, an equilibrium point can be chosen as follows:

$$X_{eq} = [x_{eq} \, 0 \, y_{eq} \, 0 \, z_{eq} \, 0 \, 0 \, 0 \;\; 0 \, 0 \, 0 \, 0] \in \mathbb{R}^{12}$$

with the following input vector:

$$u_{eq} = [mg \, 0 \, 0 \, 0] \in \mathbb{R}^4$$

This point is called a forced equilibrium point and represents the vehicle hovering at a particular position, yet maintained at this position by the total thrust $U_1 = U_{eq}$ that overcomes the gravitational acceleration.

4.2.2 Linearization

Linearization is a process that consists on finding a linear model that approximates a non-linear system presented as follows:

$$\dot{x} = f(x(t), u(t), t)$$

$$y = g(x(t), u(t), t)$$

It represents the system around an equilibrium point. It is done by deriving the following jacobians evaluated at the equilibrium point:

$$A = \frac{\partial f(x,u)}{\partial x}(x_{eq}, u_{eq})$$

$$B = \frac{\partial f(x,u)}{\partial u}(x_{eq}, u_{eq})$$

$$C = \frac{\partial g(x,u)}{\partial u}(x_{eq}, u_{eq})$$

$$D = \frac{\partial g(x,u)}{\partial u}(x_{eq}, u_{eq})$$

The system can thus be represented in the following form:

$$\dot{x} = Ax + Bu \tag{4.5}$$

$$y = Cx + Du \tag{4.6}$$

with:

- $A \in \mathbb{R}^n$: Describes the dynamics of the system. Reflecting how the states evolve with time.
- $B \in \mathbb{R}^{n \times m}$: Describes how the inputs are coupled to the system states.
- $C \in \mathbb{R}^{p \times n}$: Describes how the system states are mapped to the observed outputs. It models the sensors measurement.
- $D \in \mathbb{R}^{p \times n}$: Describes the feedforward matrix. It allows the system inputs to directly affect the system outputs.
- $x \in \mathbb{R}^n$: The states vector.
- $u \in \mathbb{R}^p$: Input or control vector.

We begin by representing the system is state-space form. To achieve this, let's define the following state-vector: $X = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}\ ]^T$. We have:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \\ \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix} \rightarrow \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \\ \dot{x}_9 \\ \dot{x}_{10} \\ \dot{x}_{11} \\ \dot{x}_{12} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{1}{m}([\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi]U_1 - k_{dx}\dot{x}_I) \\ x_4 \\ \frac{1}{m}([\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi]U_1 - k_{dy}\dot{y}_I) \\ x_6 \\ \frac{U_1}{m}(\cos\phi\cos\theta) - g \\ x_8 \\ \frac{1}{I_x}[(I_y - I_z)qr + I_r q(w_1 - w_2 + w_3 - w_4) + U_2] \\ x_{10} \\ \frac{1}{I_y}[(I_z - I_x)pr + I_r p(w_1 - w_2 + w_3 - w_4) + U_3] \\ x_{12} \\ \frac{1}{I_z}[(I_x - I_y)pq + U_4] \end{bmatrix} \quad (4.7)$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \\ \dot{x}_9 \\ \dot{x}_{10} \\ \dot{x}_{11} \\ \dot{x}_{12} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{1}{m}([\cos(x_7)\cos(x_{11})\sin(x_9) + \sin(x_7)\sin(x_{11})]U_1 - k_{dx}x_2) \\ x_4 \\ \frac{1}{m}([\cos(x_7)\sin(x_{11})\sin(x_9) - \cos(x_{11})\sin(x_7)]U_1 - k_{dy}x_4) \\ x_6 \\ \frac{U_1}{m}(\cos(x_7)\cos(x_9)) - g \\ x_8 \\ \frac{1}{I_{xx}}[(I_{yy} - I_{zz})x_{10}x_{12} + I_r x_{10}(w_1 - w_2 + w_3 - w_4) + U_2] \\ x_{10} \\ \frac{1}{I_{yy}}[(I_{zz} - I_{xx})x_8 x_{12} + I_r x_8(w_1 - w_2 + w_3 - w_4) + U_3] \\ x_{12} \\ \frac{1}{I_{zz}}[(I_{xx} - I_{yy})x_8 x_{10} + U_4] \end{bmatrix} \tag{4.8}$$

Considering the previous assumptions and the jacobians defined above, we obtain the following matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.9}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \tag{4.10}$$

Assuming full access to our outputs, in addition to no feedforward terms we get:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.12}$$

4.2.3 Stability Analysis

Stability is one of the most important properties of a system. It represents the ability of the system to operate within acceptable bounds that guarantee security, efficiency and reliability. The response of a system can be decomposed into a zero-state response (forced response) and a zero-input response (natural response). We study stabilities of these later separately.

4.2.3.1 BIBO stability

BIBO stability is related to the zero states response and represents the ability of the system to produce bounded outputs for any applied bounded inputs. To check this property we first computed the transfer functions of the system and checked the location of the poles on the complex s-plane. Notice that we have four inputs and 12 maximum outputs (according to sensors). We chose an identity matrix of dimension (12×12) as shown by the 4.11, to analyze all the poles of the 12 different transfer functions relating the four inputs to the 12-outputs.

- Total thrust to z-position:

  input 1 to output 5:G(s)=$\frac{0.5208}{s^2}$

- Total thrust to velocity about z-axis

  input 1 to output 6:G(s)=$\frac{0.5208}{s}$

- Roll control input to y-position

  input 2 to output 3:G(s)=$\frac{-196.4}{s^4}$

- Roll control input to velocity about y-axis

  input 2 to output 4:G(s)=$\frac{-196.4}{s^3}$

- Roll control input to roll output angle

  input 2 to output 7:G(s)=$\frac{20.04}{s^2}$

- Roll control input to rotational velocity about x-axis

  input 2 to output 8:G(s)=$\frac{20.04}{s}$

- Pitch control input to x-position

  input 3 to output 1:G(s)=$\frac{196.4}{s^4}$

- Pitch control input to velocity about x-axis

  input 3 to output 2:G(s)=$\frac{196.4}{s^3}$

- Pitch control input to Pitch output angle

  input 3 to output 9:G(s)=$\frac{20.04}{s^2}$

- Pitch control input to rotational velocity about y-axis

  input 3 to output 10:G(s)=$\frac{20.04}{s}$

- Yaw control input to Yaw output angle

  input 4 to output 11:G(s)=$\frac{4.157}{s^2}$

- Yaw control input to rotational velocity about z-axis

  input 4 to output 12:G(s)=$\frac{4.157}{s}$

From the transfer functions we see that we have 12 repeated poles equals to zero meaning that the open-loop system is not BIBO stable since the condition for BIBO stability is that all the poles lie on the negative part of the complex s-plane.

4.2.3.2 Internal Stability

This stability is related to the zero-input response and is about how the internal states of the system behave. To check this property we compute the eigenvalues of the matrix A that represent the topology of the system. Using Matlab, we obtained a repeated eigenvalue $\lambda = 0$ of order 12. At this point, the system could be marginally stable, yet the zero eigenvalue is not a simple root of the minimal polynomial that is given by: min$poly(\lambda) = \lambda^4$. In addition to the previous result the system also generates Jordan blocks of order greater than 1 for the same eigenvalue definitely showing that the eigenvalue zero is not simple root of the minimal polynomial. All these later facts make the open-loop of the system definitively unstable.

Figure 4.1 represents the step response of some states of our open-loop system (Z-position,ϕ, θ, ψ). All the states are diverging showing beyond doubt that the open-loop system is unstable.

$$J = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

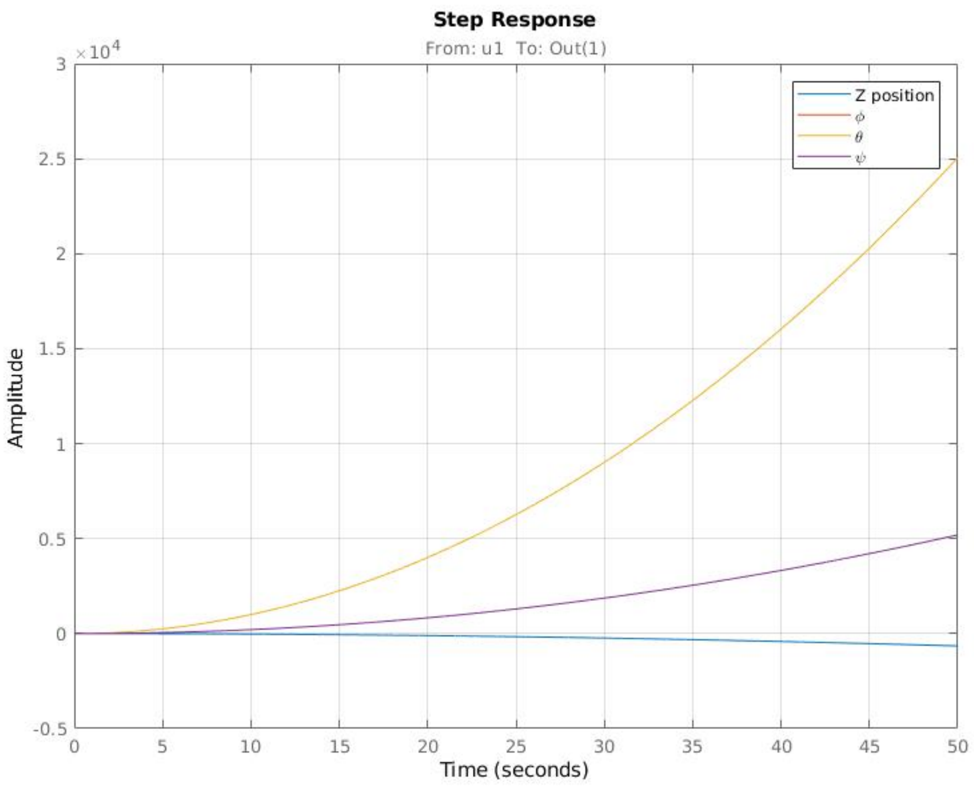


Figure 4.1: Diverging Open loop States.

4.2.4 Controllabiltiy and Observability

4.2.4.1 Controllability

Controllability is an important property of control system, it plays a crucial role since it represents the ability to use an input to drive the system from any initial state to any final one in a finite amount of time. For a system to be fully controllable its controllability matrix $C = [B \;\; AB \;\; A^2B...A^{n-1}B]$ must be full-rank. In our case, using the Matlab commands $ctrb(A, B)$ and $rank(ctrb(A, B))$ we found a controllability matrix of dimension 12×48 with rank equals to twelve. The matrix being fully controllable it is possible using feedback techniques to affect the system behavior by using various control methods.

4.2.4.2 Observability

Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. For a system to be fully observable its observability matrix $O = [C\ CA\ CA^2, ...CA^{n-1}]^T$ must be full-rank. In our case, using the Matlab commands $obsv(A, C)$ and $rank(obsv)$ we found an Observability matrix of dimension 72×12 with column rank = 12.

4.2.5 Linear Control Techniques

The main goal of any electrical or electronic control system is to measure, monitor, and control a process. We have seen from the previous analysis that our open-loop system is not stable, yet thanks to its controllability property we can affect its states to achieve stability and control our system to desired goal. In this part we use feedback techniques (Figure 4.2) consisting on monitoring its outputs and "feeding" some/or all of them back to compare the actual output to the desired one so as to reduce the error and bring the output of the system back to the desired response.

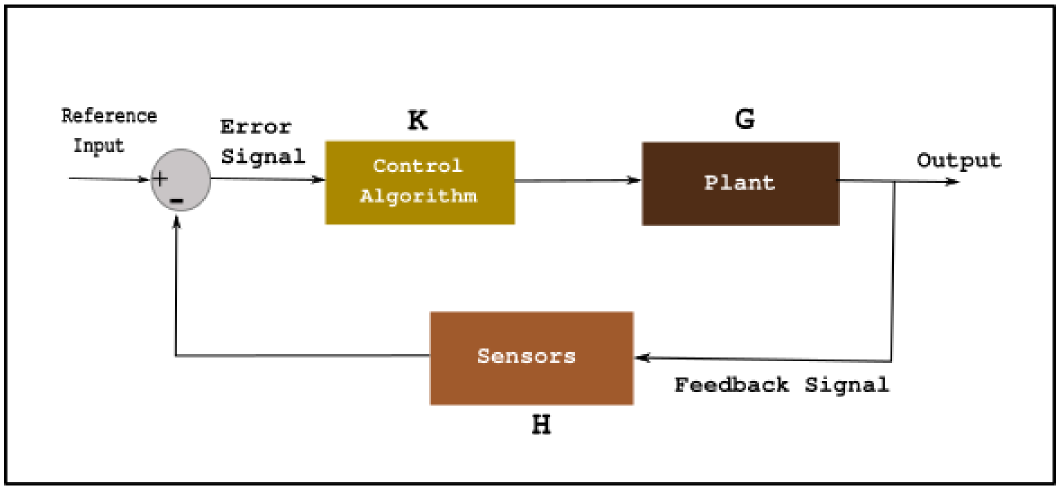


Figure 4.2: Feedback Control.

4.2.5.1 Proportional Integral And Derivative

The basic idea behind a PID controller is to read a sensor, then compute the desired actuator output by calculating the proportional, integral, and derivative responses. The sum of those three components gives us the final compensated output (Figure 4.3). In our case, the controller calculates the difference between the actual and desired state and produces an error value. The actual states that are the measurements from the quadrotor's sensors are fed back to compute this error signal, closing the loop from output to input. The output of any PID control system is a control value that will drive the system closer to the desired state.

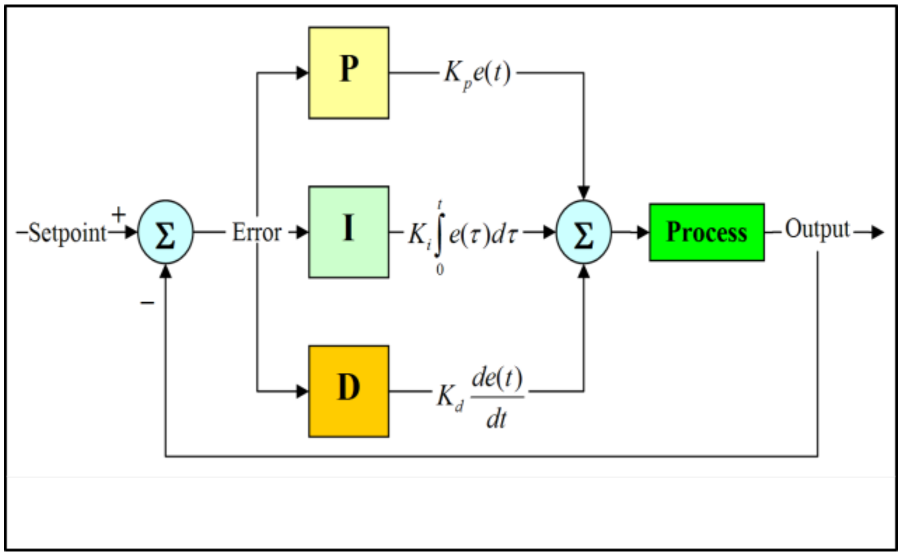


Figure 4.3: Proportional and Integral Derivative.

This control technique consists of three components:

- The proportional component: Any value greater than zero that multiply the error signal.
- The integral term: It is used to eliminate the steady-state error and is a multiple of the sum of errors accumulated over time.
- The derivative term: This component is used to target the rate of change of the error to reduce or avoid the overshoot of the system beyond the desired state by dampening the response of the system.

The Controller can be summarized as follows:

- t: Time
- K_P: Proportional Control Gain
- K_I: Integral Control Gain
- K_D: Derivative Control Gain
- $x_D(t)$: Desired value of state x at time t
- P: Proportional Control term

  $P = K_p * e(t)$
- I: Integral Control term

  $I = K_I \int_0^t e(t-1)dt$
- D: Derivative Control term

  $D = K_D * \dot{e}(t)$
- $x(t)$: Actual value of state x at time t
- $e(t)$: Value of the error between desired and actual states at time t
- $\dot{e}(t)$: Derivative of the error at time t
- $u(t)$: Control Input

  $u(t) = K_p * e(t) + K_I \int_0^t e(t-1)d(t) + K_D * \dot{e}(t)$

4.2.5.2 Quadrotor Control Using PID

To control our vehicle, we will use a nested control loop that consist of four different loops controlling positions, both attitude and altitude, angular velocities and the four motors' speed (Figure 4.4).

- Position Control Loop:

This controller is implemented to assure that the desired position is achieved. In fact, small changes in roll or pitch or even yaw angles can create a huge change in position. The inputs of this controller are the position data obtained from a global positioning system such as GPS for outdoor navigation or a motion capture system such as VICON for indoor navigation. This controller output is the desired roll and pitch angles to achieve desired X,Y positions.

$$\phi_d(t) = K_p * (X_d^G - X^G) + K_I \int_0^t (X_d^G - X^G)dt + K_D * (\dot{X_d^G} - \dot{X^G})$$

$$\theta_d(t) = K_p * (Y_d^G - Y^G) + K_I \int_0^t (Y_d^G - Y^G)dt + K_D * (\dot{Y_d^G} - \dot{Y^G})$$

- Attitude and Altitude Control:

The inputs of this control loop are the desired roll and pitch angles obtained from the position controller, in addition to the desired yaw angle. Altitude is related to the total thrust of the vehicle that affects the aggressiveness of the vehicle in the x and y directions. Because yaw movement influences the vertical acceleration of the vehicle along the z-axis and because the vertical acceleration component is also present in the X and Y direction, we need to be able to monitor the desired value of the yaw angle to achieve our final goal. Clearly, this controller goal is to compute the desired angular velocities and the total thrust needed to achieve these desired roll and pitch values.

$$U_1 = \frac{1}{cos(\phi)cos(\theta)}[K_p * (Z_d^G - Z^G) + K_I \int_0^t (Z_d^G - Z^G)dt + K_D * (\dot{Z_d^G} - \dot{Z^G})]$$

$$p_d(t) = K_p * (\phi_d - \phi) + K_I \int_0^t (\phi_d - \phi)dt + K_D * (\frac{e_\phi - e_\phi(t-1)}{dt})$$

$$q_d(t) = K_p * (\theta_d - \theta) + K_I \int_0^t (\theta_d - \theta)dt + K_D * (\frac{e_\theta - e_\theta(t-1)}{dt})$$

$$r_d(t) = K_p * (\psi_d - \psi) + K_I \int_0^t (\psi_d - \psi)dt + K_D * (\frac{e_\psi - e_\psi(t-1)}{dt})$$

- Angular Velocity Control:

This controller inputs are the desired angular velocities. It computes the error between these desired rates and the ones measured by the gyroscope that is used to obtain the other three moments control inputs respectively along the x, y and z axis.

$$U_2(t) = K_p * (p_d - p) + K_I \int_0^t (p_d - p)dt + K_D * (\frac{e_p - e_p(t-1)}{dt})$$

$$U_3(t) = K_p * (q_d - q) + K_I \int_0^t (q_d - q)dt + K_D * (\frac{e_q - e_q(t-1)}{dt})$$

$$U_4(t) = K_p * (r_d - r) + K_I \int_0^t (r_d - r)dt + K_D * (\frac{e_r - e_r(t-1)}{dt})$$

- Motor Control:

  This Controller is used to obtain the speed for each motors to achieve our desired goal. These speeds are sent to the motor's electronic speed controller (ESC) that is a PWM controller in order to make the motors spin at the required speed to produce desired thrust and control moments. The Desired motors speeds are computed using the inverse of the control matrix. This process is defined as follows:

$$\begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{bmatrix} = \begin{bmatrix} K_T & K_T & K_T & K_T \\ 0 & -lK_T & 0 & lK_T \\ lK_T & 0 & -lK_T & 0 \\ K_d & -K_d & K_d & -K_d \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{4.13}$$

The following diagram illustrates the idea behind the nested control loop method.

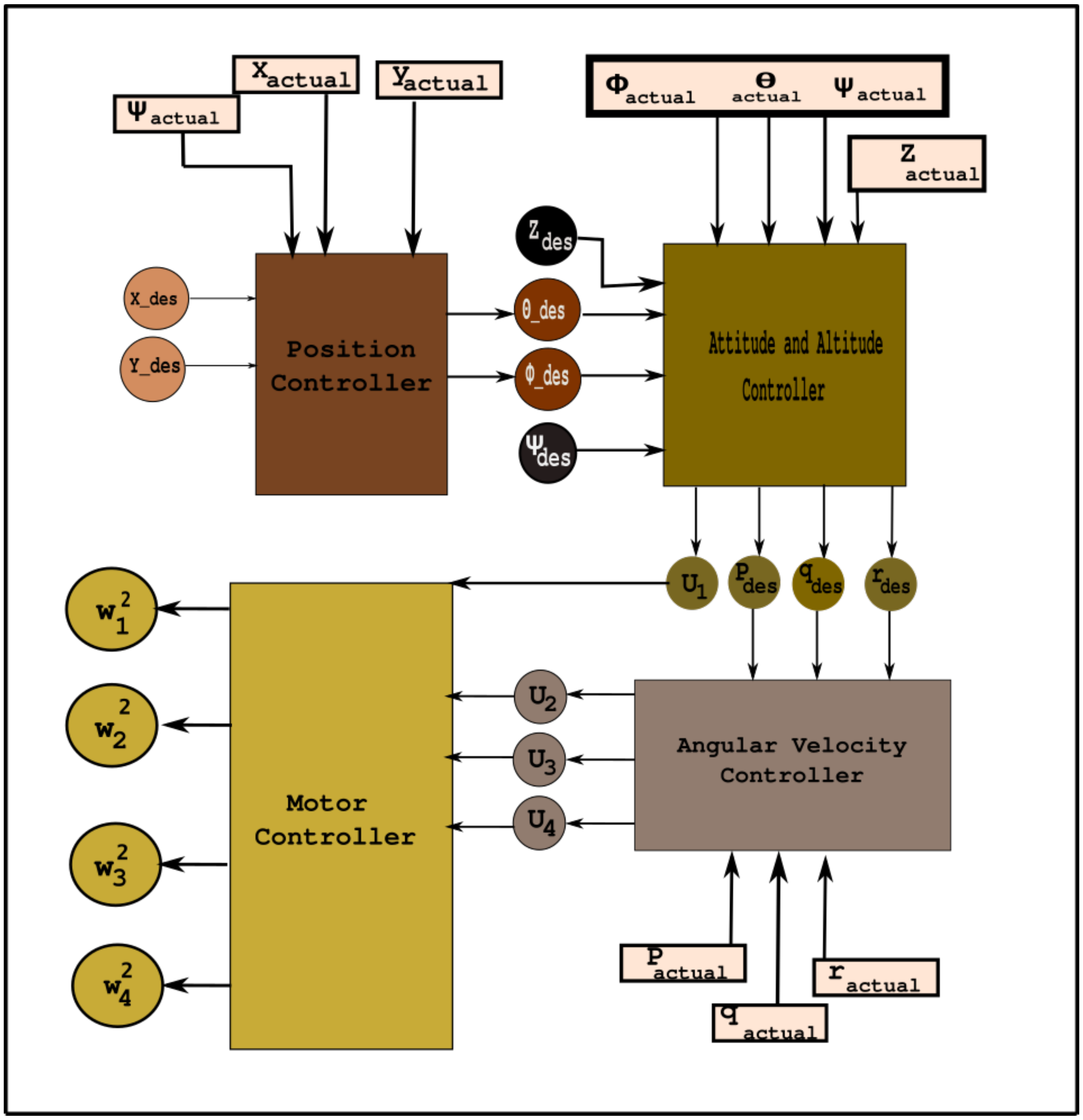
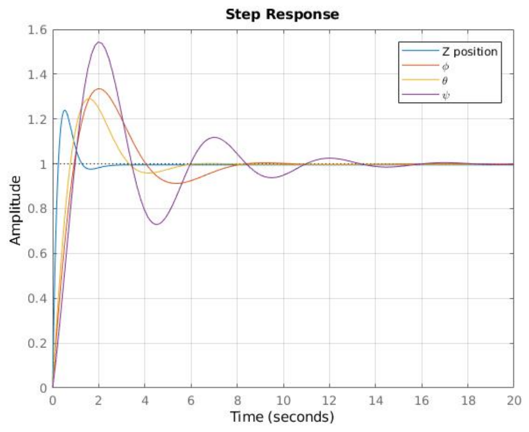
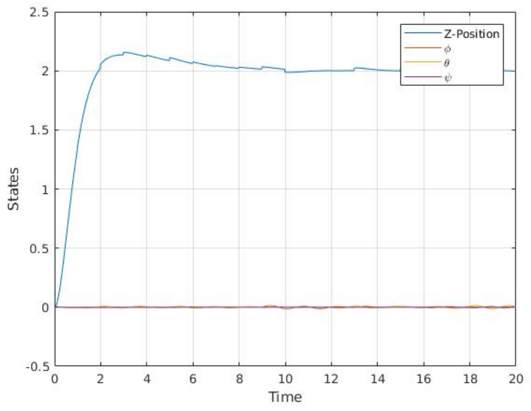


Figure 4.4: Nested Control Loop Diagram.

4.2.5.3 PID Control Results



(a) Step Response



(b) Attitude and Attitude Control

Figure 4.5: Step response and Altitude, Attitude control.

Time-Domain Characteristics				
	Roll	Pitch	Yaw	Z Position
RiseTime:	0.7274	0.5968	0.7568	0.1914
SettlingTime:	7.4354	5.1610	12.5030	1.8885
% Overshoot:	33.4684%	29.0291 %	54.3458%	23.9087%
Peak:	1.3347	1.2903	1.5435	1.2391
PeakTime:	2.0714	1.5571	1.9850	0.5318

Table 4.1: Closed loop Characteristics (for Step Response).

PID controllers are the most used method to control quadrotors, for they are very intuitive and easy to implement in real hardware. One of the disadvantage of this control method is the tuning process (choosing the right gain values) that can be tedious. In addition to be previous downside, this method is not adaptive and not very robust. In fact, changing the battery or even the propellers can lead to changes in the system response. Table 4.1 shows the step responses (Figure 4.5a) characteristics. This response is definitely stable and shows interesting information such as a low rise-time. Depending on the tuning, these responses can be modified for more suitable results. The Figure 4.5b shows a basic altitude and attitude control which objective is a simple take-off and hover at 2 meters (Z=0; $\phi = 0$; $\theta = 0$; $\psi = 0$). The response shows a small overshoot regarding the z-position and some small oscillations around zero for the angles.

4.2.5.4 Linear Quadratic Regulator

The preceding method can be categorized in the classical control area. It heavily relies on trial and error process. To obtain good values for gains P, I and D, a tuning process is used in an iterative way to get parameters judged acceptable according to some defined goals such as peak overshoot, settling time, rise time...the tuning process can be long and tedious. The concept of LQR is an optimal control method that is concerned in operating a dynamical system at minimum cost. This algorithm considerably reduces the amount of work to optimize the controller.

Finite horizon, LQR equations:

$$\dot{x} = Ax(t) + Bu(t) \tag{4.14}$$

$$J = \frac{1}{2}\int_0^T (x^T Q x + u^T R u)dt + \frac{1}{2}x^T(T)P_1 x(T) \tag{4.15}$$

with:

- J: Quadratic cost function
- $Q \geq 0$:Positive semi-definite and symmetric

- $R > 0$:Positive definite and symmetric
- $P_1 \geq 0$: Positive semi-definite and symmetric

The goal is to find the optimal control law that minimizes the cost function. To achieve that few steps are required.

- solve equations (4.14 and 4.15) using the maximum principle that states that:

Theorem 1: Maximum Principle

If (x_{opt}, u_{opt}) is optimal, then there exists $\lambda_{opt}(t)$

$t \in \Re$ and $\nu_{opt} \in \Re^q$ such that:

$$\dot{x_i} = (\frac{\partial H}{\partial \lambda_i}) \quad \text{and} \quad -\dot{\lambda}_i = (\frac{\partial H}{\partial x_i})$$

Given the boundary conditions:

n initial conditions:

$$x(0) \in R^n$$

q terminal constraints:

$$\Psi(x(T)) = 0 :$$

n-q final values of the lagrange multiplier:

$$\lambda(T) = \frac{\partial V}{\partial x}(x(T)) + \nu^T \frac{\partial \Psi}{\partial x}$$

with ν as a free variable.

Such that:

The Hamiltonian H respects the following property :

$$H(x_{opt}(t), u_{opt}(t), \lambda_{opt}(t)) \leq H(x_{opt}(t), u(t), \lambda_{opt}(t)) \; \forall u \in \Omega$$

Note:

The maximum principle allows the input to be constrained to lie in a set Ω, allowing the possibility of bounded inputs. If $\Omega = R^m$(unconstrained input) and H is differentiable, then a necessary condition for the optimal input is:

$$\frac{\partial H}{\partial u} = 0$$

Notice that even though we are minimizing the cost, the theorem is still usually called the maximum principle. Applying this theorem to our linear quadratic problem gives:

$$H = x^T Q x + u^T R u + \lambda^T (Ax + Bu) \tag{4.16}$$

$$\dot{x} = Ax + Bu = (\frac{\partial H}{\partial \lambda})^T \qquad x(0) = x_0 \tag{4.17}$$

$$-\dot{\lambda} = Qx + A^T \lambda = (\frac{\partial H}{\partial x})^T \qquad \lambda(T) = P_1 x(T) \tag{4.18}$$

$$0 = Ru + \lambda^T B = (\frac{\partial H}{\partial u})^T \tag{4.19}$$

The optimal control input $u(t) = -R^{-1} B^T \lambda$ is obtained by solving equation (4.18) with $\lambda = P(t)x(t)$.

- Solve the Riccati Ordinary Differential :

The P(t) matrix is obtained by solving the well known Riccati Ordinary differential equation that is:

$$-\dot{P} = PA + A^T P - PBR^{-1} B^T P + Q \qquad P(T) = P_1 \tag{4.20}$$

- Optimal control Input:

The optimal control input is given by:

$$u(t) = -R^{-1} B^T P(t) x(t)$$

(4.21)

4.2.5.5 Quadrotor Control Using LQR

The linearization procedure permitted to obtain a linear system represented by the A,B,C matrices. In this section, we design a feedback controller based on the Linear Quadratic Regulator(LQR) in order to stabilize the system.
The linear system ODE is represented by the following linear equations:

$$\dot{x} = Ax(t) + Bu(t)$$

considering the input:

$$u(t) = -Kx(t)$$

The closed-loop equation is given by:

$$\dot{x} = (A - BK)x(t)$$

The remaining task is to find a correct gain K such that the new closed-loop system behaves according to some design criteria. In our case we want first to stabilize the system and then to drive the steady state error to zero in addition to a minimum settling time. The gain K that is simply given by the expression $K = -R^{-1}B^{T}P(t)$ derived in the previous section was obtained using the Matlab 'lqr' command for our specific linearized system. The next section present the linear control method results obtained from a real vehicle.

4.2.6 LQR control methods result

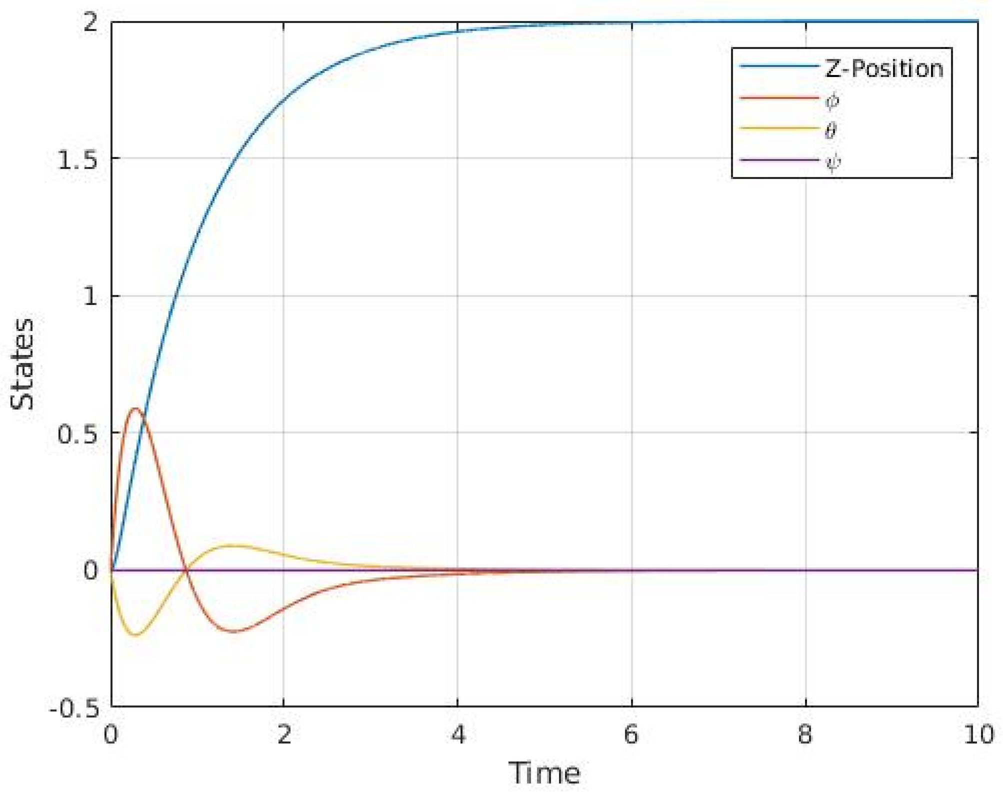


Figure 4.6: Altitude and Attitude LQR Control.

Figure 4.6 presents the response of the system with the LQR Controller. The Z-position never overshoots and all the angles converge to zero with few oscillations. Similarly to the PID controller, it is possible to go through a tuning process to reach some design criteria. This is done by playing with the Q and R matrices representing respectively some constraints on the states and the inputs.

4.2.7 Linear control methods result

The analysis and methods explored in this chapter gave us a deep insight of the natural behavior of the vehicle. Furthermore, PID and LQR methods provided us a means to control and modify the behavior of our system. The analysis and control algorithms have been implement in Matlab. Both methods show good results, with an advantage for the LQR controller that require less tuning processes. The problem of these methods is that they assume in most of the cases that all the states are measurable. When this previous assumption is not fulfilled an observer must be designed to reconstruct the states form the outputs information.

4.2.8 Observability and Observers Design

4.2.8.1 Observability

Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. Observability is closely related to sensors and observers can be seen as virtual models used when measurement is not available. In order to design an observer, our system must be observable. This property of the system depends on the matrices A and the output matrix C.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9.8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.22}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.23)$$

For a system to be fully Observable its Observability matrix

$$O = [C, CA, CA^2, ...CA^{n-1}]^T$$

must be full column rank. This property is easily verifiable using Matlab. The command $rank(obsv(A, C))$ returned an observability matrix of dimension 72×12 with column rank being 12. The matrix being fully observable, we can reconstruct our states using observers techniques

4.2.8.2 Linear Observer Design

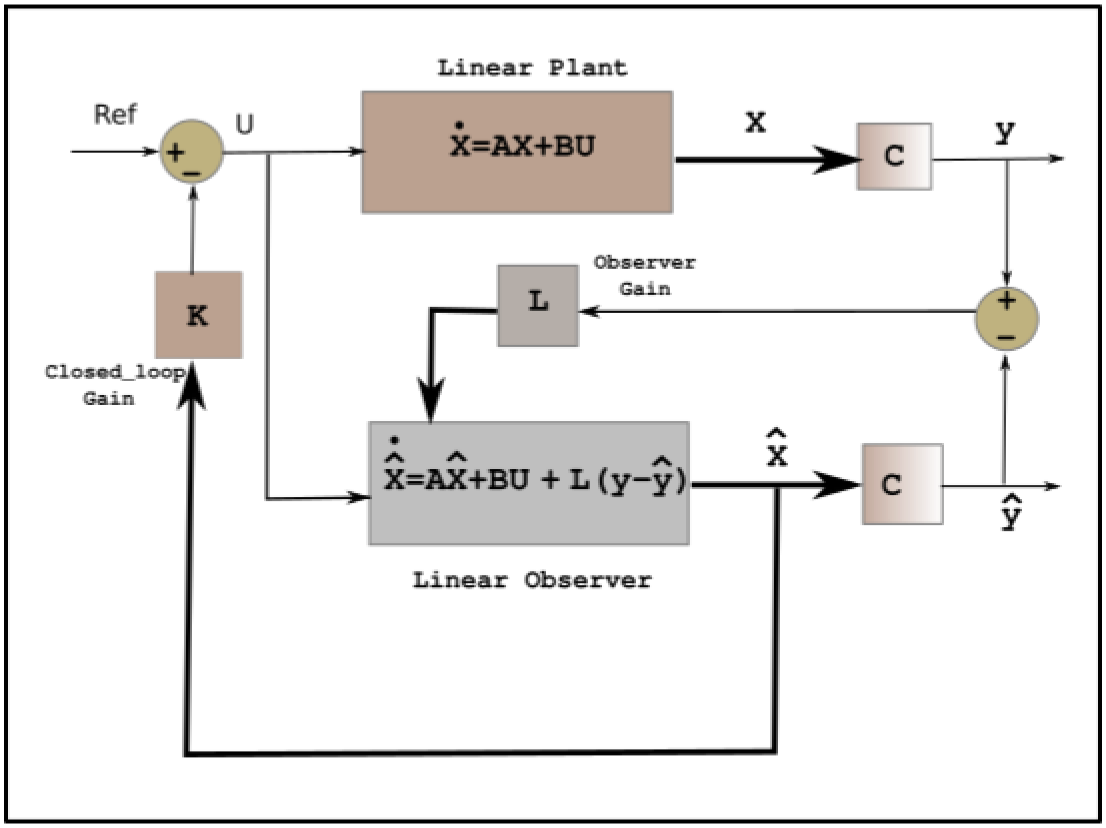


Figure 4.7: Linear States Observer Diagram.

In the previous section we have been able to control our system using two different feedback control methods. It was assumed that all state variables were available. In the case that this assumption fails an observer can be used to reconstruct the states. It exploits the fact that the outputs are just a linear combination of the state variables ($y(t) = Cx(t)$). Notice that in general, the dimension of the output signal is much smaller than the dimension of the state variable.

$$dim\{y(t)\} = l = c < n = dim\{x(t)\}$$

Considering a linear system of order n, which output variables are available all the time, we can design an artificial dynamic system of the same order that will estimate the original system state-space variables at all times.

- Linear system dynamics

$$\dot{x} = Ax + Bu, \quad x(t_0) = x_0 = \text{Unknown} \tag{4.24}$$

$$y(t) = Cx(t) \tag{4.25}$$

- Observer dynamics

$$\dot{\hat{x}} = A\hat{x} + Bu, \quad \hat{x}(t_0) = \hat{x}_0 = \hat{x}_0 \tag{4.26}$$

$$\hat{y}(t) = C\hat{x}(t) \tag{4.27}$$

- Error signal between the original system and the observer outputs

$$y(t) - \hat{y}(t) = Cx((t) - C\hat{x}(t) = Ce(t) \tag{4.28}$$

This error will be used as a feedback signal to the observer with the objective to asymptotically drive this later to zero. This is done by choosing the appropriate observer gain represented by the matrix L in the following equations.

- Observer including measurement error

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y(t) - \hat{y}(t)) = A\hat{x} + Bu + LCe(t) \tag{4.29}$$

From equations (4.5 and 4.29) we can derive the dynamics of the observation error.

- Observer error Dynamics:

$$\dot{e} = (A - LC)e(t) \quad e(t_0) \quad \text{unknown} \tag{4.30}$$

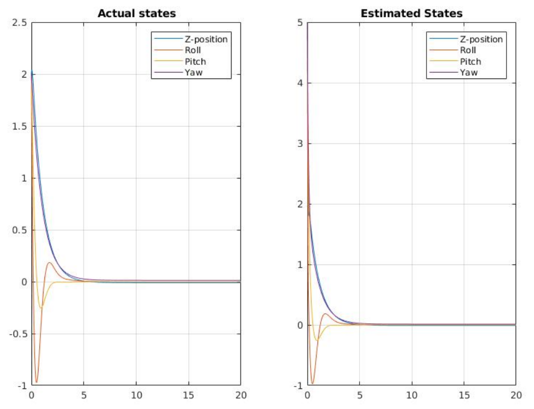
The observer gain L must be chosen such that the matrix $A - LC$ of equation (4.30) has all its eigenvalues with negative part. The estimation speed depends on these eigenvalues. In fact, the more far they are in the left half-plane the faster the observer will converge to the actual states, yet very fast observers generate noise. A good choice for the observer eigenvalues is that they be 10 times faster than the one of the system. It can be achieved by choosing a gain L that set the real-part of the smallest eigenvalue of our observer 10 times bigger than the real part of the biggest eigenvalue of our closed loop system.

- Convergence rate setting:

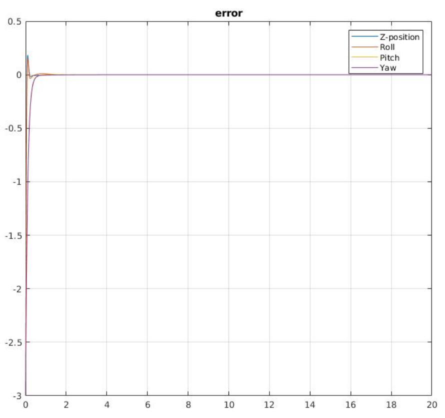
$$|\mathbb{R}_{eal}(\lambda_{min}(A - LC))|_{observer} > 10 * |\mathbb{R}_{eal}(\lambda_{max}(A - BK))|_{system} \tag{4.31}$$

with K the gain matrix of the closed loop system such that $U = -Kx(t)$ and the closed loop defined as $\dot{x} = (A - BK)x(t)$ is asymptotically stable.

4.2.8.3 Observer Design Results



(a) Actual States vs Estimated states



(b) Errors

Figure 4.8: Actual States, Estimated states and Errors.

Figure 4.8 presents the simulation results with a step input where the altitude, roll, pitch and yaw angles are estimated. Figure 4.8a shows the actual and the estimated states meanwhile Figure 4.8b illustrates their relative errors. Notice that even if the actual and estimated states start with different initial conditions, the observer states successfully converges to the actual states leading to an asymptotic convergence of the error to zero. Table 4.2 illustrates the eigenvalues choices.

Eigenvalues		
	Actual states	Estimated States
Z-position:	$1.0e+03$*(-0.00225+0.0022i)	$1.0e+04$*(-0.00225+0.0022i)
Roll:	$1.0e+03$*(-0.0022+0.0022i)	$1.0e+04$*(-0.0022+0.0022i)
Pitch:	$1.0e+03$*(-0.0010+0.0000i)	$1.0e+04$*(-0.0010+0.0000i)
Yaw:	$1.0e+03$*(-0.0010+0.0000i)	$1.0e+04$*(-0.0010+0.0000i)

Table 4.2: Eigenvalues choices.

4.2.8.4 Kalman Filtering

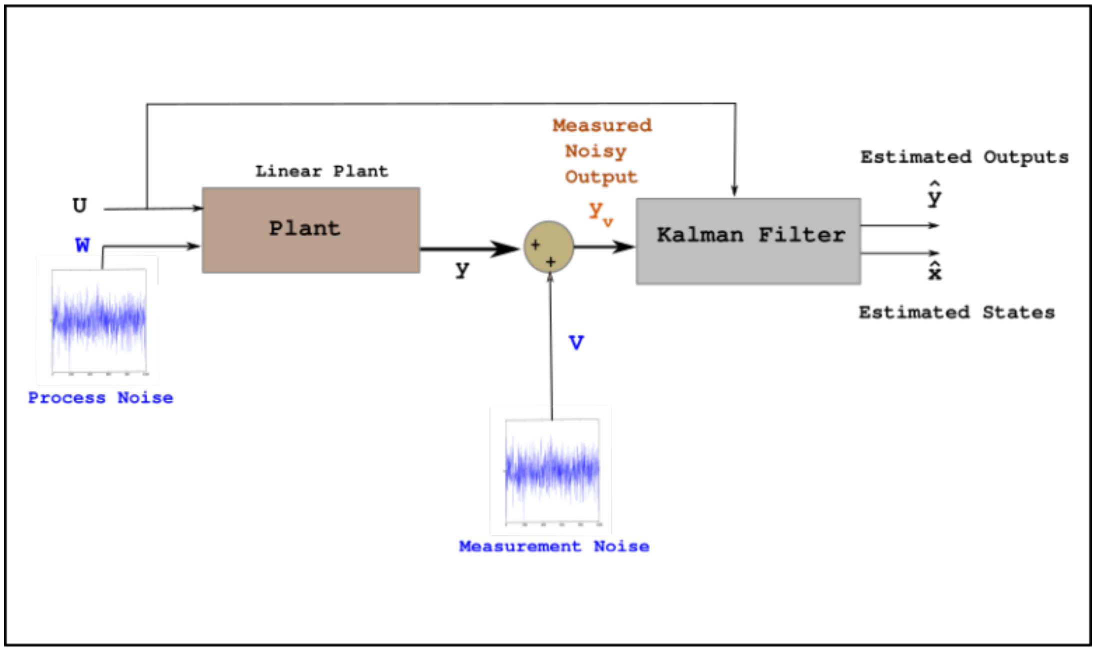


Figure 4.9: State Estimate with Kalman Filter.

The Kalman Filter also called optimal states observer is a states estimator designed for stochastic processes contrary to the previous observer that works for deterministic cases. It combines measurement and prediction to find the optimal estimate of the states measured, in the presence of process and measurement noises. It is ideal for systems which are continuously changing and are subject to uncertainties in both the process and the measurements. It can be broken down in two phases which are: the prediction and update phase.

Let's consider the following linear system:

$$x_k = Ax_{k-1} + Bu_k + w_k \tag{4.32}$$

$$y_k = Cx_k + v_k \tag{4.33}$$

where:
$w_k \sim \mathcal{N}(0, Q)$: Process noise modeled as a gaussian distribution with zero mean and covariance Q
$v_k \sim \mathcal{N}(0, R)$: measurement noise modeled as a gaussian distribution with zero mean and covariance R.

Equations (4.32 and 4.33) represent a discrete time linear system subject to uncertainties. The process noise w_k represents uncertainties in the process (could be the wind in the case of our quadrotor) and v_k, the measurement noise represents the noise that could be added to sensors readings.

- The Prediction phase:

$$\hat{x}^- = A_k \hat{x}_{k-1} + B_k u_k \tag{4.34}$$

$$P_k^- = AP_{k-1}A_k^T + Q_k \tag{4.35}$$

with:
$\hat{x}^-$: The a priori estimate representing the estimate before the current measurement is taken.

$\hat{x}_{k-1}$: The previous estimate

P_k^-: The a priori error covariance that captures uncertainties in the process.

P_{k-1}: The previous error covariance matrix.

Equation (4.34) is used to predict the new best estimate from the previous best estimate in addition to a correction for known external influences, meanwhile equation (4.35) predicts a new uncertainty from the old one with some additional uncertainty from the environment.

- Update Phase:

  Let's consider the following difference:

$$v(k+1) = z(k+1) - C\hat{x}(k+1) \tag{4.36}$$

with:

$z(k+1)$: Data measured by the sensors

$C\hat{x}(k+1)$: Data that the sensors are predicted to measure.

Some of the difference will be due to the noise in the sensor (measurement noise), yet the remaining discrepancy indicates the error between the predicted states and the sensors observations.

The update phase can be summarized by the following equations:

$$\hat{x}'_k = \hat{x}_k + K(z_k - C_k\hat{x}_k) \tag{4.37}$$

$$\hat{P}'_k = P_k - KC_kP_k \tag{4.38}$$

$$K = P_kC_k^T(C_kP_kC_k^T + R_k)^{-1} \tag{4.39}$$

with:

$\hat{x}'_k$: The actual new best estimate

$\hat{x}(k)$: The actual estimated state

$\hat{P'}_k$: The actual new best estimated error covariance

$\hat{P}_k$: The actual estimated error covariance

K: The Kalman Gain matrix.

The update phase goal is to refine the estimate with upcoming measurements. It takes advantage of the estimated states and the noisy sensors readings mean and variance to outcome a gain that is used to refine the estimate.

Equations (4.37, 4.38 and 4.39) represent a mapping of the innovation into a correction for the predicted state. It optimally adjust the estimate based on what the sensor has observed.

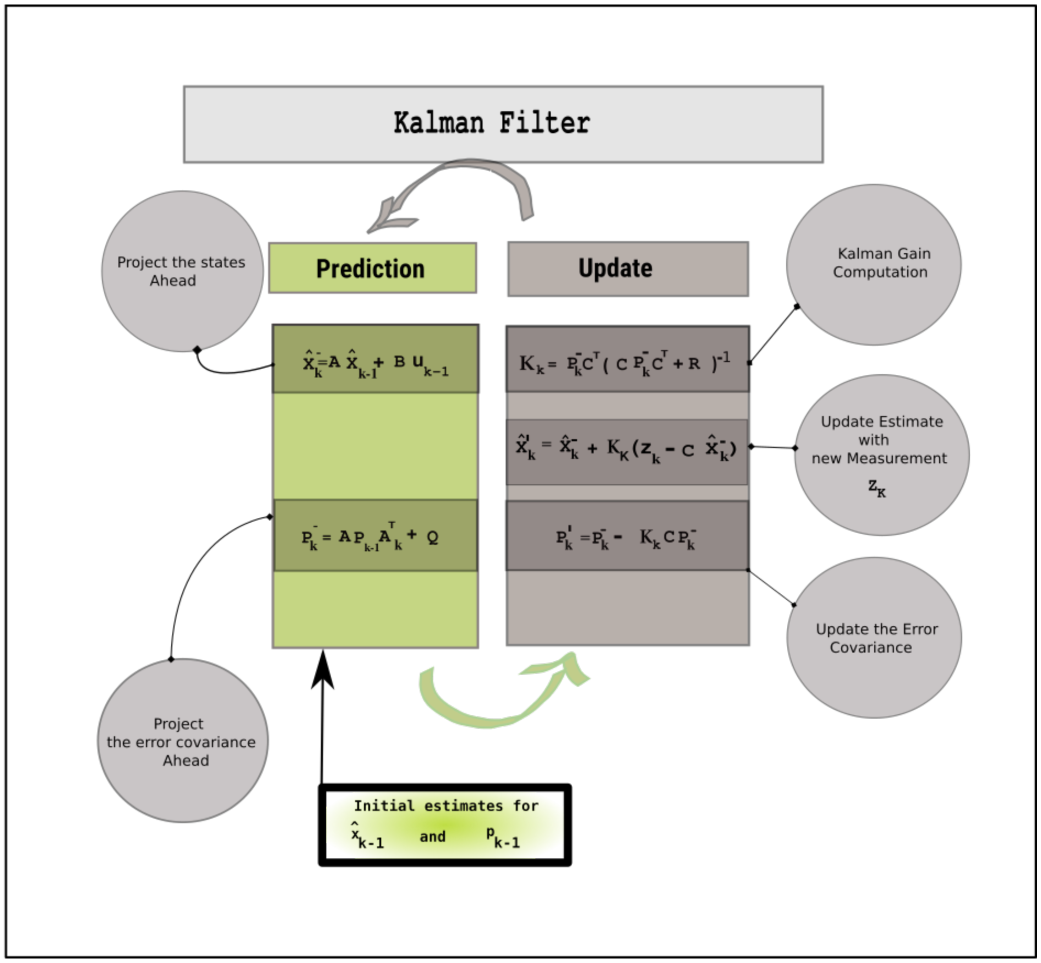

Kalman Filter
Project the states Ahead
Prediction
Update
Kalman Gain Computation
$\hat{x}^-_k = A\hat{x}_{k-1} + B u_{k-1}$
$K_k = P^-_k C^T (C P^-_k C^T + R)^{-1}$
Update Estimate with new Measurement z_K
$\hat{x}'_k = \hat{x}^-_k + K_K(z_k - C\hat{x}^-_k)$
$P^-_k = A P_{k-1} A^T_k + Q$
$P'_k = P^-_k - K_k C P^-_k$
Update the Error Covariance
Project the error covariance Ahead
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Figure 4.10: Kalman filter Diagram.

4.2.8.5 Kalman Filter Results

- Simulink Implementation

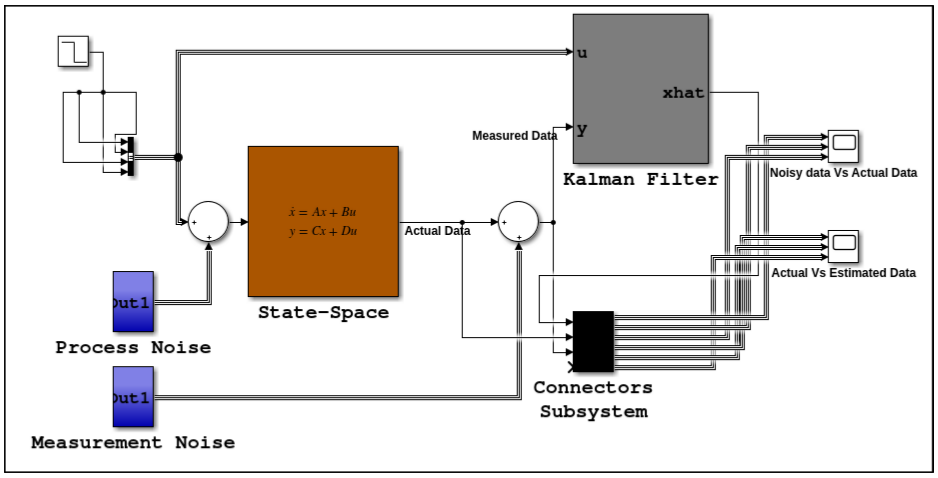
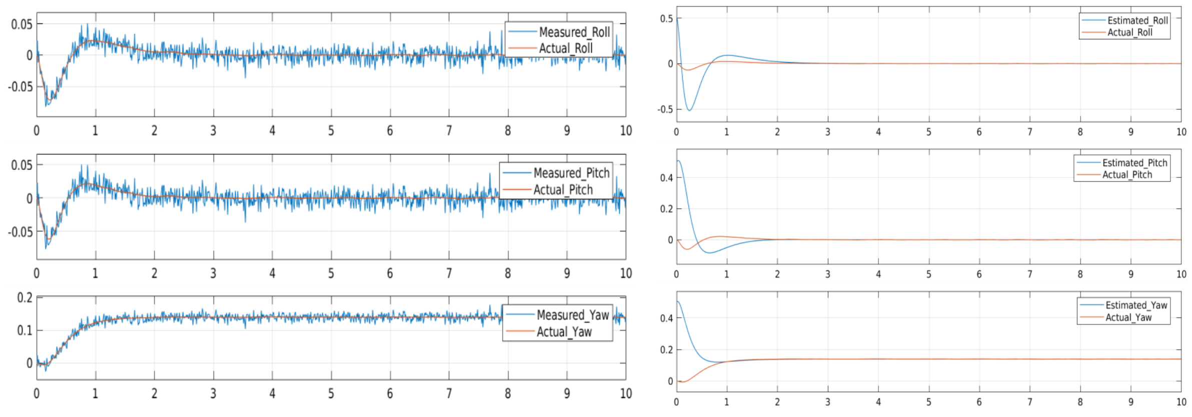


Figure 4.11: Simulink Implementation.

- Results



(a) Measured vs Actual States (b) Estimated vs Actual States

Figure 4.12: Actual, Measured and Estimated States.

Kalman Filter Parameters				
	Initial value	Converging time	Q Parameter	R Parameter
Estimated Roll	0.5	≈2.2s	10^{-3}	10^{-4}
Estimated pitch	0.5	≈1.8s	10^{-3}	10^{-4}
Estimated yaw	0.5	≈0.8s	10^{-3}	10^{-4}

Table 4.3: Kalman Filter Parameters.

Figure 4.12 shows the results of the kalman filter implemented on Matlab and Simulink (Figure 4.11) and applied to the system subject to two Gaussian distributed noises v, and w. The Figure 4.12a shows the actual response (in orange) and the measured response (in blue). The measured response is the response affected by the noises and the actual response is the one that the filter should estimate. The filtered output (in blue) is obtained in the Figure 4.12b and is the estimated output of the filter. Notice that even if the states initial conditions of the filter (0.5) are different from the ones of the actual system(0), the filter output states successfully converge to the actual ones. Table 4.3 presents the approximate converging time of the estimated states to the actual ones. The Q parameter represents the process noise variance associated with the noise in the states. A large value of Q leads the kalman filter to track large changes in the states (large uncertainties in the states equations) which is equivalent to trust less the estimated data, meaning that the filter will correct more with the measurement update. The R parameter represents the measurement noise variance and define how much information from the measurement is used. A high value tells to the Kalman filter that the measurements are not very accurate and that the filter should correct less with the measurement update.

4.3 Nonlinear Approach

Nonlinear systems are a class of systems that does not respect the principle of superposition. The outputs of the system do not vary directly with respect to its inputs. The equations of motion derived in the second chapter permits to categorize our system as nonlinear. In Section 4.2 we studied the behavior of our quadrotor and then modify this behavior via two different control methods. The problem of the analysis and previous methods is that they are only valid in a region around this equilibrium point. In addition to the previous downside, robustness is merely achieved because of the non consideration of several non linearities. The goal of this part is to present a nonlinear approach of the problem and investigate the sliding mode control method for our vehicle.

4.3.1 Non Linear Control and Analysis via Sliding Mode Control theory

Sliding mode is a control method that comes from variable structure control (VSC), that is a form of discontinuous nonlinear control. The method alters the dynamics of a nonlinear system by application of a high-frequency switching control. The state-feedback control law is not a continuous function of time; it switches from one smooth condition to another. The goal of sliding mode controls is to drive the system states onto a particular surface in the state space (sliding surface). Once the surface is reached the controller keeps the states on the close neighbourhood of the sliding surface [21].

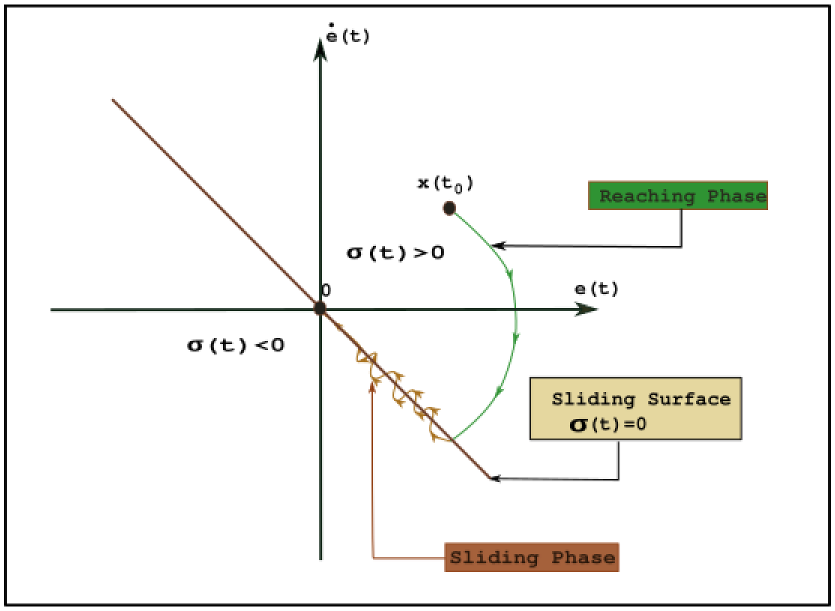


Figure 4.13: Sliding mode phase portrait.

4.3.1.1 Sliding mode Controller Design

- Sliding Surface Design $\sigma(x)$:

  This first phase consist on defining the sliding surface that is a scalar function of the system state.

$$\sigma(x) : \mathbb{R}^n \rightarrow \mathbb{R}$$

.

This is the most sensitive part of the controller design because the sliding surface represents the compensate dynamics we want to achieve. In order to achieve asymptotic convergence of the our state variables to zero in the presence of disturbance we have to drive σ to zero in finite time using the control input U. Most of the time, the sliding surface depends on the tracking error e together with a certain number of its derivatives

$$\sigma = \sigma(e, \dot{e}, ..., e^k)$$

The typical choice for the sliding manifold is a linear combination of the type:

$$\sigma = e^k + \sum_{i=0}^{k-1} c_i e^i$$

where $k = r - 1$ with r the input output relative degree.

- Control Input Design

  Step 1:

  The first step involves the design of a switching function so that the system motion on the sliding manifold (termed the sliding motion) satisfies the design specifications. This is done by choosing an appropriate Lyapunov function $V = f(\sigma)$ that prove stability through the application of a particular input.

  Step 2:

  The second step is concerned with the selection of a control law, which will make the sliding manifold attractive to the system states in the presence of external and internal disturbances and uncertainties. Note that this control law is not necessarily discontinuous.

Let's assume angles variations. The previous assumptions lead to the following equations of motion:

$$\begin{bmatrix} \ddot{x}_I \\ \ddot{y}_I \\ \ddot{z}_I \end{bmatrix} = \begin{bmatrix} \frac{1}{m}[\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi]U_1 \\ \frac{1}{m}[\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi]U_1 \\ \frac{1}{m}[\cos\phi\cos\theta]U_1 - g \end{bmatrix} \tag{4.40}$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{I_{xx}}[(I_{yy} - I_{zz})\dot{\theta}\dot{\psi} + U_2] \\ \frac{1}{I_{yy}}[(I_{zz} - I_{xx})\dot{\phi}\dot{\psi} + U_3] \\ \frac{1}{I_{zz}}[(I_{xx} - I_{yy})\dot{\phi}\dot{\theta} + U_4] \end{bmatrix} \tag{4.41}$$

As stated in section 4.1, the system is defined except at the gimbal lock(pitch angle is 90 degrees), yet we no more limit the control technique to the assumption of small oscillations and the equilibrium point. The following subsection is dedicated to find a controller for the altitude and attitude of the quadrotor.

4.3.1.2 Quadrotor Control Using Sliding Mode

- Choice of the Sliding surface in function of the tracking error:

  Errors:

  Altitude Error:

$$e_z = z_d - z \tag{4.42}$$

  Roll Error:

$$e_\phi = \phi_d - \phi \tag{4.43}$$

  Pitch Error:

$$e_\theta = \theta_d - \theta \tag{4.44}$$

Yaw Error:

$$e_\psi = \psi_d - \psi \tag{4.45}$$

Sliding surfaces:

Altitude Sliding Surface:

$$\sigma_z = \dot{e}_z + \lambda_z e_z \tag{4.46}$$

Roll Sliding Surface:

$$\sigma_\phi = \dot{e}_\phi + \lambda_\phi e_\phi \tag{4.47}$$

Pitch Sliding Surface:

$$\sigma_\theta = \dot{e}_\theta + \lambda_\theta e_\theta \tag{4.48}$$

Yaw Sliding Surface:

$$\sigma_\psi = \dot{e}_\psi + \lambda_\psi e_\psi \tag{4.49}$$

The λ_s here represent the slope of the sliding surface $(\lambda > 0)$

- Choice of Lyapunov functions proving Stability

Altitude Case:

$$V(\sigma_z) = \frac{1}{2}\sigma_z^2 \tag{4.50}$$

Roll Case:

$$V(\sigma_\phi) = \frac{1}{2}\sigma_\phi^2 \tag{4.51}$$

Pitch Case:

$$V(\sigma_\theta) = \frac{1}{2}\sigma_\theta^2 \tag{4.52}$$

Yaw Case:

$$V(\sigma_\psi) = \frac{1}{2}\sigma_\psi^2 \tag{4.53}$$

Using Lyapunov stability criteria,The previous Lyapunov function must satisfy

1.
$$\lim_{|\sigma|\to\infty} V(\sigma_\alpha) = \infty,\ \forall\ \alpha = \phi, \theta\ or\ \psi$$

2.
$$\dot{V}(\sigma_\alpha) < 0 \quad \sigma \neq 0,\ \forall\ \alpha = \phi, \theta\ or\ \psi$$

According to the second condition,

$$\dot{V}(\sigma_\alpha) < 0 \to \sigma_\alpha \dot{\sigma_\alpha} < 0$$

The stability condition can be obtained as follows:

$$\dot{\sigma}_z = -\ddot{z} + \ddot{z}_d + \lambda_z \dot{e}_z = f_1 - \frac{cos(\phi)cos(\theta)}{m} u_1 + \lambda_z \dot{e}_z + \ddot{z}_d \tag{4.54}$$

$$\dot{\sigma}_\phi = -\ddot{\phi} + \ddot{\phi}_d + \lambda_\phi \dot{e}_\phi = -f_2 - \frac{1}{I_{xx}} u_2 + \lambda_\phi \dot{e}_\phi + \ddot{\phi}_d \tag{4.55}$$

$$\dot{\sigma}_\theta = -\ddot{\theta} + \ddot{\theta}_d + \lambda_\theta \dot{e}_\theta = -f_3 - \frac{1}{I_{yy}} u_3 + \lambda_\theta \dot{e}_\theta + \ddot{\theta}_d \tag{4.56}$$

$$\dot{\sigma}_\psi = -\ddot{\psi} + \ddot{\psi}_d + \lambda_\psi \dot{e}_\psi = -f_4 - \frac{1}{I_{zz}} u_4 + \lambda_\psi \dot{e}_\psi + \ddot{\psi}_d \tag{4.57}$$

with:

$$f_1 = g \tag{4.58}$$

$$f_2 = \dot{\theta}\dot{\psi} \frac{I_{yy} - I_{zz}}{I_{xx}} \tag{4.59}$$

$$f_3 = \dot{\phi}\dot{\psi} \frac{I_{zz} - I_{xx}}{I_{yy}} \tag{4.60}$$

$$f_4 = \dot{\phi}\dot{\theta} \frac{I_{xx} - I_{yy}}{I_{zz}} \tag{4.61}$$

The next step consists on choosing the control inputs u_2, u_3, u_4 such that the second condition of the lyapunov criteria is satisfied. This lead to the following control inputs:

$$u_1 = \frac{m(g + \lambda_z \dot{e}_z + \ddot{z}_d + v_1)}{cos(\phi)cos(\theta)} = u_{eq_1} + v_1 \tag{4.62}$$

$$u_2 = I_{xx}[(-\dot{\theta}\dot{\psi}\frac{I_{yy} - I_{zz}}{I_{xx}}) + \lambda_\phi \dot{e}_\phi + \ddot{\phi}_d + v_2] = u_{eq_2} + v_2 \tag{4.63}$$

$$u_3 = I_{yy}[(-\dot{\phi}\dot{\psi}\frac{I_{zz} - I_{xx}}{I_{yy}}) + \lambda_\theta \dot{e}_\theta + \ddot{\theta}_d + v_3] = u_{eq_3} + v_3 \tag{4.64}$$

$$u_4 = I_{zz}[(-\dot{\phi}\dot{\theta}\frac{I_{xx} - I_{yy}}{I_{zz}}) + \lambda_\psi \dot{e}_\psi + \ddot{\psi}_d + v_4] = u_{eq_4} + v_4 \tag{4.65}$$

v_1, v_2, v_3, v_4: are the corrective controls which compensate the deviations from their respective sliding surface in order to reach this later. The most common and simplest function used to achieve this task is the signum function along with a tuning parameter that defines the rate of convergence to the surface.

$$v_\alpha = K_\alpha sgn(s_\alpha),\ \alpha = \phi,\ \theta\ or\ \psi.$$

It is good to notice that when the system reaches the sliding surface, its dynamics becomes the one described by the sliding surface that has been proven to be stable via lyapunov. The problem of this function is that it exhibit high-frequency oscillations near the surface.

$$sgn(s_\alpha) = \begin{cases} +1, & \sigma_\alpha > 0 \\ 0, & \sigma_\alpha = 0 \\ -1, & \sigma_\alpha < 0 \end{cases} \tag{4.66}$$

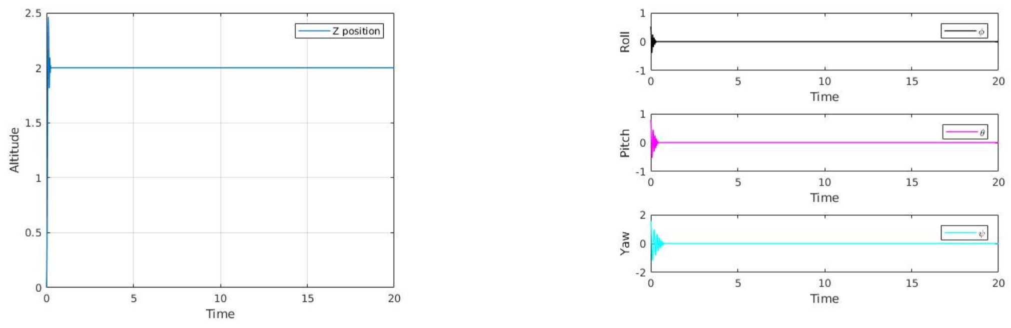
This is called the chattering effect. In order to deal with this problem the sign function is approximated as as follow:

$$sgn(s_\alpha) \approx \frac{s_\alpha}{|s_\alpha| + \epsilon_\alpha} \tag{4.67}$$

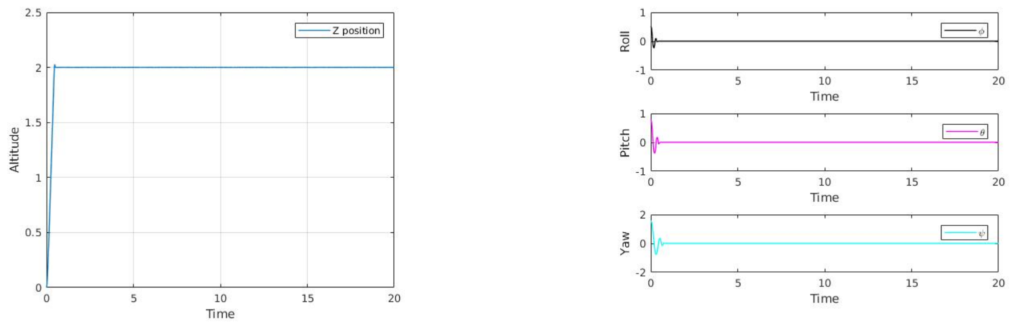
The parameter ϵ_α is responsible of the chattering reduction, in fact a small value of this parameter leads back to the original sign function. u_{eq_1}, u_{eq_2}, u_{eq_3} are respectively the equivalent controls of the altitude,roll,pitch and yaw. They each represents the continuous part of their respective sliding mode controllers and make the derivative of the sliding surface equal to zero in order for the states to stay on their respective sliding surface.

4.3.1.3 Sliding Mode Results

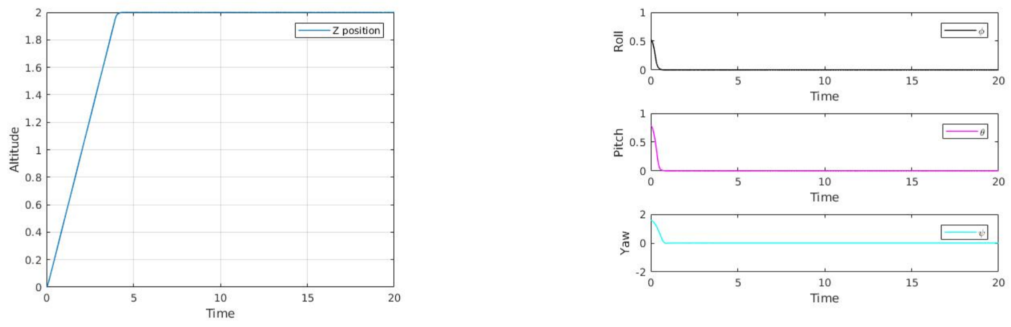
The following figures illustrate the results obtained via simulation for the sliding mode controller. Figure 4.14 presents the results with no explicit chattering attenuation parameter. The discrete part of the controller is implemented using the signum function $v_\alpha = K_\alpha sgn(s_\alpha)$. The K parameters define how fast the respective states converge to their respective surface, with high values producing more oscillations. The λ_α parameters represent the slope of the surface that also affect the rate of convergence of the states since the sliding manifold represents a desired dynamic for our system. We chose to fix the slope for each of the three states and just varied their respective K parameter. Lower values of K lead to a considerable reduction of the chattering effect at a cost of robustness. In fact for $k < 10$ the states failed reach their desired final values($z = 2$,$\phi = 0$,$\theta = 0$,$\psi = 0$) for initial values ($z = 0$,$\phi = \frac{\pi}{6}$,$\theta = \frac{\pi}{4}$,$\psi = \frac{\pi}{2}$). In the other side, Figure 4.15 shows the results obtained using the discrete part of the controller as defined in equation (4.67) for same values of K and λ. The tuning parameters ϵ were used to reduce oscillations. The higher they are the lower are the oscillations, yet the trade off between chattering and robustness is still present with a slight difference with the previous case. In fact, we can just play with the ϵ parameter to damp our response and reduce the oscillations for a fixed value of the gain K.



$K_z = 1000; K_\phi = 1000, K_\theta = 1000, K_\psi = 1000. \quad \lambda_z = 20; \lambda_\phi = 2.5, \lambda_\theta = 2.5, \lambda_\psi = 2.5$

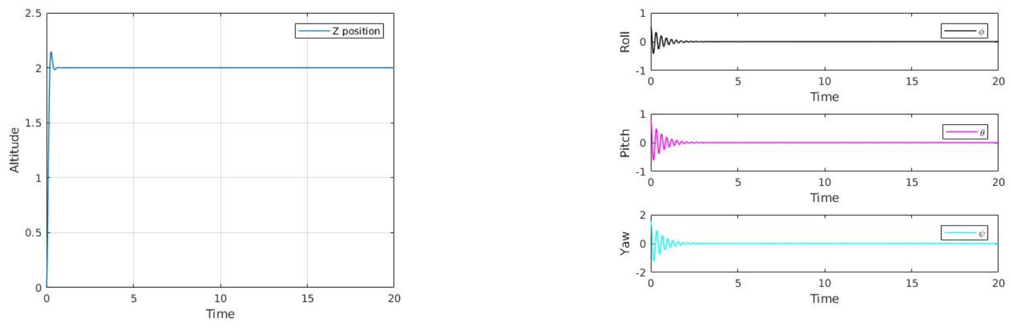


$K_z = 100; K_\phi = 100, K_\theta = 100, K_\psi = 100. \quad \lambda_z = 20; \lambda_\phi = 2.5, \lambda_\theta = 2.5, \lambda_\psi = 2.5$

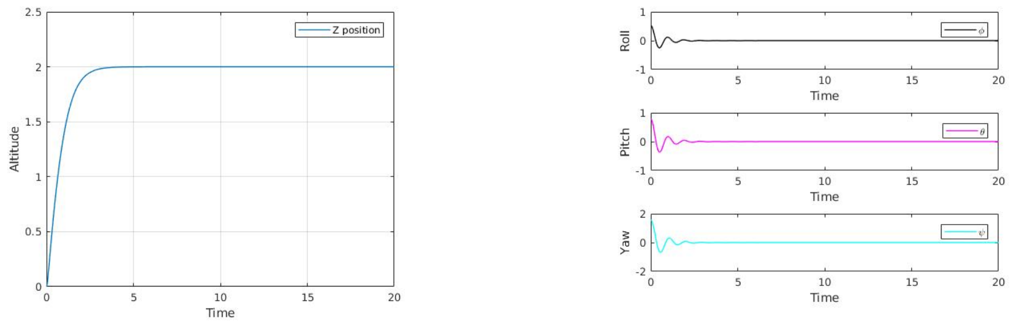


$K_z = 10; K_\phi = 10, K_\theta = 10, K_\psi = 10. \quad \lambda_z = 20; \lambda_\phi = 2.5, \lambda_\theta = 2.5, \lambda_\psi = 2.5$

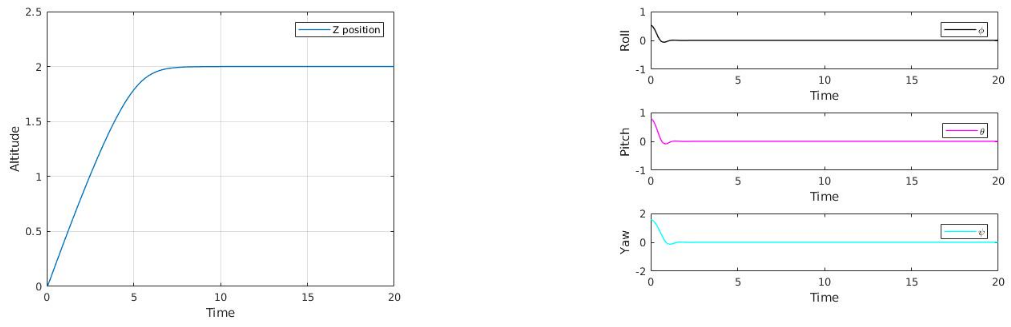
Figure 4.14: Response with chattering.



$K_z = 1000;\ K_\phi = 1000\ , K_\theta = 1000, K_\psi = 1000.\quad \lambda_z = 20;\ \lambda_\phi = 2.5,\ \lambda_\theta = 2.5,\ \lambda_\psi = 2.5$
$\epsilon_z = 3000;\ \epsilon_\phi = 2000\ , \epsilon_\theta = 2000, \epsilon_\psi = 2000.$



$K_z = 100;\ K_\phi = 100,\ K_\theta = 100, K_\psi = 100.\quad \lambda_z = 20; \lambda_\phi = 2.5,\ \lambda_\theta = 2.5, \lambda_\psi = 2.5$
$\epsilon_z = 30;\ \epsilon_\phi = 20\ , \epsilon_\theta = 20, \epsilon_\psi = 20.$



$K_z = 10;\ K_\phi = 10\ , K_\theta = 10, K_\psi = 10.\quad \lambda_z = 20;\ \lambda_\phi = 2.5,\ \lambda_\theta = 2.5,\ \lambda_\psi = 2.5$
$\epsilon_z = 3;\ \epsilon_\phi = 2\ , \epsilon_\theta = 2, \epsilon_\psi = 2.$

Figure 4.15: Response with chattering reduction.

4.3.2 Nonlinear states estimators

Nonlinear states estimators are a type of estimators designed to deal with nonlinear systems. In the subsections 4.2.8.4 and 4.2.8.5 we presented and designed a kalman filter for our linearized system. We present in this section an extension of this filter for nonlinear systems. The extended kalman filter is widely used in the realm of robotics and combines the technique of linearization (at each time steps) to preserve the Gaussian properties of the estimated noise. In fact, it linearizes the system at each stage by employing the best estimates of the state vector as the reference values.

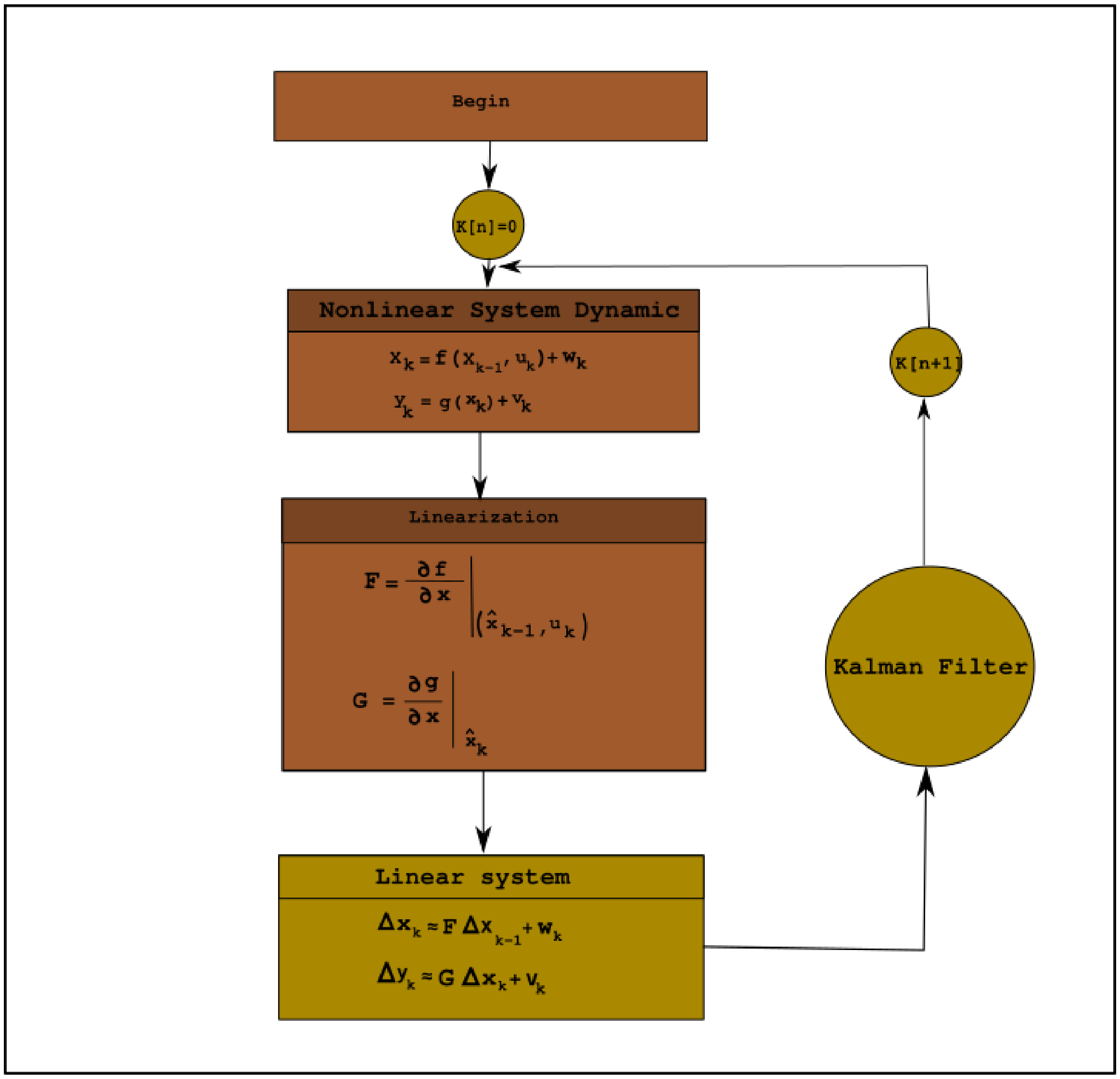
4.3.2.1 Extended Kalman Filter

let's consider the following nonlinear dynamic:

$$x_k = f(x_{k-1}, u_k) + w_k \tag{4.68}$$

$$y_k = g(x_k) + v_k \tag{4.69}$$

where $f(x_{k-1}, u_k)$ and $g(x_k)$ represent respectively the nonlinear state transition and measurement functions.Kalman filtering assume Gaussian distribution of the uncertainties in both the model and measurements. When the system is linear the state transition and measurement functions are both linear and preserve the Gaussian distribution property,because of superposition principle of linear systems. This is not the case for nonlinear systems. To take advantage of the kalman filter assets, a linearization process is done at every time steps to approximate the nonlinear system and estimate the states.



Begin
K[n]=0
Nonlinear System Dynamic
$x_k = f(x_{k-1}, u_k) + w_k$
$y_k = g(x_k) + v_k$
Linearization
$F = \frac{\partial f}{\partial x}\Big|_{(\hat{x}_{k-1}, u_k)}$
$G = \frac{\partial g}{\partial x}\Big|_{\hat{x}_k}$
Linear system
$\Delta x_k \approx F \Delta x_{k-1} + w_k$
$\Delta y_k \approx G \Delta x_k + v_k$
Kalman Filter
K[n+1]

Figure 4.16: Extended Kalman filter flow-diagram.

The following equations summarize the behavior of the extended kalman filter with similar steps as the one of the kalman filter studies in the previous chapter.

- Prediction Phase

$$\hat{x}^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{4.70}$$

$$P_k^- = A_k P_{k-1} A_k^T + w_k Q_{k-1} w_k^T \tag{4.71}$$

- Update Phase

$$\hat{x}'_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \tag{4.72}$$

$$\hat{P}'_k = P_k^- - K_k H_k P_k^- \tag{4.73}$$

$$K = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \tag{4.74}$$

with:

$$A_k = \frac{\partial f}{\partial x}(\hat{x}_{k-1}, u_{k-1}, 0)$$
$$w_k = \frac{\partial f}{\partial w}(\hat{x}_{k-1}, u_{k-1}, 0)$$
$$H_k = \frac{\partial h}{\partial x}(x_k, 0)$$
$$V_k = \frac{\partial h}{\partial x}(x_k, 0)$$

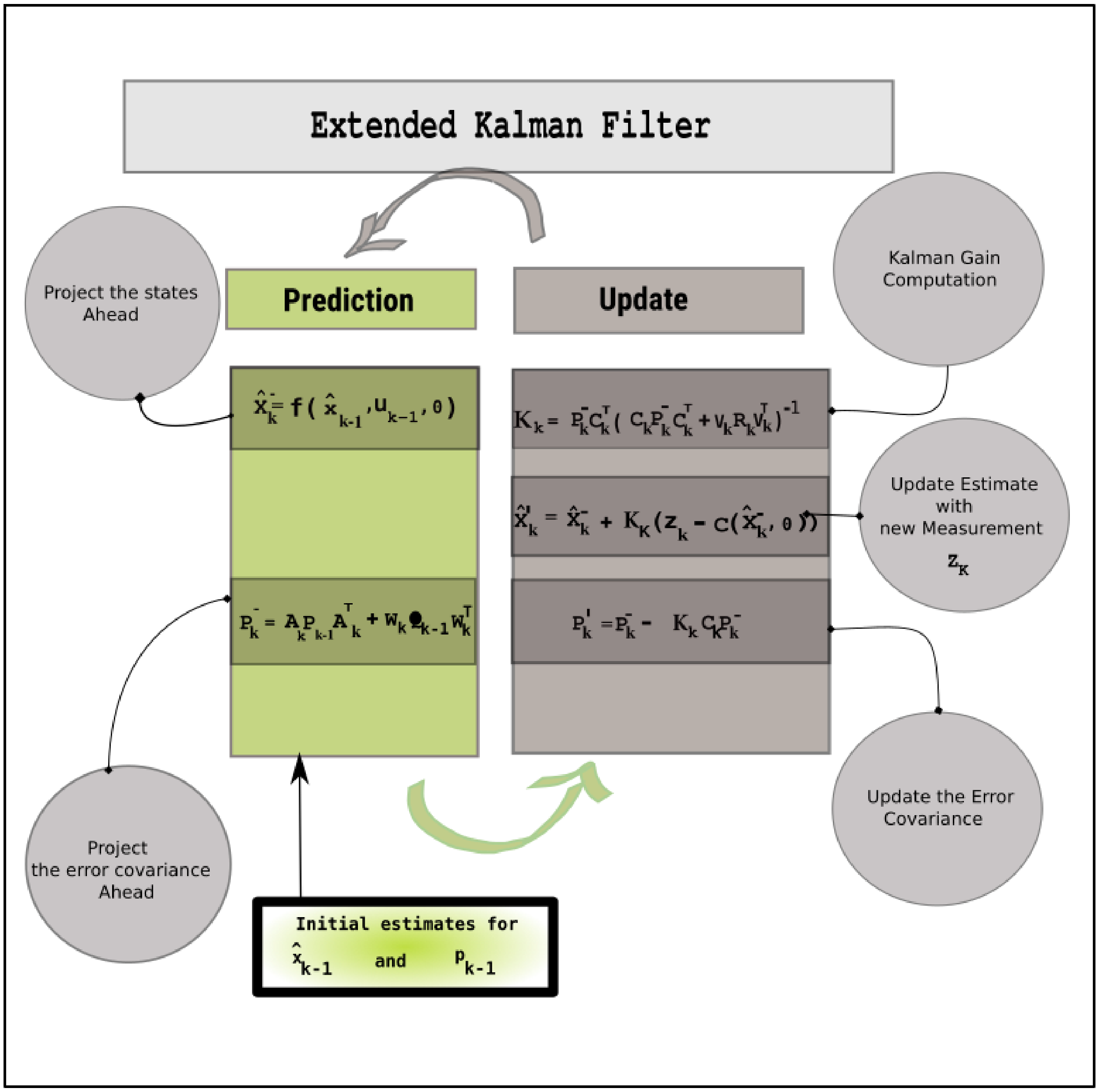

Extended Kalman Filter
Prediction
Update
Project the states Ahead
Kalman Gain Computation
$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$
$K_k = P_k^- C_k^T (C_k P_k^- C_k^T + V_k R_k V_k^T)^{-1}$
Update Estimate with new Measurement z_k
$\hat{x}_k' = \hat{x}_k^- + K_k(z_k - c(\hat{x}_k^-, 0))$
$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
$P_k' = P_k^- - K_k C_k P_k^-$
Project the error covariance Ahead
Update the Error Covariance
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Figure 4.17: Extended Kalman filter diagram.

4.3.2.2 Extended Kalman Filter Results

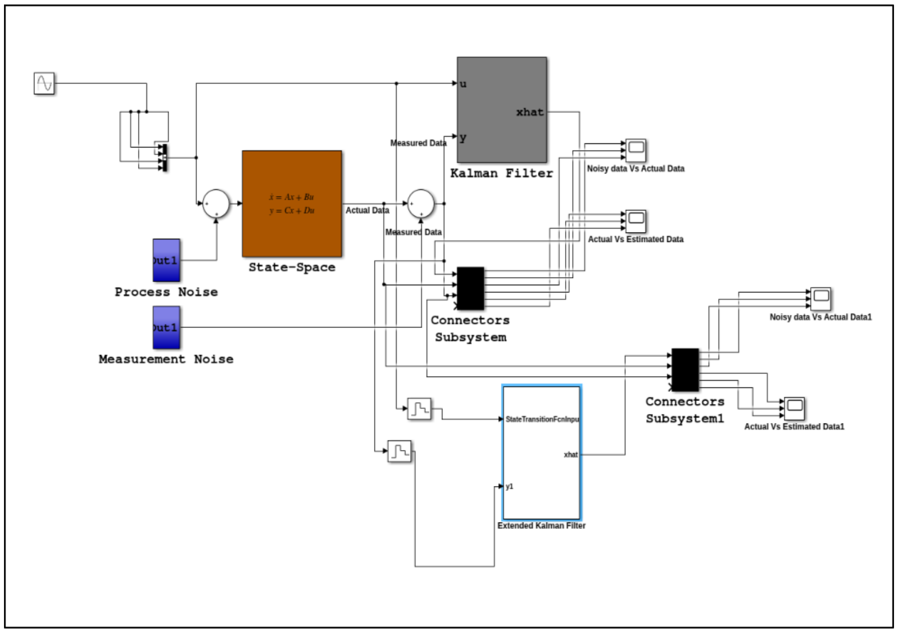
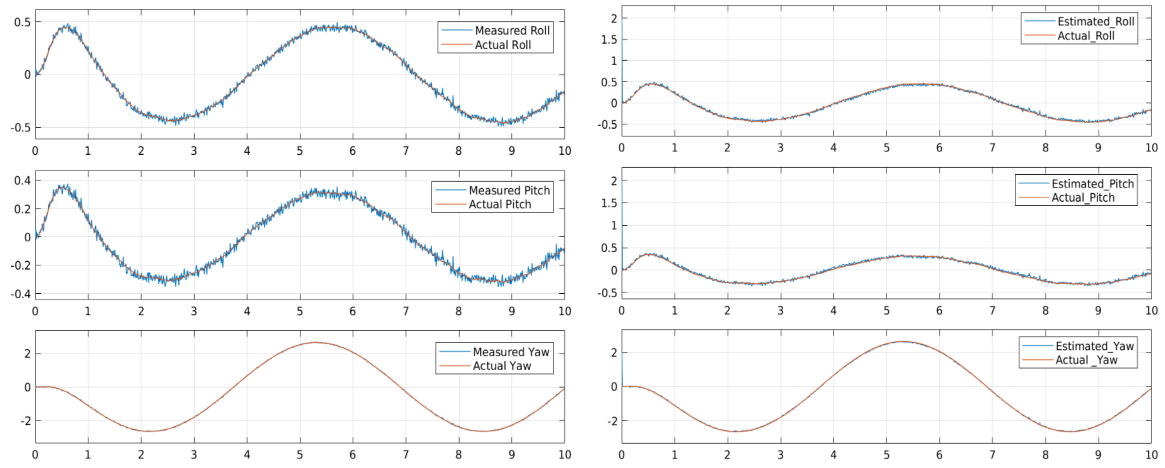


Figure 4.18: Simulink Implementation.



(a) Measured vs Actual States (b) Estimated vs Actual States

Figure 4.19: Actual, Measured and Estimated States.

Extended Kalman Filter Parameters				
	Initial value	Converging time	Q Parameter	R Parameter
Estimated Roll	2	∞	$0.75 * 10^{-3}$	$0.5 * 10^{-3}$
Estimated pitch	2	∞	$0.75 * 10^{-3}$	$0.5 * 10^{-3}$
Estimated yaw	2	∞	$0.75 * 10^{-3}$	$0.5 * 10^{-3}$

Table 4.4: Extended Kalman Filter Parameters.

Figure 4.19 illustrates the results obtained from Matlab and Simulink design (Figure 4.18). Figure 4.19a shows the actual response depicted in orange and the measured response in blue. The filtered output is presented in blue in Figure 4.19b and is the estimated output of the filter. The initial values for the EKF are set to 2 meanwhile the one of the system are set to zero. note that the filter here fails to converge exactly to the actual states but succeed in considerably reducing the error between the estimated states and the actual ones. This error can considerably be reduced by adding other sensors to our system or by finely tuning the parameters Q and R. Table 4.4 summarizes the parameters used for the simulation. The next chapter illustrates how this filter can be used in robotics applications to fuse data from different sensors and improve the estimation.

4.4 System Analysis and Control Methods Conclusion

The previous control methods simulations were done using the quadrotor parameters of table 4.5. Table 4.6 summarizes the assets and drawbacks (already mentioned during the design and analysis) of the control and estimation methods explored. The three control methods showed satisfying results regarding their capacity to stabilize and control the system. Robustness is achieved using the sliding mode controller, yet at a cost of energy consumption due to the chattering effect. The PID controller in addition to be more intuitive and easy to implement is the most spread controller in autopilots systems, yet it lacks of robustness that leads to time consuming and tedious manual tuning. The LQR controller succeeds to reduce the tuning process, yet is less intuitive and lack of implemented libraries for autopilots. Regarding the estimations methods, the Luenberger observer has an advantage regarding its simplicity, yet is only used for deterministic cases leading to a low capacity to reject noises. Kalman and Extended Kalman Filters have the asset to address the problem of noise rejection (Gaussian). The problem of nonlinear states estimation is solved using the Extended kalman filter, but this latter is computationally expensive due to the permanent linearization process.

Quadrotor parameters and values		
Parameters	Values	Unit
g	9.8	m/s^2
masse	1.920	Kg
Lenght	225	mm
I_x	0.04989	$kg-m^2$
I_y	0.04989	$kg-m^2$
I_z	0.24057	$kg-m^2$

Table 4.5: Quadrotor Parameters.

Assets and Drawbacks		
Control methods	Assets	Drawbacks
PID	Intuitive concept, Easy to implement, Available codes already implemented and used by several autopilot systems.	Not adaptive,Low robustness,Tedious manual tuning,Not Optimal, Low precision,Low Disturbance rejection, Sensitive to the model accuracy.
LQR	Adaptative,Optimal,Minimum manual tuning.	Low disturbance rejection,Low robustness, Not intuitive.
Sliding Mode	Robust, Adaptive, optimal, Accurate, Great disturbance rejection,low manual tuning,Fast convergence response.	Not intuitive,High chattering, High energy consumption.
Estimation methods	Assets	Drawbacks
luenberger observer	Intuitive concept,Easy to implement.	For linear systems, Low Noise rejection,Not Adaptive,Tuning process required, Only for deterministic cases.
Kalman Filter	Good disturbance rejection, Adaptive,Handles Gaussian noise.	Only for linear systems,Not intuitive,Tuning Process required.
Extended Kalman Filter	Great disturbance and uncertainties rejection,Adaptive,Used for nonlinear systems,Handles Gaussian noise, Available robust codes implement in autopilots systems.	Requires continuous linearization,Not Intuitive,Tuning Process required,computationally expensive.

Table 4.6: Control and Estimated Methods assets and Drawbacks.

Chapter 5

Detection, Pose Estimation and Tracking

The previous chapters served as a solid foundation to understand and master theoretical aspects related to quadrotors' control and navigation. This part gathers all the previous applications to solve the problem of autonomous targets tracking. The target is represented as a car-like robot. The first section of this chapter describes the general problem that is subdivided into three sub-problems: the target detection and localization problem, the poses estimation problems and the autonomous tracking problem. The poses estimation problems appears in two flavors: the aerial vehicle pose estimation and the ground vehicle current pose estimation and future pose prediction. The second section presents the solutions and tools used to address or mitigates these sub-problems to achieve our tracking goal. The third and fourth parts of this chapter are oriented towards simulation and physical implementation. In the third section, we begin by presenting the tools and concepts used to build the application in the simulation. These tools are the 3D simulator gazebo which integrates realistic physics and dynamics of systems; the Robot Operating System (ROS) that is a robotics middleware providing a great collection of packages already wrapping applications related to control, data fusion, visual servoing and more; and finally Ardupilot, an open source autopilot used to control vehicles. The second subsection of this same section is dedicated to the design of the application based on the previous enumerated concepts and illustrates results and limitations. The fourth and final part of this chapter shows the low-cost physical quadrotor and presents the different components used in its design along with the real implementations results.

5.1 Problem Formulation

The mission consists of using an aerial vehicle, equipped with a GPS an IMU and a down-facing camera (Figure 5.1) to follow a moving target represented by a car-like robot. The Detection and localization problem is associated to the non-cooperative constraints of the mission. This fact means that there is no explicit exchange of information between the aerial and the ground vehicle. In other words, the ground vehicle does not communicate its position, orientation, linear and angular velocities to the aerial one. The second issue, the poses estimations reveals two challenges. The first challenge is the accurate localization of the aerial vehicle in the world frame. The second challenge comes directly after the target detection and localization. In fact, the detection using the down-facing camera provides the position of the target in the camera frame that is not enough to design an efficient control method to track the moving target. The third and last challenge is related to the ground robot tracking considering the fact that any translational movement of the quadrotor necessitates a rotation around the roll or pitch axes (equations (3.54) illustrates this fact). The down-facing camera being directly mounted on the robot base-link, partial occlusion or even loss of tracking can occur. This later fact leads to the target future pose prediction problem coupled with a tracking challenge considering the car like-robot natural motion in the 2D plane. The next sub-subsection presents the car-like robot dynamics and encapsulates the pursuit challenge in a minimization problem.

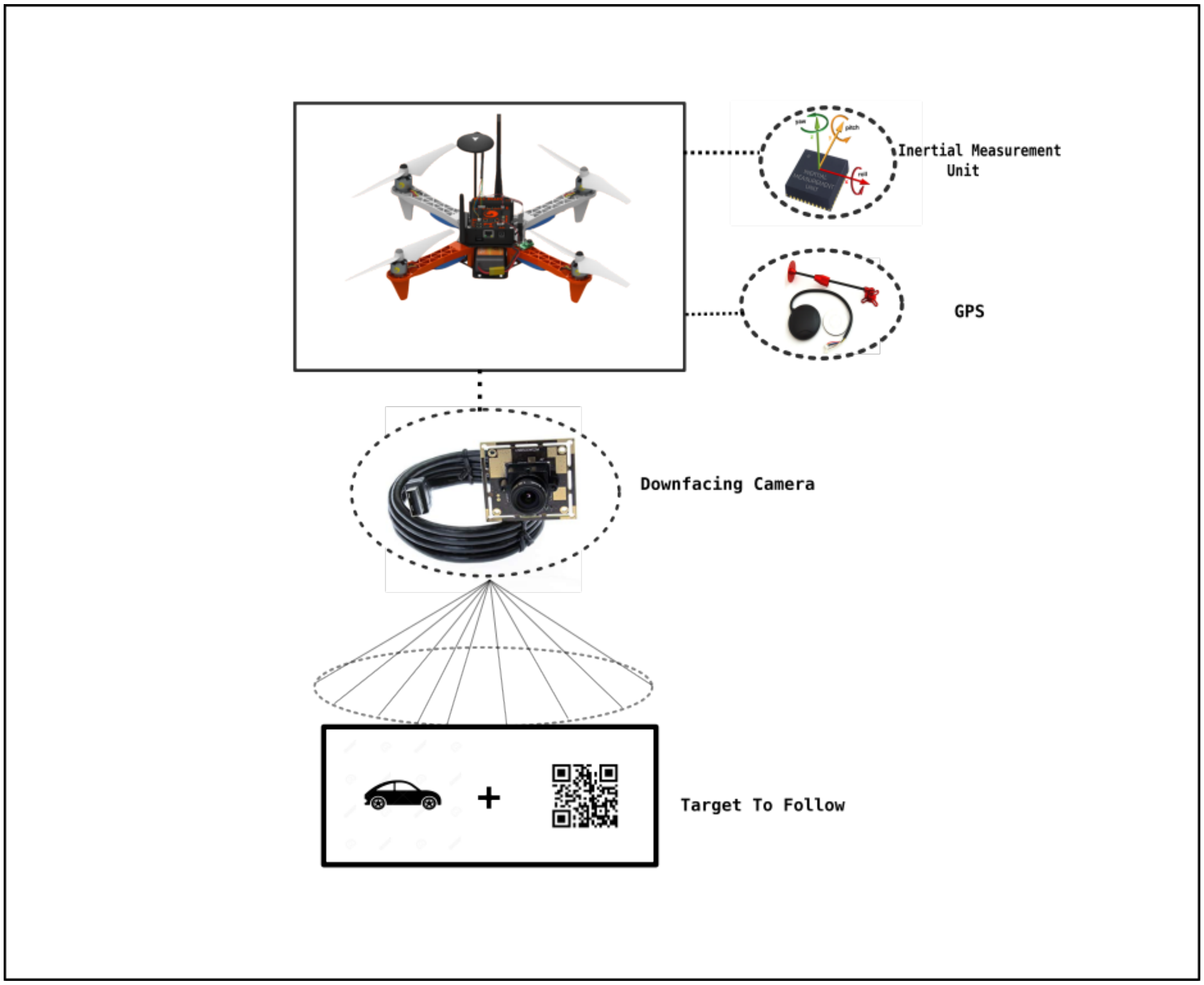


Figure 5.1: Mission structure Diagram.

5.1.1 Car-like robot Motion

To model the car, we use a bicycle model (Figure 5.2). The rear wheel is fixed to the body and the front wheel rotates about the vertical axis to steer the vehicle. The pose of the vehicle is represented by the coordinate frame $\{V\}$. The x-axis is the vehicle forward direction and its origin is at the centre of the rear axle. The vehicle can be represented by the following configuration(q=(x,y,θ)) . The vehicle's velocity is by definition ν in the vehicle x-direction and zero in the y-direction because the wheel cannot slip side ways. In the vehicle frame $\{V\}$ we have $V_{\dot{x}} = \nu$ and $V_{\dot{y}} = 0$. The ICR(Instantaneous centre of rotation) represents the intersection of the lines of no motion(along which wheels can't move). $\dot{\theta} = \frac{v}{R_1}$ is the angular velocity of the circular path of the reference point with $R_1 = \frac{L}{\tan \Upsilon}$. The kinematics model of the vehicle represented by the velocity of the robot in the world frame is.

$$\dot{x} = \nu \cos(\theta) \tag{5.1}$$

$$\dot{y} = \nu \sin(\theta) \tag{5.2}$$

$$\dot{\theta} = \frac{\nu}{L} \tan \Upsilon \tag{5.3}$$

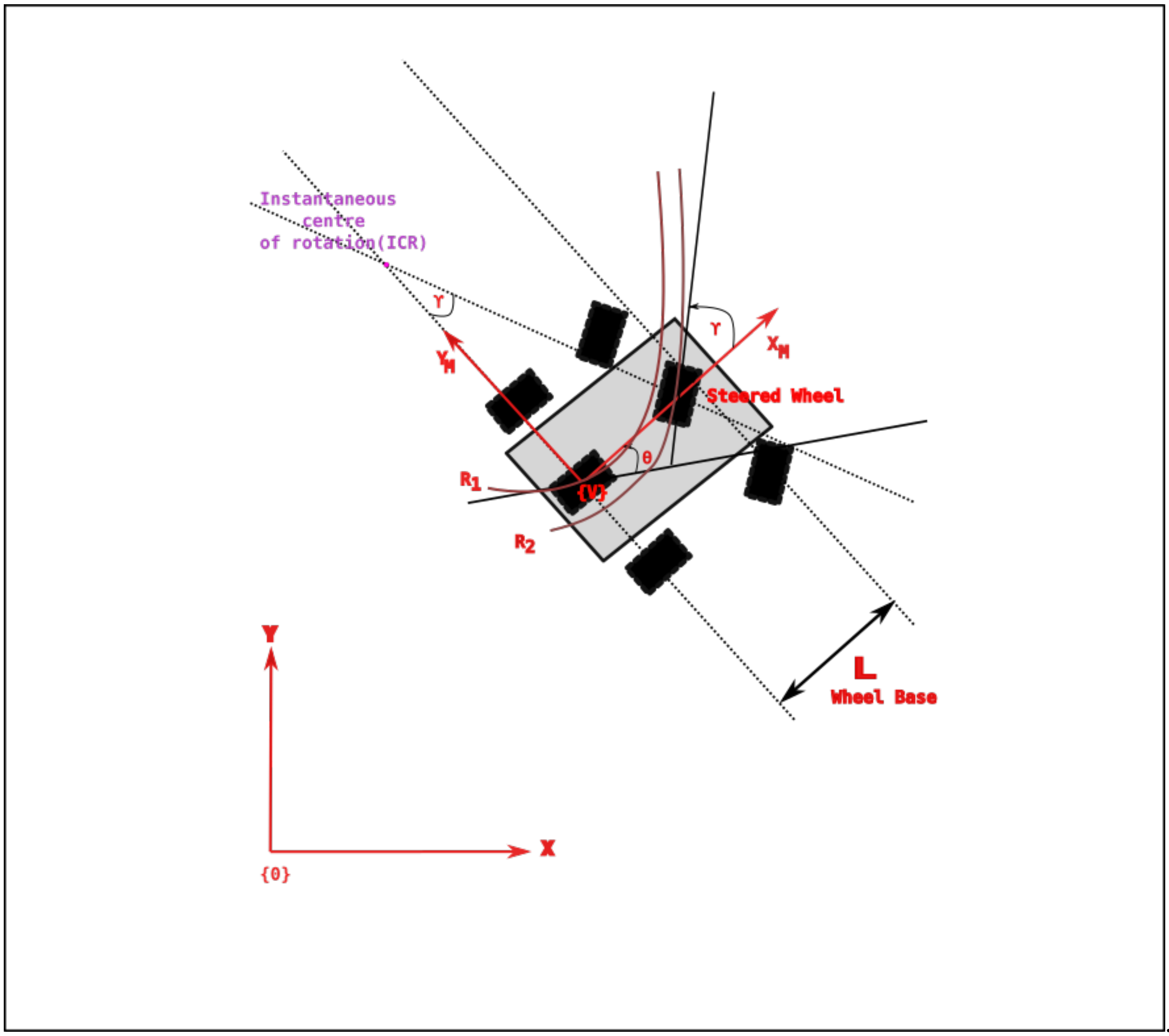


Figure 5.2: Nonholonomic Robot.

It is important to note that: for $\nu = 0$, $\dot{\theta} = 0$ meaning that we can't turn if we don't move. The model is also undefined for θ=$\frac{\pi}{2}$. Considering the kinematics of the ground vehicle,the problem can be mathematically summarized as :

$$min\left\| Drone\_pose - ({}^{World\_frame}\xi_{Target\_pose})\right\| \tag{5.4}$$

S.t

$$\dot{x}_{car} = \nu\cos(\theta) \tag{5.5}$$

$$\dot{y}_{car} = \nu\sin(\theta) \tag{5.6}$$

$$\dot{\theta}_{car} = \frac{\nu}{L}\tan\Upsilon \tag{5.7}$$

5.1.2 Visual servoing

The goal of visual servoing is to control the pose of a robot relative to a target by using visual features extracted from the image. We consider the end-point closed-loop configuration in which the camera is mounted on the base_link of the robot. The image of the target is a function of the relative pose ${}^{C}\xi_{T}$ (pose of the target with respect to the camera). Features such as coordinates of points or the parameters of lines of ellipses are extracted from the image. These features are also function of the relative pose ${}^{C}\xi_{T}$. There are two different approaches of visual servo control. Position Based Visual Servo(PBVS) and Image Based Visual Servo(IBVS). We use the PBVS technique that uses observed visual features, a calibrated camera and a known geometric model of the target to find the pose of the target with respect to the camera. Figure 5.3 illustrates the basic principle of PBVS with ${}^{C*}\xi_{T}$ the desired relative pose with respect to the camera. The goal is to find the motion required to move the camera from its initial pose to this final desired one.

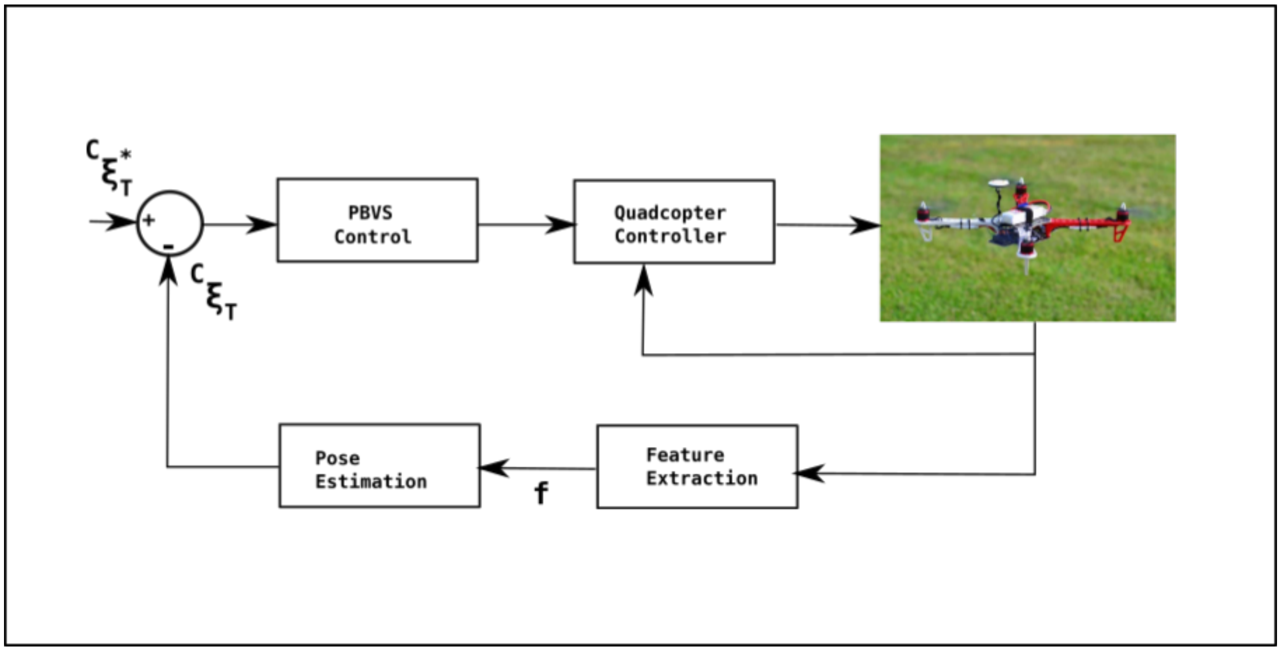


Figure 5.3: Visual Servoing.

5.1.2.1 Visual Servoing Control

5.1.2.1.1 Basic Concept:

The visual Servoing control problem can be summarized as minimizing the following error $e(t)$ where:

$$e(t) = s(m(t), a) - s^* \qquad where: \tag{5.8}$$

$m(t)$: Vector representing the set of image measurements (coordinates of interest points or the image coordinates of the centroid of an object).

a: Set of parameters representing the potential additional knowledge about the system (camera intrinsic parameter, 3-D models of objects)

s: Vector containing a set of 3-D parameters estimated from image measurements.

s^*: Vector containing the desired values of the features.

Supposing S constant, it is posible to design a basic velocity controller defined by the following equations:

$$\dot{s} = L_s V_c \tag{5.9}$$

$$\dot{e} = L_e V_c \tag{5.10}$$

$L_s \in \mathbb{R}^{k\times 6}$: Feature jacobian or interaction matrix related to s with k the number of visual features.

$V_c = (v_c, w_c)$: the spacial velocity of the camera with v_c and w_c being respectively the instantaneous linear and angular velocities of the camera's frame. Equation (5.9) relates the time variation of s and the camera velocity meanwhile equation (5.10) obtained using equations (5.8 and 5.9), relates the time variation the of error and the camera velocity.

Considering V_c as the input of the robot controller with $L_e = L_s$ an exponential decrease of the error ($\dot{e} = -\lambda e$) is obtained using V_c such that

$$V_c = -\lambda L_e^{+} e \qquad where \tag{5.11}$$

$L_e^{+} \in \mathbb{R}^{k\times 6}$ chosen as the Moore-Penrose pseudo-inverse $L_e^{+} = (L_e^T L_e)^{-1} L_e^T$. Because it is not possible to perfectly know L_e or L_e^{+} an approximation is made, thus the controller law becomes:

$$V_c = -\lambda \hat{L_e^{+}} e \qquad where \tag{5.12}$$

$\hat{L_e^{+}}$: The approximation of the speudo-inverse of the feature jacobian matrix.

The difference between the PBVS and IBVS can be captured in the definition of the vector . [12] provides deep understanding related to the different approaches. Position based visual servoing uses the pose of the camera with respect to some reference coordinate frame to define . To compute that pose from a set of measurements in one image, a 3D model of the the object

observed and the camera intrinsic parameters must be known.In equation (5.8), the parameter a captures the camera intrinsic parameters and the 3D model of the object.
The vector can be mathematically expressed to be $s = (t, \theta u)$ with t a translation vector and θu representing a angle/axis parametrization for a rotation. Defining the translation vector as $s = ({}^{c^*}t_c, \theta u)$ with $s^* = 0$, $e = s$ the interaction matrix is given in [1] as followed:

$$\begin{bmatrix} R & 0 \\ 0 & L_{\theta u} \end{bmatrix} \tag{5.13}$$

with the following control law:

$$v_c = -\lambda R^{T c^*} t_c \tag{5.14}$$

$$W_c = -\lambda \theta u \tag{5.15}$$

5.1.2.1.2 Case of a Moving target

Let's Consider the case of a moving target [13] with a constant desired value s^* for the features. The varying desired feature are denote $s^*(t)$. equation (5.10) becomes:

$$\dot{e}(t) = \dot{s} = L_e V_c + \frac{\partial e}{\partial t} \tag{5.16}$$

$\frac{\partial e}{\partial t}$ represents the time variation of e due to the generally unknown target motion. To ensure an exponential decoupled decrease of the error, we use the following equation

$$V_c = -\lambda \hat{L_e^+} e - \hat{L_e^+} \frac{\hat{\partial e}}{\partial t} \tag{5.17}$$

$\frac{\hat{\partial e}}{\partial t}$ is an estimation of $\frac{\partial e}{\partial t}$ which goal is to compensate the target motion. Injecting equation (5.17) in (5.16), gives the following time variation of the error

$$\dot{e}(t) = -\lambda L_e \hat{L_e^+} e - (L_e \hat{L_e^+}) \frac{\hat{\partial e}}{\partial t} + \frac{\partial e}{\partial t} \tag{5.18}$$

The error converges to zero if $L_e\hat{L_e^+} > 0$ and if the estimation $\hat{\frac{\partial e}{\partial t}}$ is accurate such that $(L_e\hat{L_e^+})\hat{\frac{\partial e}{\partial t}} = \frac{\partial e}{\partial t}$. [13] and [22] propose some methods such as a kalman filter or a an integral term added in classic control methods to cancel the tracking error. In the following chapter, we combine an Extended Kalman Filter with a PD controller to mitigate the the tracking error.

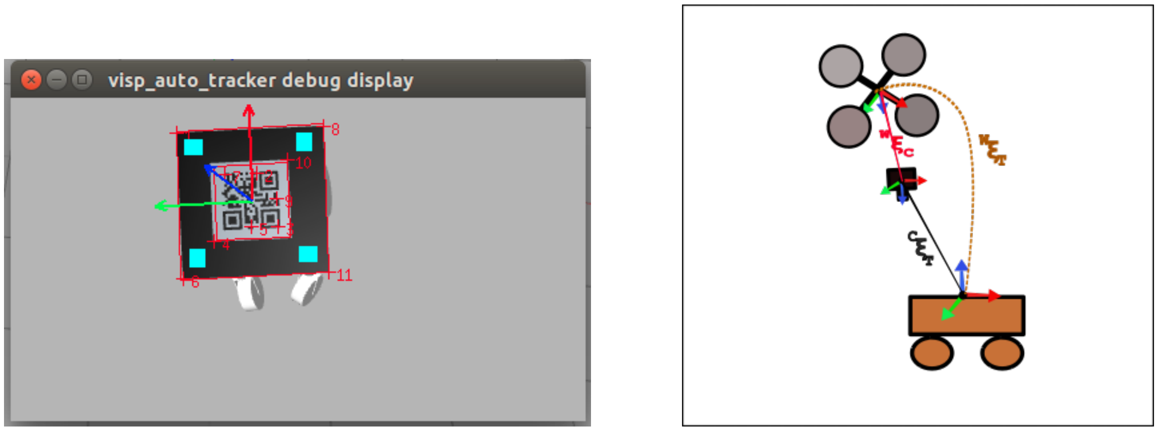
5.2 Solution

The solution is built around estimations principle and rely on the EKF algorithm. Since the aerial vehicle movements are dictated by the mobile vehicle ones, its position and velocity are highly related to this later. We proceed by first detecting the ground robot position that we transform in the aerial robot world_frame, then we use consecutive poses in the same frame to obtain the mobile robot velocity. A simple Proportional and derivative Controller is used to guide the aerial robot to this position considering the ground vehicle velocity. When loss of tracking occurs we use implicitly estimates of the pose and velocity of the mobile robot, based on the already known pose and velocity of the aerial one to continuously guide the aerial robot to the predicted future location until a new measurement from the camera is available. We recursively proceed by, first fusing the aerial robot information to acquire its position in real-time while permanently estimating the future pose of the aerial vehicle that reflects the displacement of the mobile one, summarized by the equations (5.4) to (5.7). Figure 5.6 summarizes the implemented algorithm. The following subsections explains in detail the solutions.

5.2.1 Target Detection and Localization

The detection was achieved using the open source visual servoing platform library (VISP) that is able to compute control laws which can be applied to robotic systems. This library also provides a set of visual features that can be tracked using real-time image processing or computer vision algorithms; Therefore, a QR_code was mounted on top of the ground vehicle to serve as a detection object. The outcome of the detection phase is the pose of the QR_code relative to the down-facing camera(${}^{C}\xi_T$). The localization phase directly follows the detection one and aims to obtain the target pose in the world frame in which the quadrotor global position is defined. This is achieved using the composition's property of coordinate frames (chapter 3) and is given by the following relation:

$${}^{world\_frame}\xi_T = {}^{world\_frame}\xi_C \oplus {}^{C}\xi_T$$



(a) Target Detection (b) Target Localization

Figure 5.4: Target Detection and Localization.

5.2.2 Poses Estimation

5.2.2.1 Quadrotor Pose Estimation

The quadrotor pose is obtained via the combination of the GPS (Global positioning System) and IMU (Inertial Measurement Unit) sensors data (Figure 6.4). They produce respectively the following outputs:

- GPS: Position and velocity
- IMU (Accelerometer + Gyroscope + Magnetometer):
  - Accelerometer $(\ddot{x}, \ddot{y}, \ddot{z})$: Acceleration values across the XYZ axes.

– Gyroscope $(\dot{\phi}, \dot{\theta}, \dot{\psi})$: Angular velocity about three axes: pitch (x-axis), roll (y-axis) and yaw (z-axis).

– Magnetometer (vector towards Earth's magnetic North): Tilt with respect to the Earth's magnetic vectoring, parallel to the ground.

These sensors produce different, yet complementary outputs. Using integration and fusion (via an Extended Kalman Filter) of the data from the accelerometer and gyroscope, it is possible to obtain translational and rotational positions and velocities $(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$. To obtain absolute heading, these data are fused with the magnetometer readings using again an Extended Kalman filter (subsection 4.3.3). The outcome of this first filtering process produces a continuously stream of positions and velocities, yet not accurate (regarding translational position and velocities) for long term navigation, because of the integration process that injects errors in the measurement. To correct the drift caused by the integration, Those data are fused with the GPS data that is subject to discrete jumps but are more accurate regarding these translational positions and velocities (No drift). This process permits to exploit the assets of individual sensors while tackling their respective drawback in order to produce estimated data that are robust and more accurate than the one provided by each sensor.

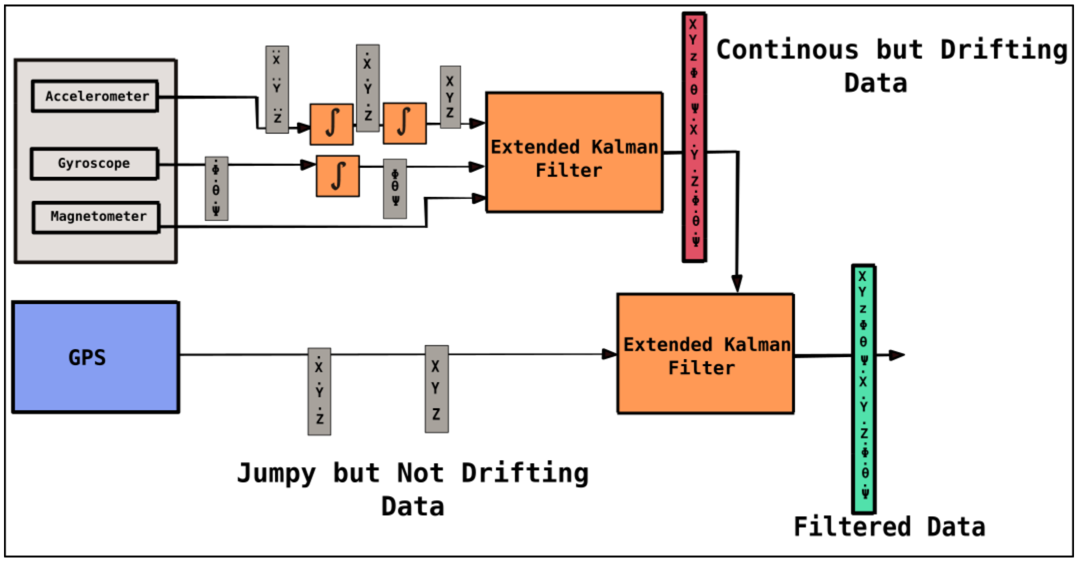


Figure 5.5: EKF for pose Estimation Diagram.

5.2.2.2 Target Pose Estimation

The previous pose estimate provides a mean to localize the aerial vehicle in real-time. In addition, it also permits to keep track of the displacement of the vehicle via odometry that aims to calculate the course of the vehicle with respect to the inertial frame. We first assume that a detection of the target was made and that the pose obtained has been transformed in the world frame (Localization). Commands are sent to the aerial vehicle to move toward this new pose. When loss of tracking occurs, we proceed by an implicit estimation to get the estimated pose of the ground vehicle. The displacement of the quadrotor being associated to the ground vehicle, its continuous pose estimates reflects the displacement of the target and which global position is refined when a new detection is made. Figure 6.5 illustrates the described procedure.

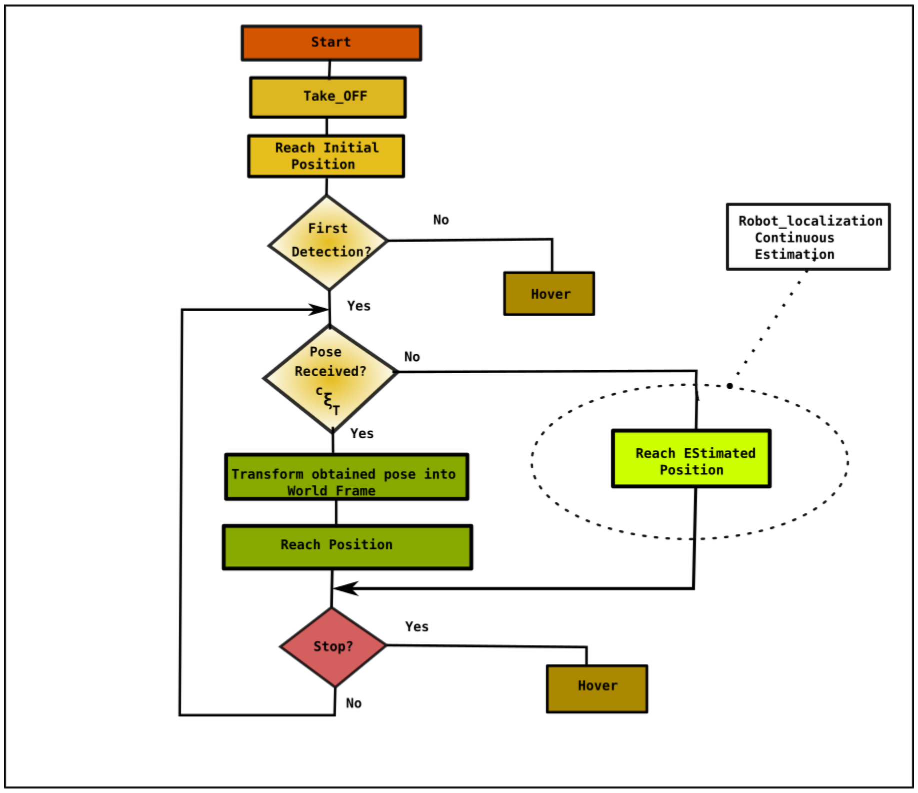


Figure 5.6: Pose Estimation and Tracking Algorithm.

5.2.3 Tracking

The tracking is done in two forms, corresponding to the presence or absence of visual data. In the first case, using two consecutive positions we estimate the velocity of the ground robot and guide the aerial robot to this position using a proportional and derivative control method.

$$P_{x(t)} = K_p * e_{x(t)} + K_D * \frac{e_{x(t)} - e_{x(t-1)}}{dt} \tag{5.19}$$

$$P_{y(t)} = K_p * e_{y(t)} + K_D * \frac{e_{y(t)} - e_{y(t-1)})}{dt} \tag{5.20}$$

With:

$P_{x(t)}$: The desired new position of the quadrotor in the x direction.

$P_{y(t)}$: The desired new position of the quadrotor in the y direction.

$e_x(t) = x(t)_{quadrotor} - x(t)_{target}$ in the world frame.

$e_y(t) = y(t)_{quadrotor} - y(t)_{target}$ in the world frame.

When no detection is made, we use an estimated pose of the aerial vehicle already calculated using an EKF to estimate the pose of the ground vehicle (the pose of the aerial one is closely related to the one of the ground). To reduce the converging error we insert the ground vehicle calculated velocity and orientation in the future desired pose where the quadrotor should go.

$$P_{x(t)} = K_p * e_{x(t)} + K_D * e_{\dot{x}(t)} \tag{5.21}$$

$$P_{y(t)} = K_p * e_{y(t)} + K_D * e_{\dot{y}(t)} \tag{5.22}$$

With:

$$e_{\dot{x}(t)} := \dot{x}(t)_{quadrotor} - \nu \cos(\theta_{target})$$

$$e_{\dot{y}(t)} := \dot{y}(t)_{quadrotor} - \nu \sin(\theta_{target})$$

5.3 Simulation

The simulation was designed around the 3D simulator gazebo coupled with the Robot Operating System (ROS) and the open source autopilot Ardupilot. The first subsection aims to present and explain these tools and concepts. The second subsection illustrates their assembly to achieve our tracking mission, along with the results obtained.

5.3.1 Basic Concepts

5.3.1.1 Gazebo

Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate robots. In addition to have a rich library, It allows the design and simulation of robots along with their sensors. By taking in account dynamics and physic properties, it provides a suitable environment to implement and test algorithms.

5.3.1.2 Robot Operating System (ROS)

ROS is an open source collection of software frameworks that provides interesting features such as hardware abstraction, low-level device control, devices drivers, libraries, visualizers, message passing between processes, packages management...It is organized into three levels of concepts that are the file system level that organizes folders and files, the computational graph level that is concerned with the organization and communication of processes and the community level that gathers all the resources that enable the ROS community to exchange software and knowledge. The following lines give a basic overview of the second level (Figure 5.7) that is the most important to master in order to build robotics applications.

- Computational graph level:

  It is organized as a peer-to-peer network of processes that are processing data together and is composed of the following entities:

  – Nodes: They are the main unit of the computation graph and are processes performing computation. A system can have many nodes with each of them responsible of a particular task.

  – Master: Its purpose is to enable nodes to locate one another and organize the communication between them. It provides naming and registration services to all the other nodes in the system.

  – Parameter Server: It is part of the master and allows data to be stored by keys.

  – Messages: They are simply data structures comprising typed fields (integer,float,boolean,string),arrays...

  – Topics: They represent the canal of communication between nodes to route messages exchanged. Nodes send out messages by publishing them via topics and acquire data by subscribing to the desired ones. It is important to note that the publish /subscribe model is a many-to-many, one-way transport.

  – Services: Services implement the request/reply interaction mode between nodes. A providing node offers a service under a name and a client uses the service by sending a request message and wait for the reply.

  – Bags: Bags are a used to save and play back ROS message data. They are an important mechanism for storing data, such as sensor data, that can be difficult to collect but necessary for developing and testing algorithms.

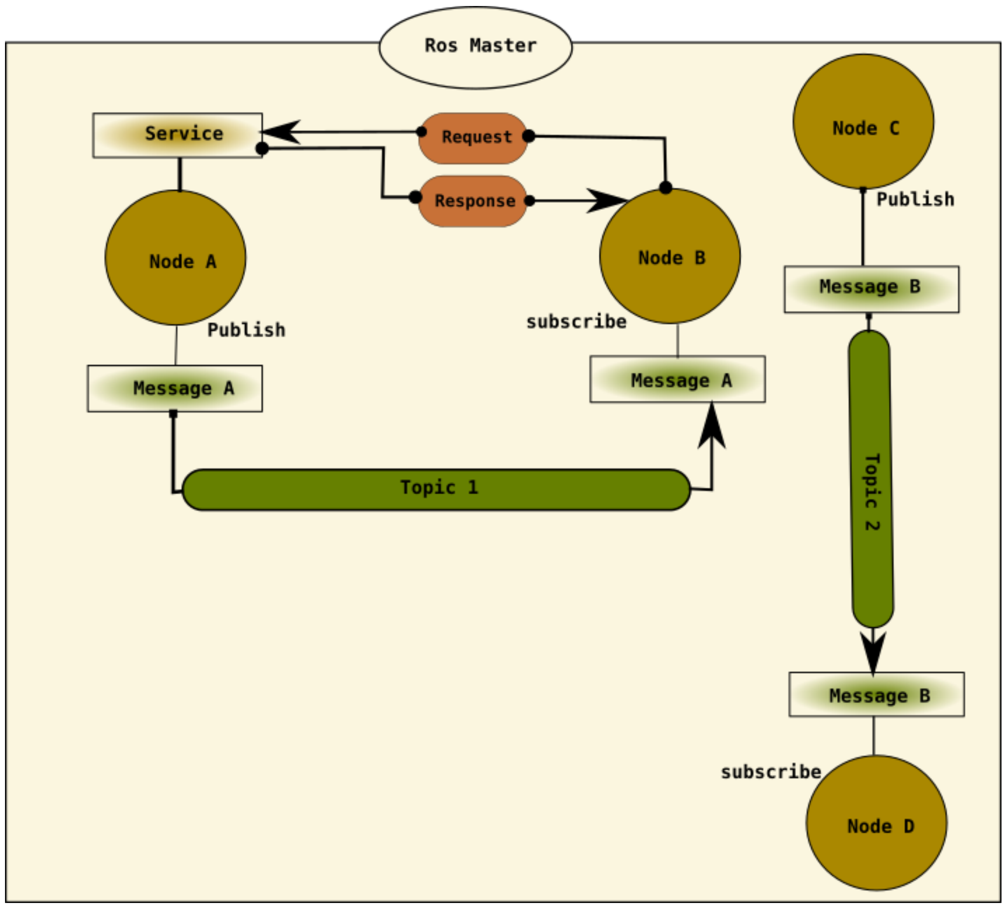


Figure 5.7: ROS Computational Graph Level.

5.3.1.3 Ardupilot Mega

Ardupilot Mega also called APM, is an open source autopilot system supporting a great variety of aerial vehicles, submarines and antenna trackers. It also supports a large numbers of flight controllers, sensors and frame types. It is composed of three main components which are:

- Hardware: It represents all the physical aspect of the vehicle and regroups output devices like ESC's, motors, gimbals..., in addition to peripheral sensors such as camera, GPS, radar, lidar, sonar..., that provide information about the vehicle environment. The hardware also includes the main controller that can be seen as the brain of the vehicle. The main controller integrates sensors such as the inertial measurement unit(IMU), pressure sensors(barometers), digital compass. To be able to estimate basic information related to the vehicle(Altitude,attitude...), together with the peripheral sensors data, the controller is capable of awareness about the environment in which it evolve, to send commands to the outputs to achieve a specific goal.

- Firmware: The firmware is running on the controller and represents a code defining a "skill set", representing the controller embedded capabilities.
- software: It is represented by a Ground Control Station(GCS) running on PC, mobile or tablets. It serves as an interface between the controller and the user and permits to set-up,test,tune the vehicle,assign missions....

The following paragraphs gives an overview of the autopilot and presents some features used in our Mission.

5.3.1.4 Ardupilot Basic Structure

Ardupilot can be decomposed into 5 main parts (Figure 5.8) that are:

- The vehicle Code: This is a group of directories that define the firmware for each vehicle type
  (Copter,Plane,APMrover2,Antenna-Tracker).
- Libraries: Libraries include sensor drivers, attitude and position estimation along with control codes.
- Hardware Abstraction Layer (HAL): This layer is responsible of the portability of the autopilot to several platforms.
- Tools directories: The tools directories represent various support directories, which support include tests,CPU Info and others various tools.
- External Support Code: Represents specific codes for some platforms providing additional features(Mavlink for example) or board support.

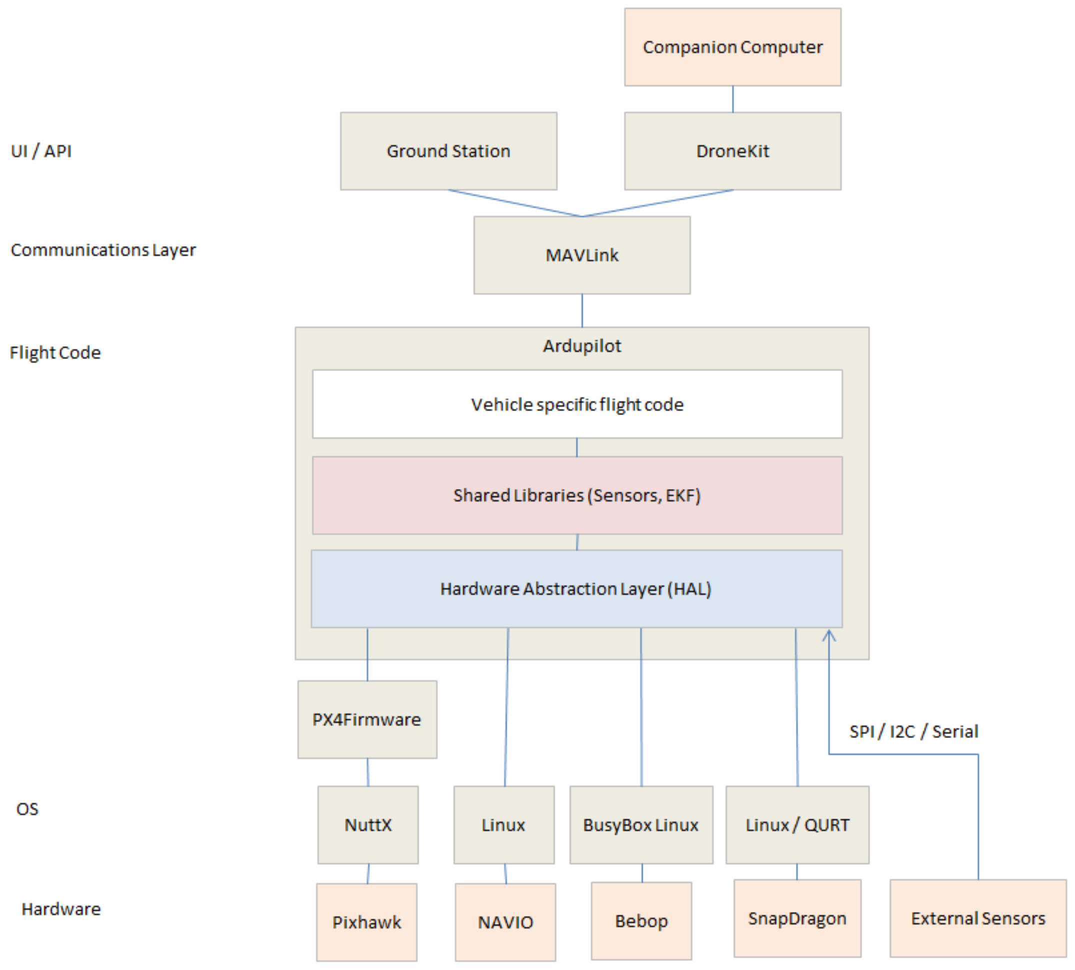
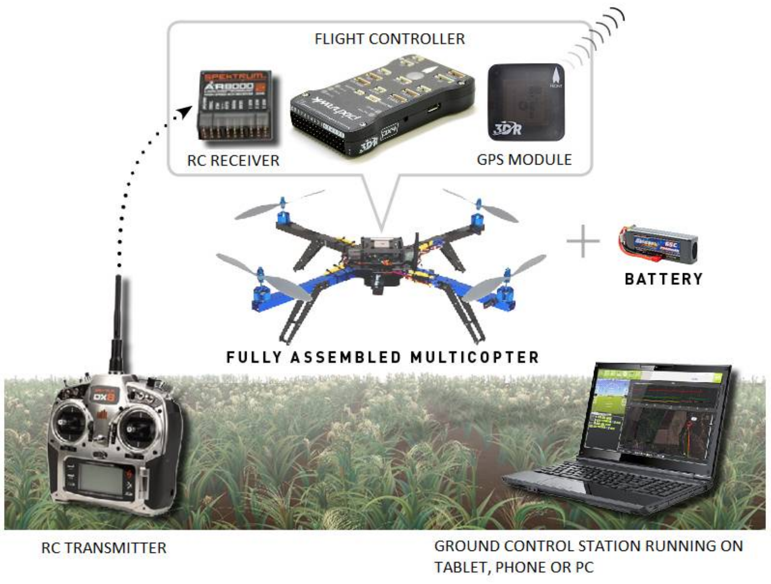


Figure 5.8: ArduPilot High-Level Architecture.

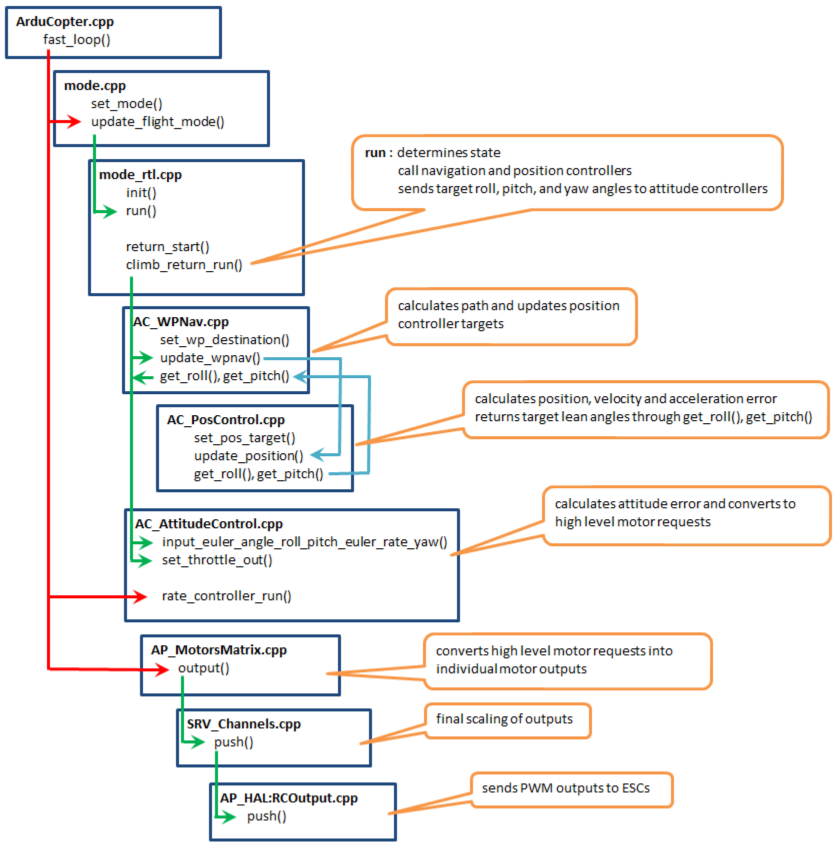
5.3.1.5 Copter

Copter is an advanced open-source autopilot system for multi-copters, helicopters and other rotor vehicles. It provides several flight modes controlled via the radio, mission commands, ground station or a companion computer (Figure 5.9a). Flight modes can be subdivided in two groups, manual and autonomous. The diagram of the Figure 5.9b illustrates the high level code flow related to the return to launch (RTL) autonomous flight mode. It serves just to explain how Ardupilot deals with autonomous mode.



FLIGHT CONTROLLER
RC RECEIVER
GPS MODULE
BATTERY
FULLY ASSEMBLED MULTICOPTER
RC TRANSMITTER
GROUND CONTROL STATION RUNNING ON TABLET, PHONE OR PC

(a) Copter's Diagram



ArduCopter.cpp
fast_loop()
mode.cpp
set_mode()
update_flight_mode()
mode_rtl.cpp
init()
run()
return_start()
climb_return_run()
run : determines state
call navigation and position controllers
sends target roll, pitch, and yaw angles to attitude controllers
AC_WPNav.cpp
set_wp_destination()
update_wpnav()
get_roll(), get_pitch()
calculates path and updates position controller targets
AC_PosControl.cpp
set_pos_target()
update_position()
get_roll(), get_pitch()
calculates position, velocity and acceleration error
returns target lean angles through get_roll(), get_pitch()
AC_AttitudeControl.cpp
input_euler_angle_roll_pitch_euler_rate_yaw()
set_throttle_out()
rate_controller_run()
calculates attitude error and converts to high level motor requests
AP_MotorsMatrix.cpp
output()
converts high level motor requests into individual motor outputs
SRV_Channels.cpp
push()
final scaling of outputs
AP_HAL:RCOutput.cpp
push()
sends PWM outputs to ESCs

(b) Autonomous Flight modes Code Structure (Example of RTL

Figure 5.9: Copter and Autonomous Flight mode Code Structure(Example of RTL) Diagrams.

For every update(100Hz for APM2.x), the function "update_flight_mode()" is called to check the vehicle mode and call the appropriate “run()” function, specific to every flight mode. This latter function converts the user’s inputs into a lean angle, rotation angle, rotation rate, climb rate that is appropriate for the chosen flight mode. The “run()” function thus sends these desired values to the navigation “AC_WPNav”, attitude“AC_AttitudeControl” and/or position “AC_PosControl” control libraries that will be explained below. The next step consists on calling the “rate_Controller_run()” function in the “AC_AttitudeControl” library that converts the previous outputs into roll, pitch and yaw inputs which are sent to the “AP_motors” library. The “AP_motors”library converts values into individual motors outputs(PMW values).

5.3.1.5.1 Attitude Control

The control method for each axis is achieved using a simple P controller for the attitude control in cascade with a PID controller for the angular velocities. The input of each proportional controller is the respective angle error (e_ϕ,e_ψ,e_θ). This controller outputs the desired angular velocity that is the input of the angular velocity controller (PID) which converts the rotation rate error into a high level motor command. Figure 5.10 depicts the control method. Notice that the Proportional controller for the angles control uses a square root relationship. This technique permits to to smooth the response and avoid overshoots. It corrects large angle errors quicker with higher rates without exceeding the maximum acceleration capabilities of the air-frame.

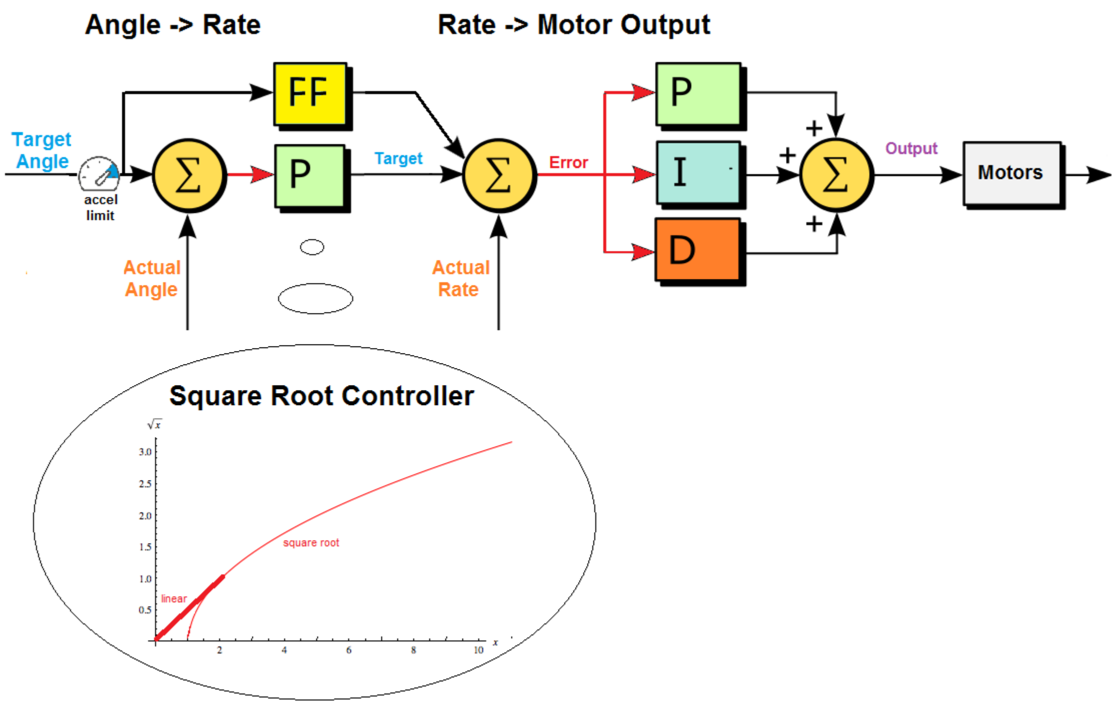
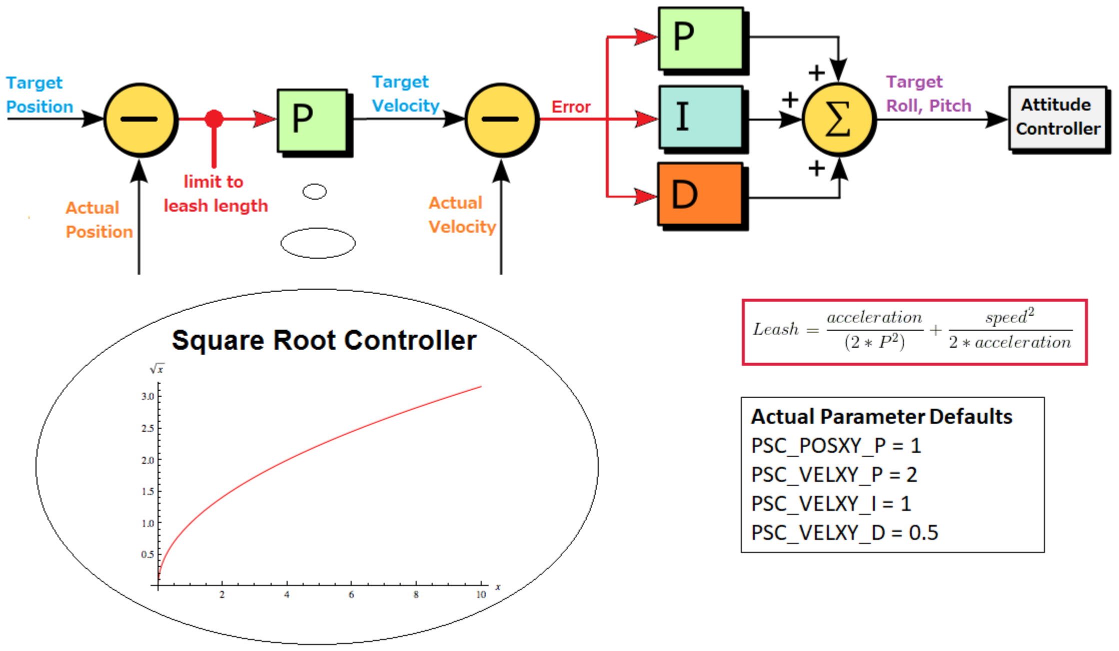


Figure 5.10: ArduCopter Attitude Control.

5.3.1.5.2 Position Control And Navigation

- Position Control

Position control is done via the “AC_PosControl” library. It is composed of a horizontal (X and Y axis) controller and a vertical (Z-axis)controller. Figure 5.11 illustrates the control strategy. As in the attitude controller, the proportional controller P “square root controller” converts position error to a target velocity, then a PID Controller is used to convert velocity error into a desired acceleration that is converted to a desired lean angle. The output of the PID controller is sent to the attitude controller presented above. The Z-axis controller structure is similar except that it uses a proportional controller to convert velocity error to a desired acceleration an acceleration PID is then used to convert acceleration error into a desired throttle that is sent to the attitude control library.



$$Leash = \frac{acceleration}{(2 * P^2)} + \frac{speed^2}{2 * acceleration}$$

Actual Parameter Defaults
PSC_POSXY_P = 1
PSC_VELXY_P = 2
PSC_VELXY_I = 1
PSC_VELXY_D = 0.5

Figure 5.11: Arducopter Position Control.

- Navigation

  Navigation in Arducopter is achieved using the “AC_WPNav” library. This library implements three independent controllers. Waypoint that flies the vehicle in a straight line to defined targeted waypoints. Spline that flies the vehicle in a smooth curved path to a defined targeted waypoint with a final velocity in order to continue smoothly towards a following waypoint. The last controller is the brake controller that tries to slow the vehicle to a stop as quickly as possible. Note that these controllers use the Position controller presented above for positions and velocities controls.

5.3.1.5.3 Data Fusion and Pose Estimation: Copter provides three versions of extended Kalman filters (EKF, EKF2, EKF3) implemented in the AP_NavEKF, AP_NavEKF2 and AP_NavEKF3 libraries. By fusing all the available measurements from rate gyroscopes, accelerometer, compass, GPS, airspeed and barometer, it is capable to accurately estimate the position, velocity and angular orientation of the aerial vehicle by rejecting measurements with significant errors.

- Data fusion

  As explored in the third subsection of the fourth chapter, the extended kalman filtering algorithm consists on two phases to fuse and estimate states. Ardupilot Algorithm does it in the following way:

  - State prediction:

    This phase uses the Inertial Measurement unit(IMU). The Angular rates are integrated to obtain the angular position. Accelerations are integrated to get the velocity that is integrated to calculate the position. Estimated gyro and accelerometer noise are used to estimate the growth in error in the angles, velocities and position calculated using IMU data. Making these parameters larger causes the filters error estimate to grow faster. If no corrections are made using other measurements (eg GPS), this error estimate will continue to grow. These estimated errors are captured in a large matrix called the 'State Covariance Matrix'.

  - State Update:

    The state prediction phase is repeated for every IMU's data received. The update phase consists on adjusting these predictions to obtain a more accurate estimate of the position, velocity and angular orientation. It is done by fusing data from other sensors (GPS,compass,airspeed,barometer...)
    . In the case of a GPS,when a measurement is available, the filter calculates the difference (innovation) between the previous predicted position and the one given by the GPS. The Innovation, State Covariance Matrix and the GPS measurement error 'EKF_POSNE_NOISE' are combined to calculate a correction to each of the filter states. The amount of correction is controlled by the assumed ratio of the error in the states to the error in the measurements. This means that if the filter thinks its own calculated position is more accurate than the GPS measurement, then the correction from the GPS measurement will be smaller. If it thinks its own calculated

position is less accurate than the GPS measurement, then the correction from the GPS measurement will be larger. The assumed accuracy of the GPS measurement is controlled by the EKF_POSNE_NOISE, parameter. Making EKF_POSNE_NOISE larger causes the filter to think the GPS position is less accurate. After this step, the amount of uncertainty in each of the states that have been updated is reduced. The filter calculates the reduction in uncertainty due to the ‘State Correction’, updates the ‘State Covariance Matrix’ and return to the prediction phase.

5.3.2 Simulation and results

This part presents the integration and results of the different concepts described in the previous subsection Figure (5.12).

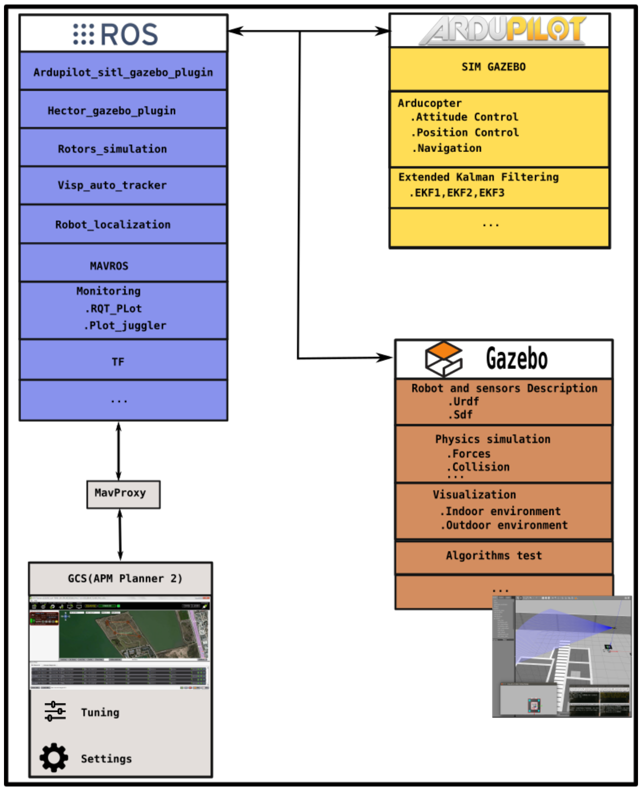
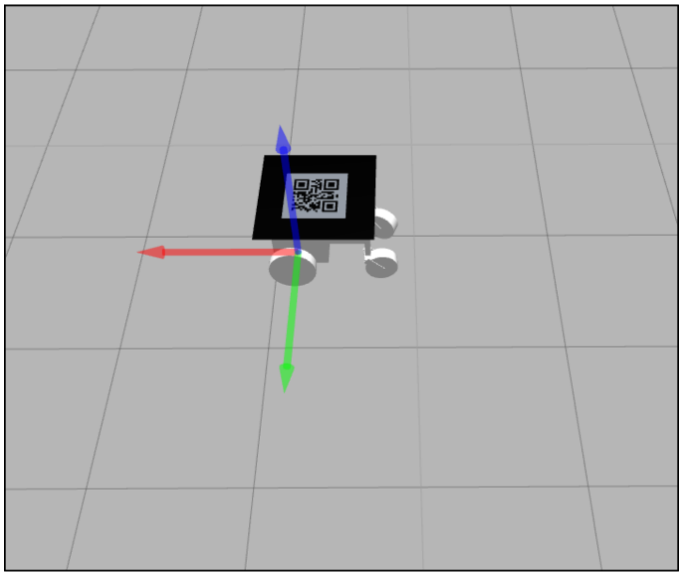


Figure 5.12: Gazebo + ROS + Ardupilot.

5.3.2.1 Gazebo

We used the simulation environment provided by Erle Robotics [23]. It already integrates a quadrotor vehicle (Figure 5.13b) along with GPS, IMU, cameras, lidar. Using the URDF (Universal Robotic Description Format) system file, we added to this environment the mobile robot along with a QR-code (Figure 5.13a). URDF is the standard ROS XML representation of the robot model (kinematics, dynamics, sensors). Figure 5.14 shows the joints and links of the two vehicles.


(a) Mobile Robot

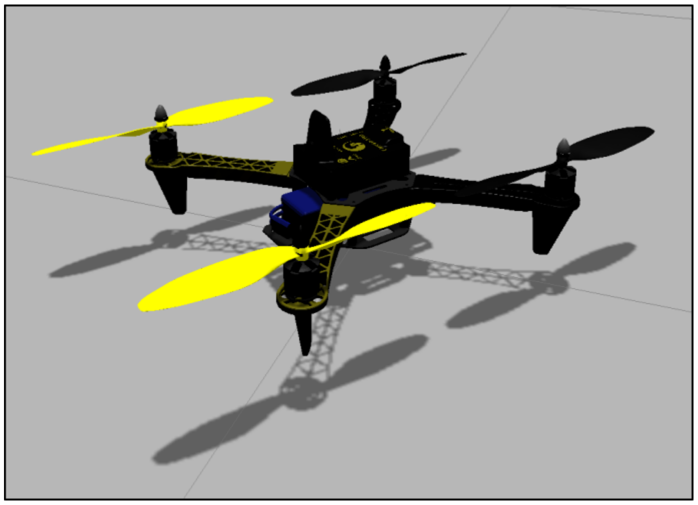
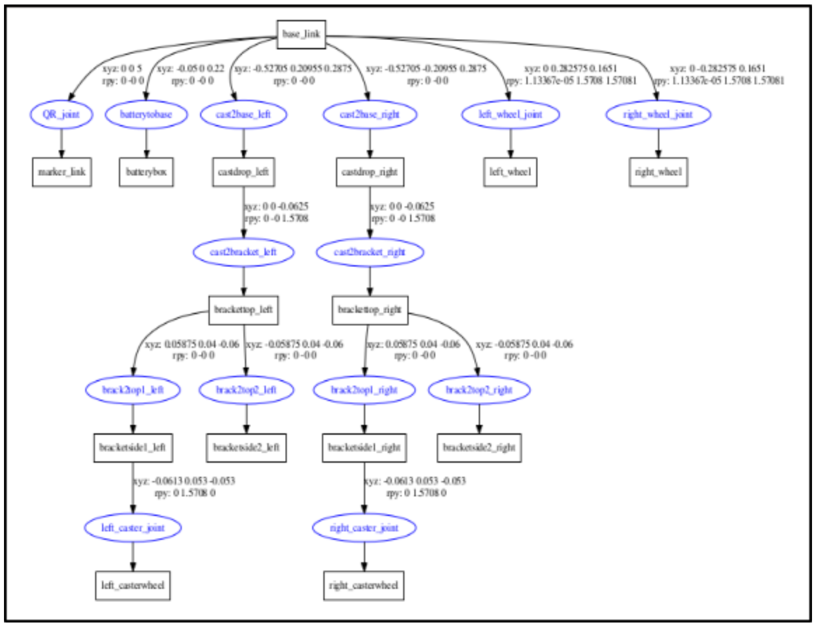
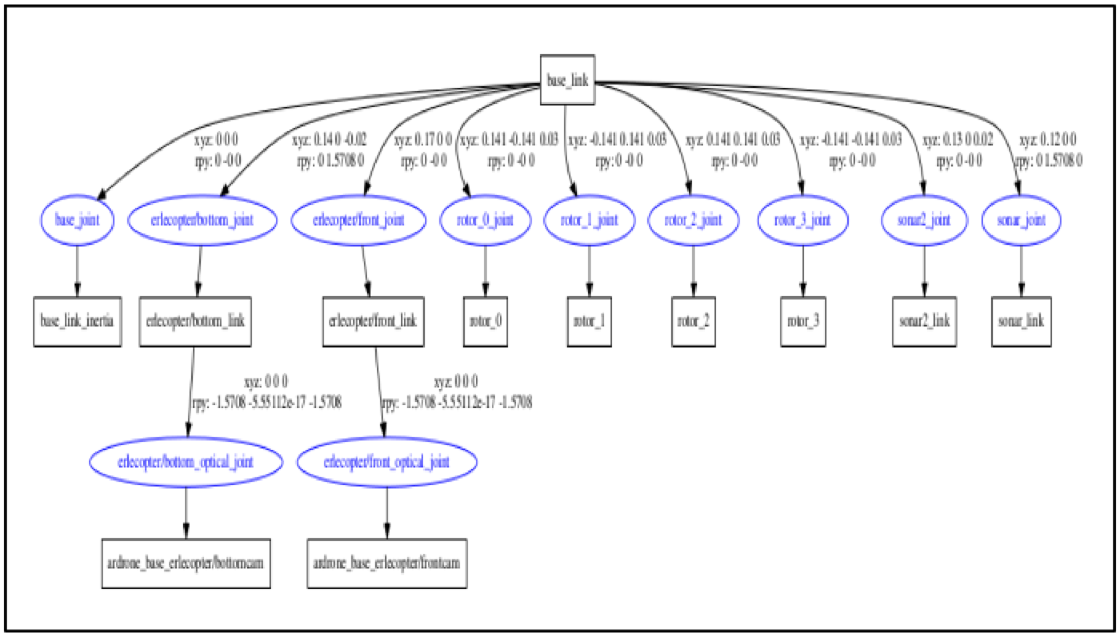

(b) ErleCopter

Figure 5.13: Quadrotor and Mobile Robot in Gazebo.



(a) Mobile Robot URDF



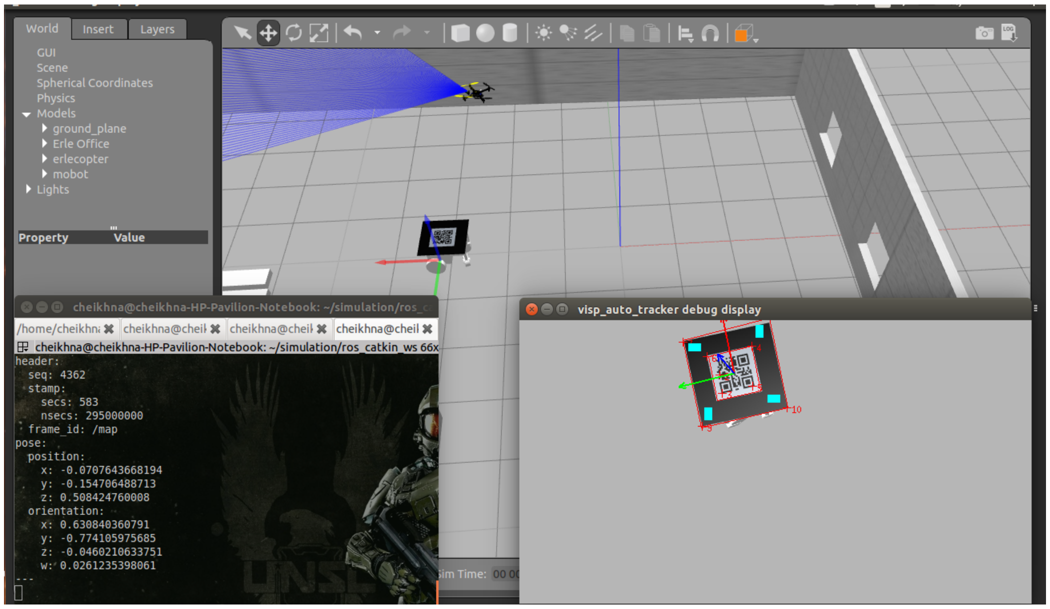
(b) Quadrotor URDF

Figure 5.14: Quadrotor and Mobile Robot URDF Tree.

5.3.2.2 ROS Packages

5.3.2.2.1 Visp_Auto_Tracker for Target Detection

Visp_Auto_tracker is an online automated pattern-based object tracker based on visual servoing. It wraps model-based trackers provided by ViSP (visual servoing library) into a ROS package. This computer vision algorithm computes the pattern position and orientation with respect to the camera. Its great assets are that it is fast enough to allow objects online tracking using a camera and re-initialize the tracker when lost of tracking occur by achieving a new detection procedure.



```
header:
  seq: 4362
  stamp:
    secs: 583
    nsecs: 295000000
  frame_id: /map
pose:
  position:
    x: -0.0707643668194
    y: -0.154706488713
    z: 0.508424760008
  orientation:
    x: 0.630840360791
    y: -0.774105975685
    z: -0.0460210633751
    w: 0.0261235398061
---
```

Figure 5.15: Target Detection.

5.3.2.2.2 TF for Target Localization

Tf is a package that permits to keep track of multiple coordinate frames over time. A frame is an origin in coordinate space that describes the pose of an object. Tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, and more, between any two coordinate frames at any desired point in time. This package was primarily used to define the robots components location, IMU,GPS,Camera for the quadrotor and wheels for the car-like robot frames by specifying their respective positions and orientation relative to the robots' base_link_frame. The second use of this package was to achieved the Localization phase of the ground robot. The following lines explain the frames used by the estimation node.

- base_link frame:

This frame is is rigidly affixed to the robot and provides a point of reference.

- Odom frame:

  This is a world-fixed frames which origin is typically aligned with the robot's starting position. This frame is computed using the an odometry sources. The pose in this frame is guaranteed to be continuous, yet it is subject to drifts over time making it not accurate for long-term reference.

- map frame:

  This is a a world fixed frame, with its Z-axis pointing upwards. The pose of a robot in this frame is permanently re-computed by a localization component based on sensors observations(GPS). This re-computation permits to eliminate drift, yet it causes discrete jumps when new sensors information arrives. This non-continuous frame is used for long-term global reference.

- Earth frame:

  The earth frame is the origin of the ECEF (earth-centered, earth-fixed) that is a conventional terrestrial system. This frame is designed to allow the interaction of multiple robots in different maps by providing a common reference for their respective maps. The following figure illustrates the relationship between frames.

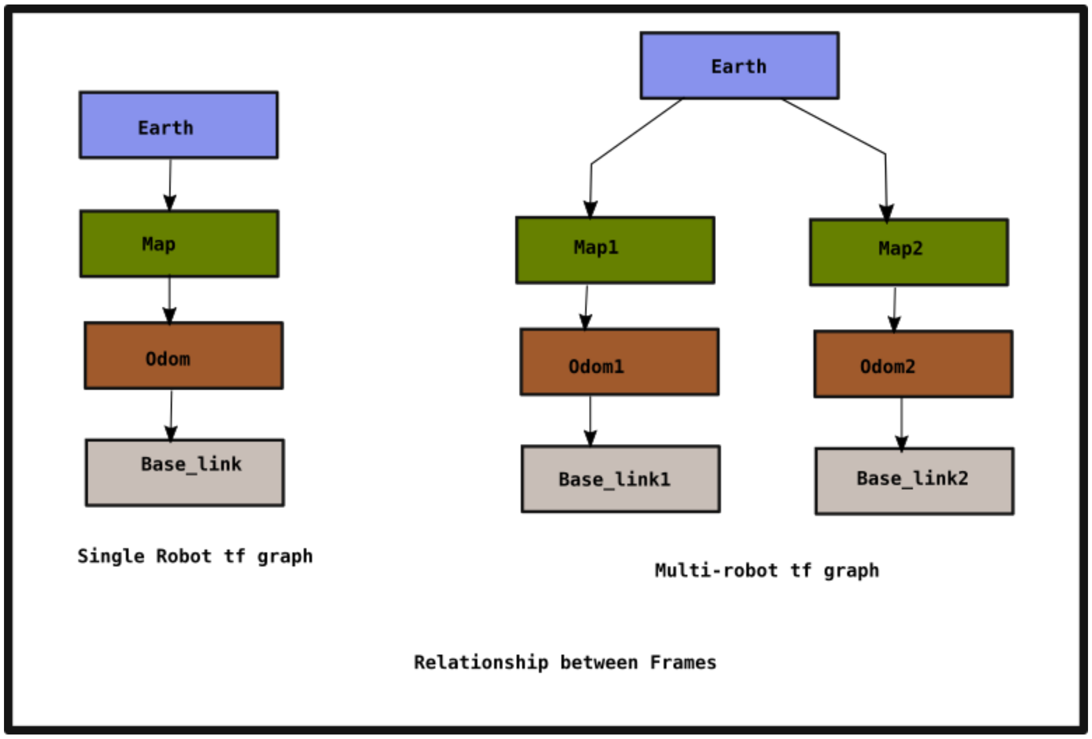


Figure 5.16: Transforms tree.

The earth frame was not used in this project because of the non-cooperative constraint of the agents. This has been solved by using the Tf capabilities to transform the pose obtained in the camera-frame into the quadrotor map-frame.

5.3.2.2.3 Robot_Localization for Poses Estimation and Ground Robot Tracking

This package serves as the core of our algorithm. It is a collection of state estimation nodes. It contains two nonlinear state estimation nodes, implementing an extended (ekf_localization_node) and unscented Kalman filter (ukf_localization_node) algorithm that tracks the 15-dimensional states of the vehicle (X,Y,Z,ϕ,θ,ψ,$\dot{X}$,$\dot{Y}$,$\dot{Z}$,$\dot{\phi}$,$\dot{\theta}$,$\dot{\psi}$,$\ddot{X}$,$\ddot{Y}$,$\ddot{Z}$). In addition, Robot_localization provides a node (navsat_transform_node) for the GPS integration. We used the ekf node to fuse the IMU, odometry and GPS data to estimate the quadrotor states. We also took advantage of the continuous estimation feature of the package. In fact, the estimation starts as soon as a single measurement is received and continues (using an internal motion model) even if no data is arrives for a long period of time. The pose estimation and tracking of the ground robot are obtained respectively using in a first time the localization process coupled with the tracking control method described in Section 5.2.3. Figure 5.17 shows the transform tree generated to achieve the poses estimation process. Figures 5.18 and 5.19 show respectively the poses estimation results at the take-off and near hover states. These results prove that the filter successfully converges to the actual desired values represented by the topics '_gazebo/model_states'. Figure 5.20 illustrates the tracking results. The localization and tracking result along the X-axis is illustrated by the Figure 5.20(a) with the blue curve representing the mobile robot trajectory along this axis. The red curve shows the successful tracking process of the quadrotor with the big jumps corresponding to the estimation process when loss of tracking occurs. Figure 5.20(b) represents the tracking process along the Y-axis. To guarantee smooth tracking a threshold was considered to start the tracking along the Y-axis since the mobile robot cannot directly move sideways. The big Jumps correspond to the estimation during loss of tracking.

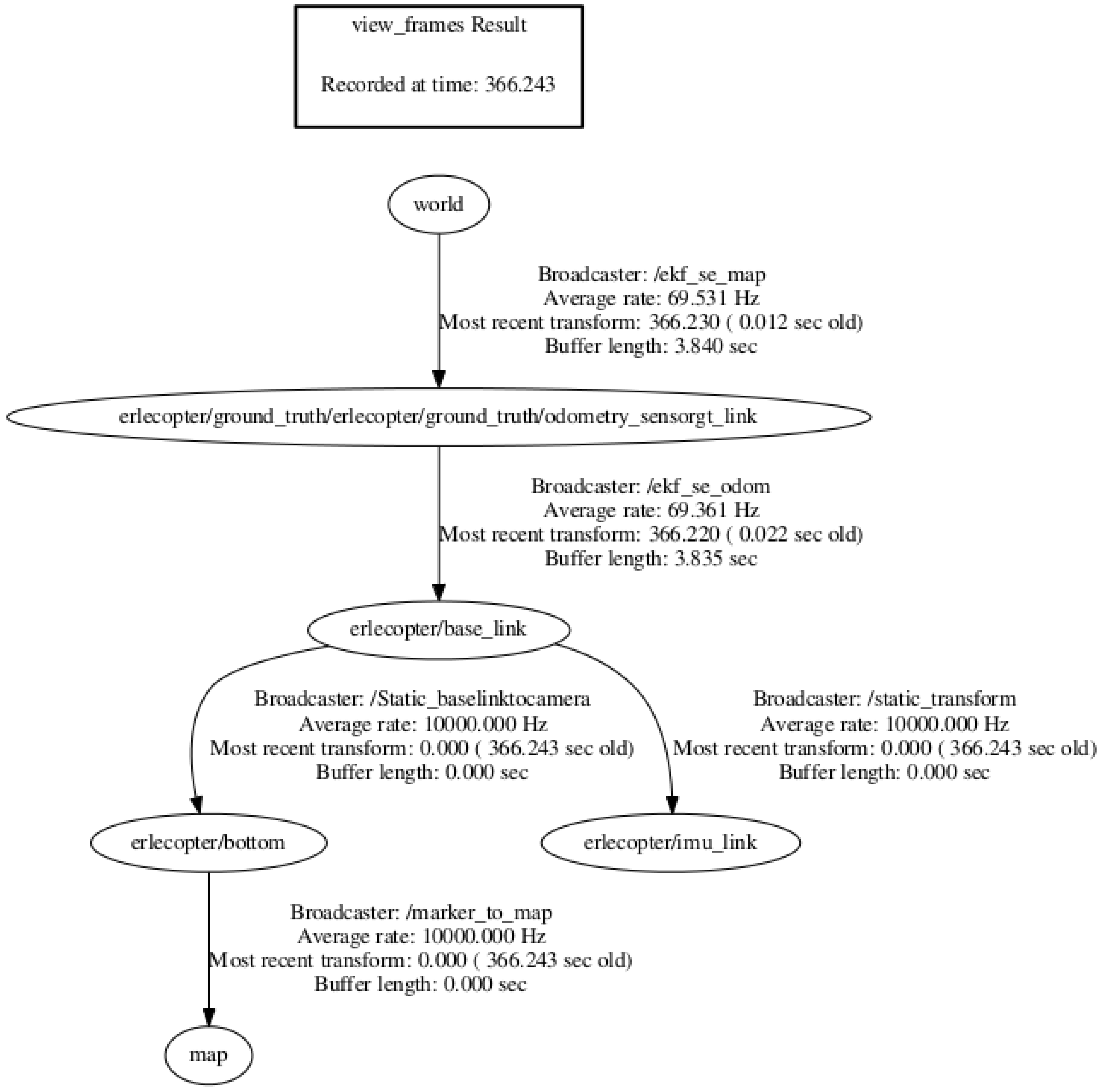

view_frames Result
Recorded at time: 366.243
world
Broadcaster: /ekf_se_map
Average rate: 69.531 Hz
Most recent transform: 366.230 (0.012 sec old)
Buffer length: 3.840 sec
erlecopter/ground_truth/erlecopter/ground_truth/odometry_sensorgt_link
Broadcaster: /ekf_se_odom
Average rate: 69.361 Hz
Most recent transform: 366.220 (0.022 sec old)
Buffer length: 3.835 sec
erlecopter/base_link
Broadcaster: /Static_baselinktocamera
Average rate: 10000.000 Hz
Most recent transform: 0.000 (366.243 sec old)
Buffer length: 0.000 sec
Broadcaster: /static_transform
Average rate: 10000.000 Hz
Most recent transform: 0.000 (366.243 sec old)
Buffer length: 0.000 sec
erlecopter/bottom
erlecopter/imu_link
Broadcaster: /marker_to_map
Average rate: 10000.000 Hz
Most recent transform: 0.000 (366.243 sec old)
Buffer length: 0.000 sec
map

Figure 5.17: Robot_localization TF.

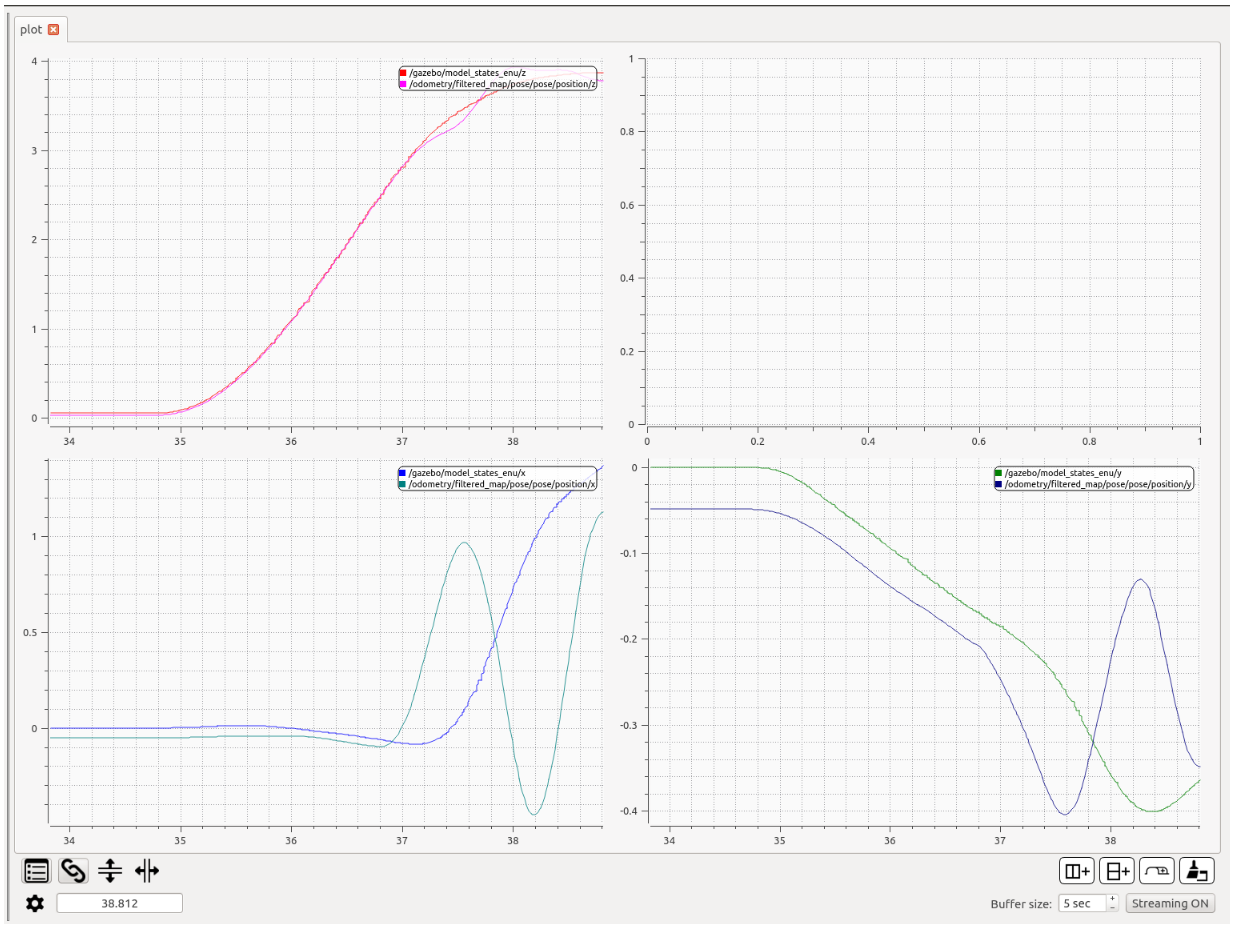
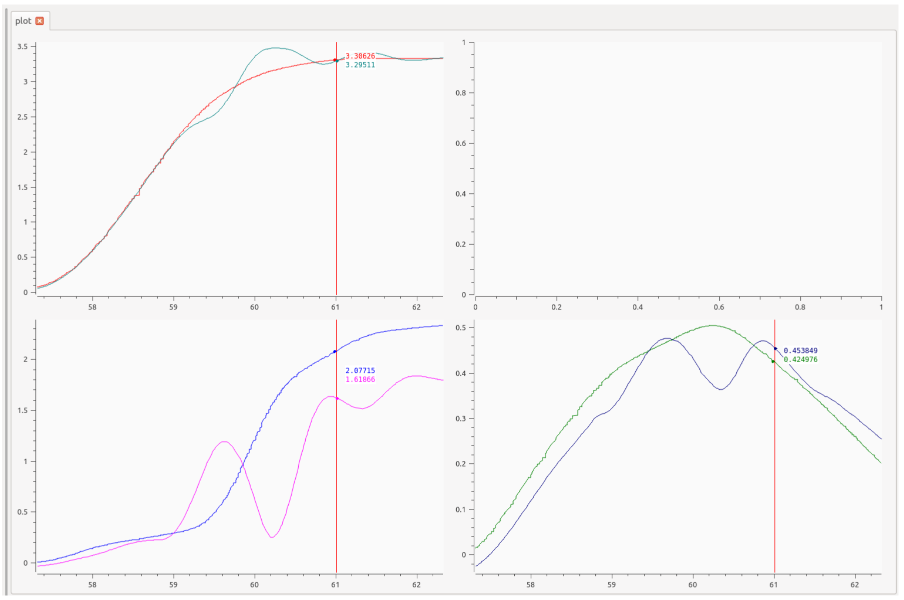
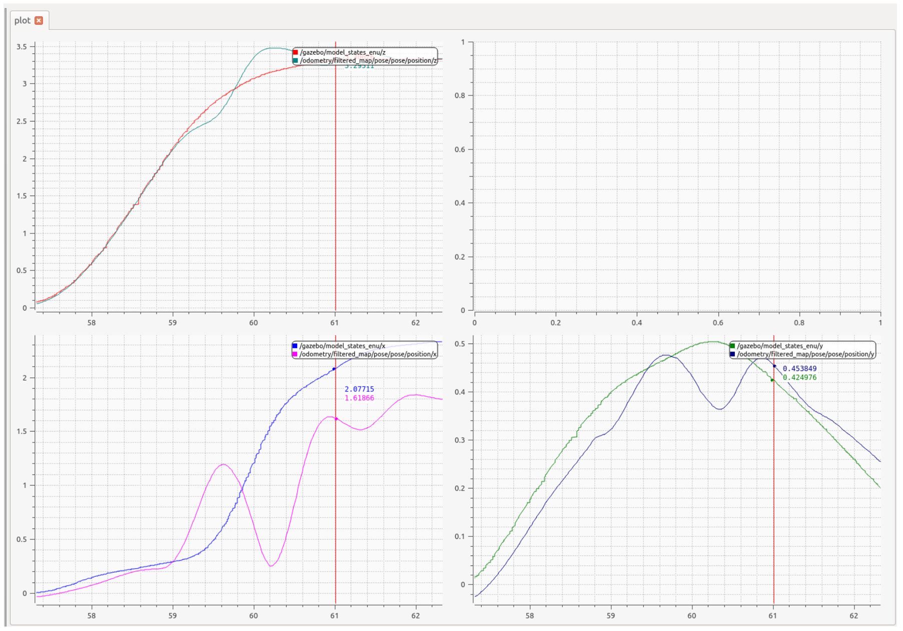


Figure 5.18: Quadrotor Pose Estimation at Take-Off.

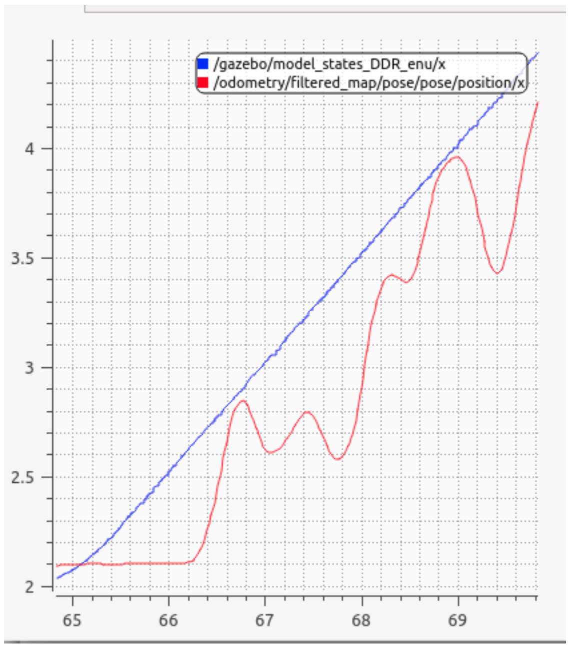


(a) Quadrotor Pose Estimation (With Errors)

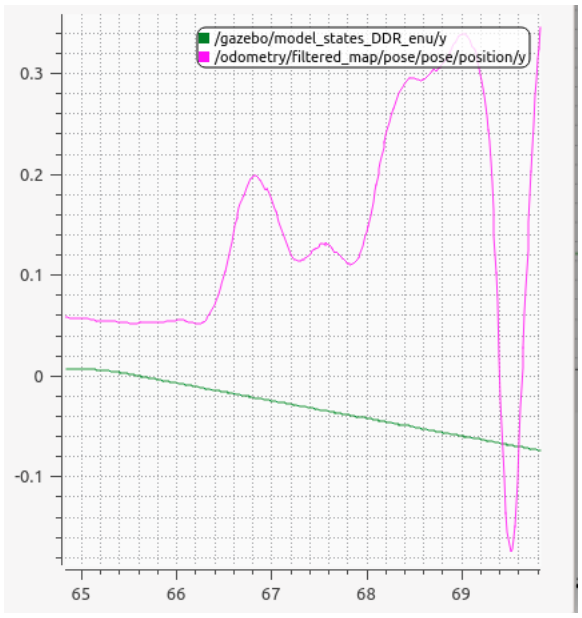


(b) Quadrotor Pose Estimation (With Topic Names)

Figure 5.19: Quadrotor Pose estimation at Hover.



(a) Tracking along X-axis



(b) Trackin aling Y-axis

Figure 5.20: Mobile Robot Localization and Tracking.

5.3.2.2.4 Mavros

Mavlink is the communication protocol between a ground station application and an autopilot board. Mavros provides communication driver for various autopilots with MAVLink communication protocol. It permits to integrate Mavlink messages into ROS and was used to send Mavlink commands. Table 6.1 summarizes the topics used in simulation. Figure 5.21 depicts the Computation graph level containing all the nodes along with the different topics. The subscriber node “/poursuite” gathers information from the detection node “/visp_auto_tracker”, the transform node “/tf”, the EKF nodes “/ekf_se_map” and “ekf_se_odom”, the “/mavros” node to achieve the mission and outputs commands via mavros. The other nodes such as the “gazebo” node where used to monitor and compare the estimated poses with the ones produced by the simulator.

ROS Topics and Their Purpose	
Topics	Purpose
/visp_auto_tracker/object_position	Returns the 3D Pose of the QR_code.
/erlecopter/gps/fix	GPS data.
erlecopter/ground_truth/odometry	Odometry data.
/erlecopter/imu	Imu data, orientation computed by FCU.
/odometry/filtered/map	Quadrotor Estimated States in the World Frame.
/mavros/setpoint_position/local	Local frame setpoint position.

Table 5.1: ROS Topics used in Simulation.

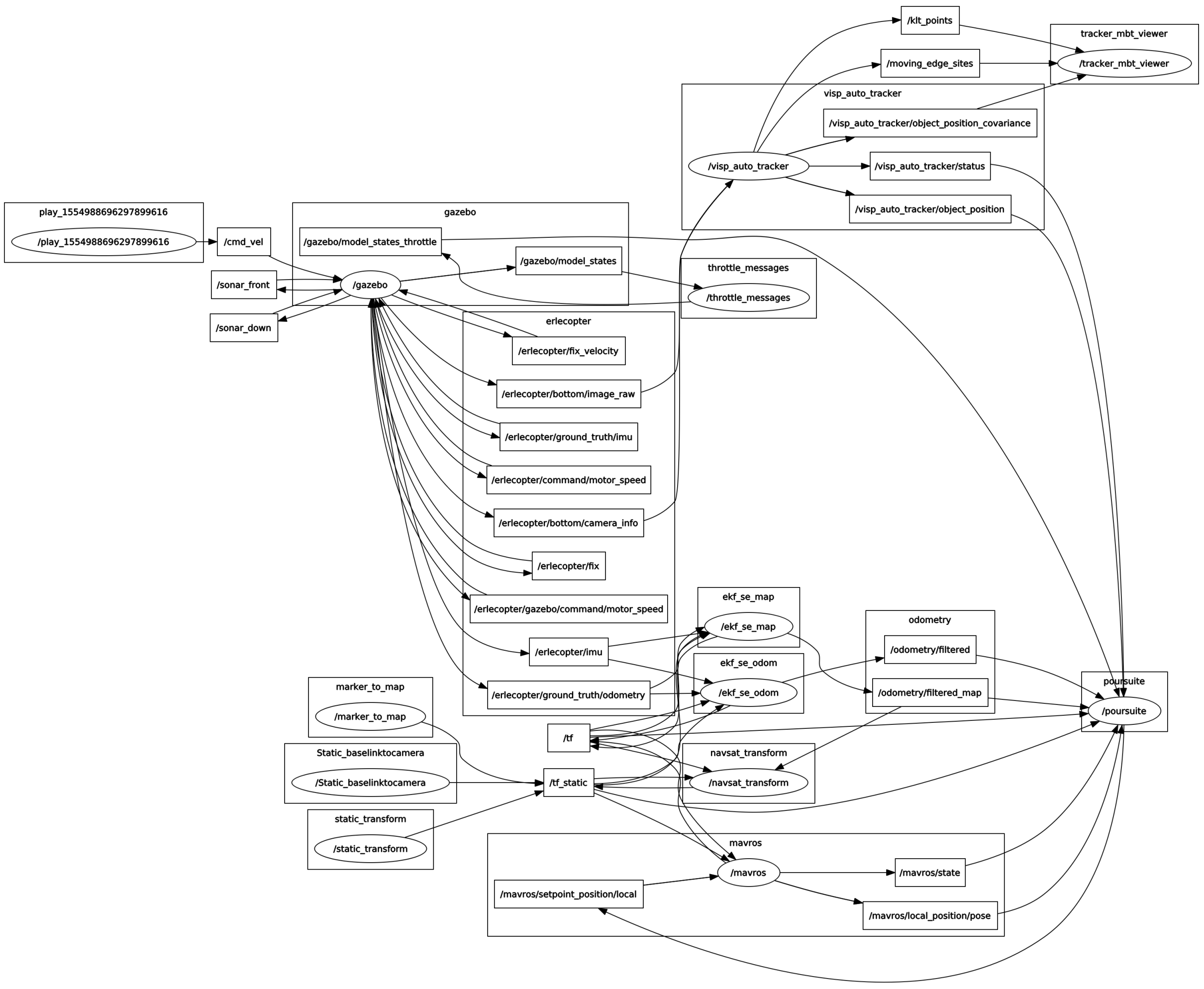
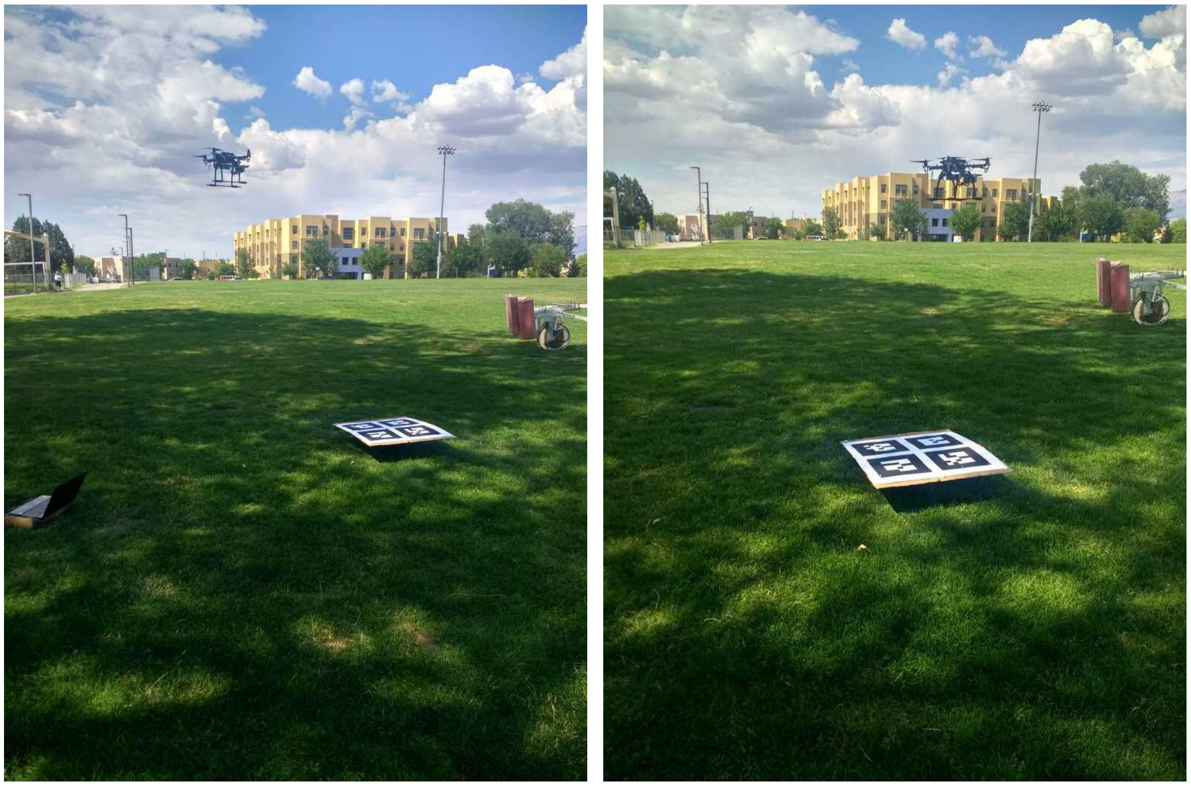


Figure 5.21: ROS_graph.

5.4 Physical Implementation

This part aims to complete all the previous work and prove the successful implementation of both the preceding theories and simulations. It presents all the hardware and software used for the real implementation. The target detection package visp_auto_tracker worked well in simulation, but showed some limitations and instabilities due to the outdoor environment lightning in addition to the quadrotor vibrations. The package ar_track_alvar [24] showed better results and was used for the detection and tracking processes in order to achieve the mission.



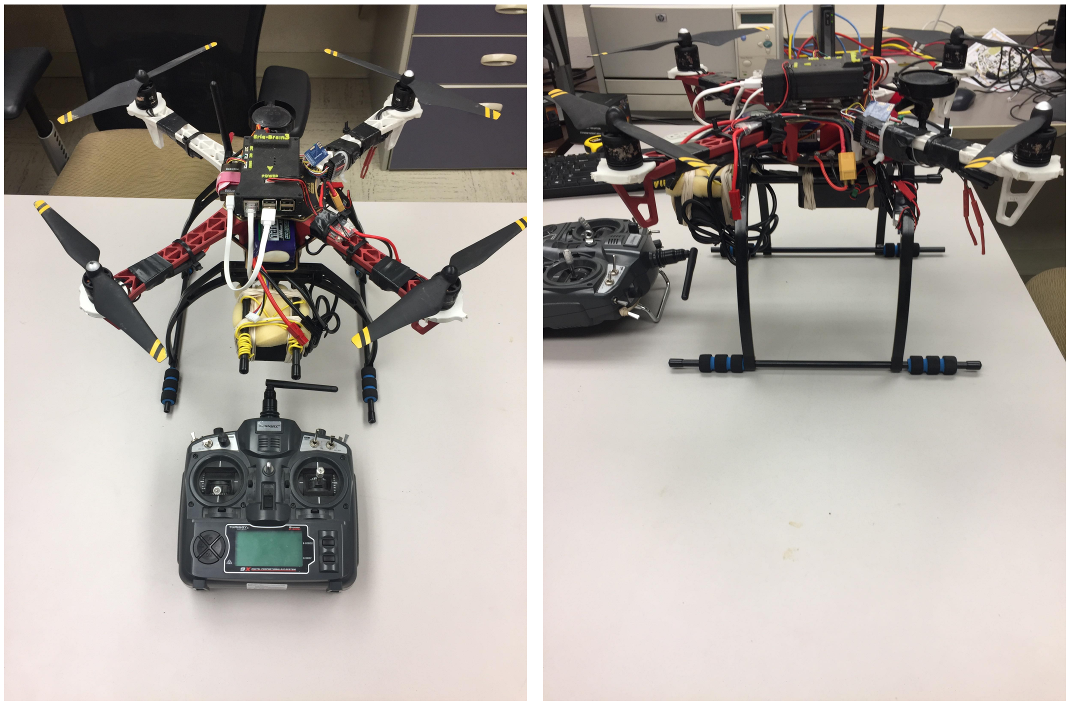
(a) Detection (b) Tracking

Figure 5.22: Real Implementation.

5.4.1 Hardware

5.4.1.1 Quadrotor

Figure 5.23 shows the low-cost quadrotor, built using the components described in Table 5.2. The autopilot is represented by the Erlebrain 3, a Linux based artificial robotic brain [25]. Notice that the Erle-brain 3 includes a companion computer with ROS kinetic already installed, yet because of difficulties related to the installation of some packages, a raspberry pi 3 model B with also ROS kinetic installed, was used as a companion computer. These two computers were operating in the same network using a crossover Ethernet cable (Figure 5.24), in order to take advantage of the distributed computing capacity of ROS.



(a) Quadrotor (Front) (b) Quadrotor (side)

Figure 5.23: Quadrotor Vehicle.

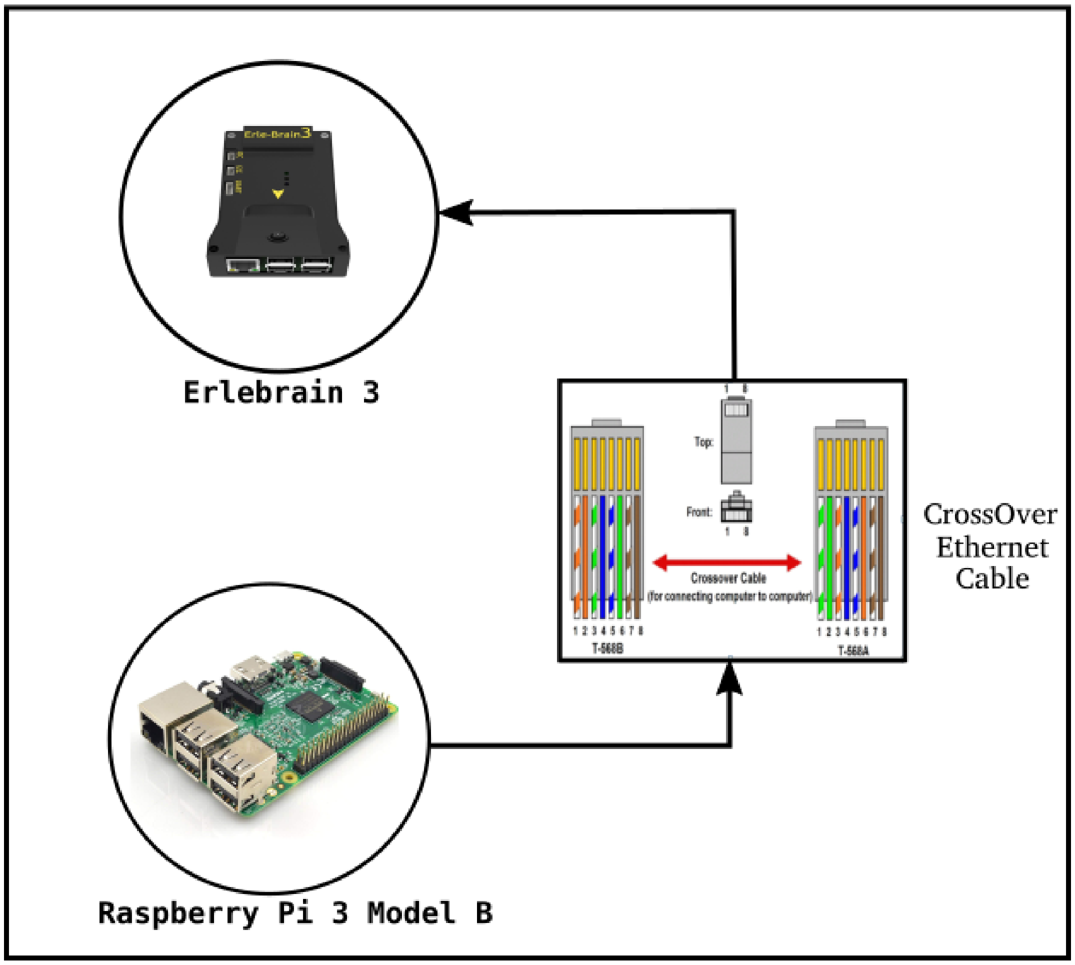


Figure 5.24: computers connection.

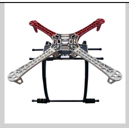
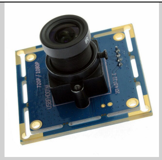
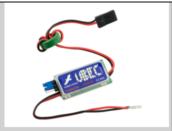
Quadrotor Components		
Parts	Description	image
Frame + Landing gears	DJI flamewheel 450 (282g,Diagonal wheelbase 450mm)	
ESCs	DJI 15A ESC for E300 Tuned Propulsion Power System 15 Amp Opto	
Propellers	DJI 9450 Props Carbon Fiber Reinforced	
Motors	DJI E300 Brushless 2212 / 920KV	
Quadrotor Battery	Turnigy nano-tech 3300mAh 4S 35 70C Lipo Pack w/XT-60	
Erlebrain 3	Linux-based artificial robotic brain	
Raspberry pi 3 Model B	Companion Computer	
GPS with compass	GPS uBlox Neo-M8N and digital compass	
Telemetry	3DR Radio 915MHZ Telemetry Kit	
Camera	2megapixel(1080P) usb camera (ELP-USBFHD01M series)	
Transmitter and receiver	Turnigy 9X 9Ch Transmitter and iA8 Receiver	
Ubec for raspberry pi powering	Hobbywing 5V 3A UBEC Step-Down Converter	

Table 5.2: Quadrotor parts and specifications.

5.4.1.2 Ground Vehicle

The ground vehicle consists of a remote controlled car-like robot with the AR tag attached on top (Figure 5.25).

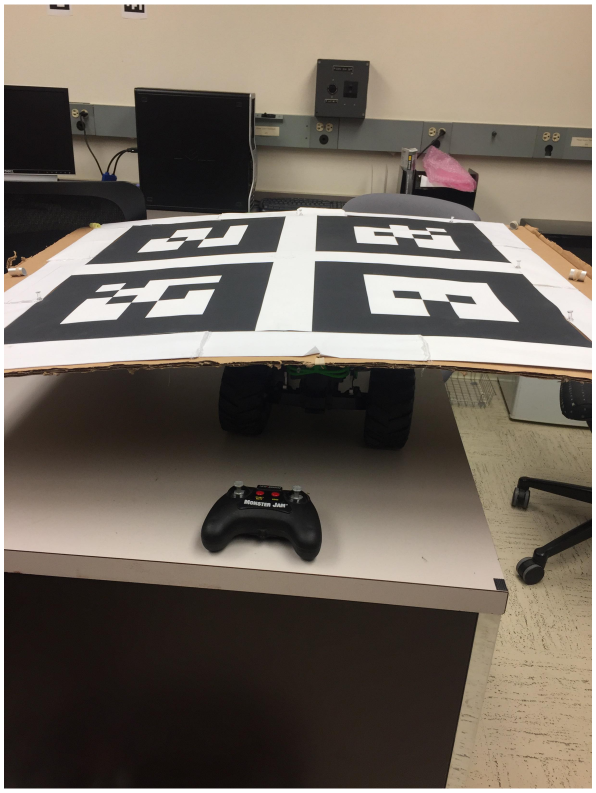


Figure 5.25: Car_like remotely controlled Robot.

5.4.1.3 Ar tag

ALVAR [26] is an open source AR tag tracking library for virtual and augmented reality. This library allows to create accurate, efficient and robust AR applications for markers, multi markers 2D images, and 3D point cloud-based tracking. Ar_track_alvar is a ROS wrapper for ALVAR that includes capabilities such as tags generation, individual tags detection and multi-tag bundles detection. We used The multi-tag bundles detection capabilities to guarantee robustness of the detection and identification process.The bundles were treated as a single unit. This configuration permitted to increase the robustness of the target detection. In fact, even if the master tag (first tag in the xml file, produced after the tags generation) is lost during the flight, any other detected tag guarantee a good estimation of the master pose. Figure 5.26 shows the generated tags treated as a unit.

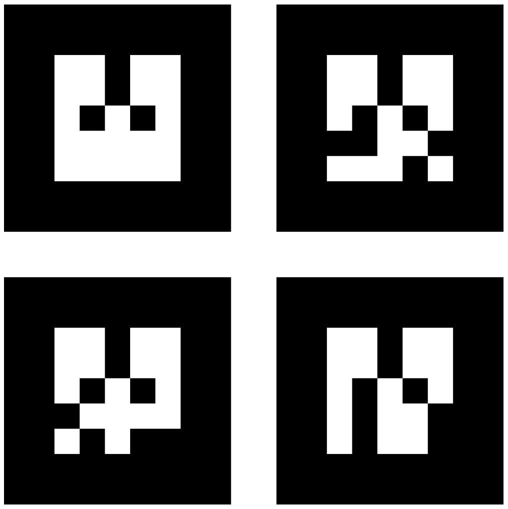


Figure 5.26: Ar tag Bundle.

5.4.2 Software

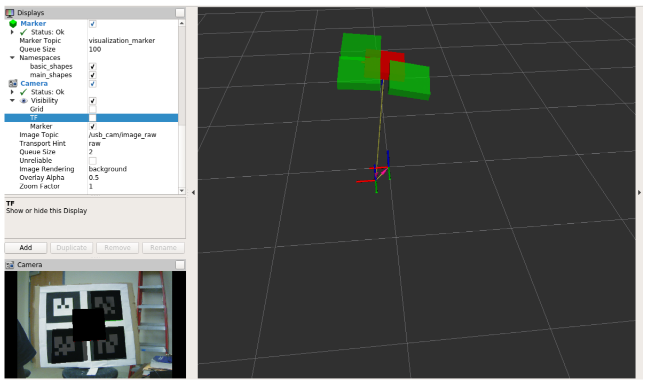
This part presents the organization of the different topics and nodes running across our two computers. All the packages were installed on the companion computer. The image processing, the quadrotor and target poses estimation in addition to the tracking nodes ran on the companion computer. The obtained data were used by the Erlebrain via the tracking node (“poursuite”) node (Figure 5.21) to achieve the mission. Table 5.3 catalogues the topics used in the mission.

ROS Topics and Their Purpose	
Topics	Purpose
/ar_pose_position	Returns the 3D Pose of the AR tag.
/erleGPS	GPS Fix.
/erleOdometry	Odometry message.
/erleIMU	Imu data, orientation computed by FCU.
/odometry/filtered_map	Quadrotor Estimated States in the World Frame.
/mavros/setpoint_position/local	Local frame setpoint position.

Table 5.3: ROS Topics used in Real Implementation.

5.4.2.1 Detection, Localization and Poses estimation

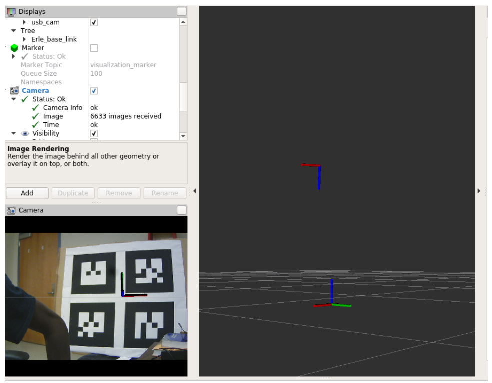
The detection was achieved thanks to four generated AR_tags of 33.02×33.02cm. These four tags were treated as a single unit with the master marker being the top left one. Some modifications were done to the generated xml file to position (during the detection) this marker at the center of the tags (image view of Figure 5.26a) in order to have the global frame of all the tags at the center of the bundle (image view of Figure 5.6b). In addition, Figure 5.26a shows the detected tags with the master tag in red, meanwhile Figure 5.26b shows the relation between the camera and marker frame with the camera as fixed frame (frame on the ground-plane).



```
Displays
Marker                ✓
  Status: Ok
  Marker Topic        visualization_marker
  Queue Size          100
  Namespaces
    basic_shapes      ✓
    main_shapes       ✓
Camera                ✓
  Status: Ok
  Visibility          ✓
    Grid              ☐
    TF                ☐
    Marker            ✓
  Image Topic         /usb_cam/image_raw
  Transport Hint      raw
  Queue Size          2
  Unreliable          ☐
  Image Rendering     background
  Overlay Alpha       0.5
  Zoom Factor         1

TF
Show or hide this Display
```

(a) Marker Visualization Rviz



```
Displays
    usb_cam           ✓
  Tree
    Erle_base_link
Marker                ☐
  Status: Ok
  Marker Topic        visualization_marker
  Queue Size          100
  Namespaces
Camera                ✓
  Status: Ok
    Camera Info       ok
    Image             6633 images received
    Time              ok
  Visibility          ✓

Image Rendering
Render the image behind all other geometry or
overlay it on top, or both.
```

(b) Frames Visualization Rviz

Figure 5.27: Rviz.

The localization step was done using TF as discussed in 5.3.2.2. The TF tree is illustrated by the Figure 5.28.

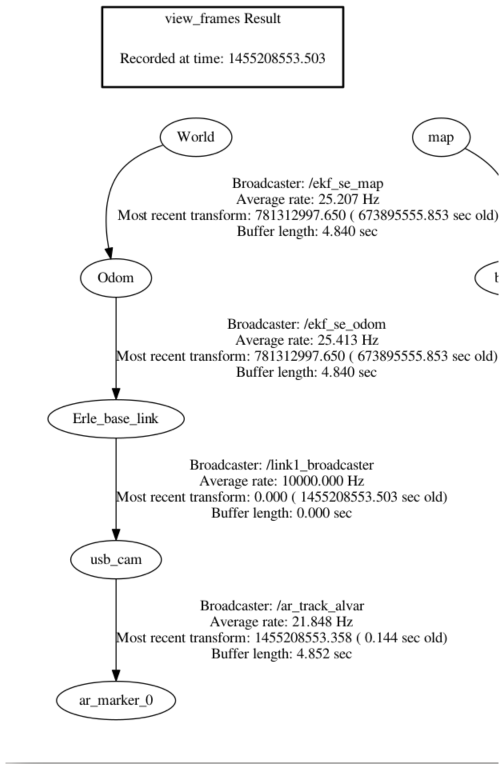
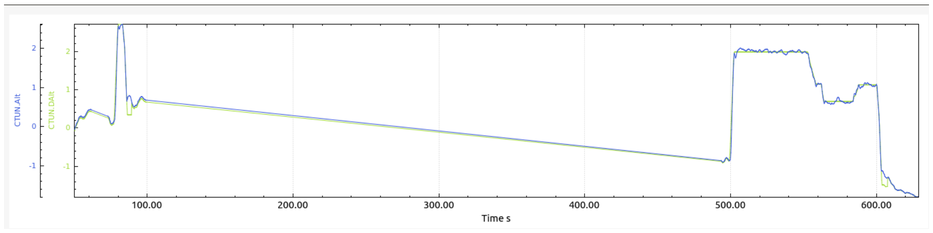

view_frames Result
Recorded at time: 1455208553.503
World
map
Broadcaster: /ekf_se_map
Average rate: 25.207 Hz
Most recent transform: 781312997.650 (673895555.853 sec old)
Buffer length: 4.840 sec
Odom
Broadcaster: /ekf_se_odom
Average rate: 25.413 Hz
Most recent transform: 781312997.650 (673895555.853 sec old)
Buffer length: 4.840 sec
Erle_base_link
Broadcaster: /link1_broadcaster
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1455208553.503 sec old)
Buffer length: 0.000 sec
usb_cam
Broadcaster: /ar_track_alvar
Average rate: 21.848 Hz
Most recent transform: 1455208553.358 (0.144 sec old)
Buffer length: 4.852 sec
ar_marker_0

Figure 5.28: Tf tree.

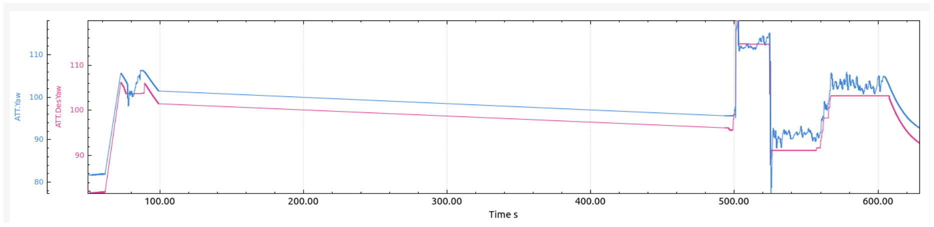
5.4.3 Real Implementation results

5.4.3.1 Quadrotor Altitude and Attitude behavior

This part, presents the data obtained from the real quadrotor. Figures 5.29 and 5.30 represent the real quadrotor data (after manual tuning) from the ground-station (APM planner 2) Data Log files. They respectively illustrate the Altitude, roll, pitch and yaw along with their desired values (CTUN.DALT, ATT.DesRoll, ATT.DesPitch, ATT.DesYaw). These data show a pretty stable vehicle. Notice that better results can be obtained using the auto-tune capabilities of APM of manually fine tuning the different axis of the vehicle.

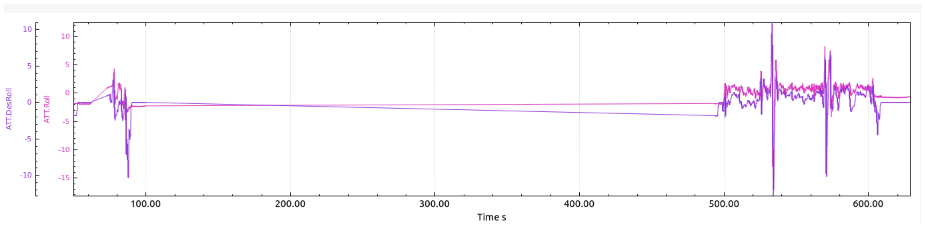


(a) Actual Altitude (Blue) vs Desired Altitude (Green)

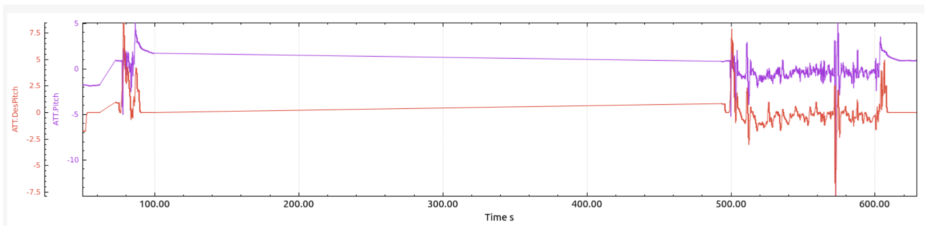


(b) Actual Yaw (Blue) vs Desired Roll (Red)

Figure 5.29: Altitude and Yaw.



(a) Actual Roll (Pink) vs Desired Roll (Purple)

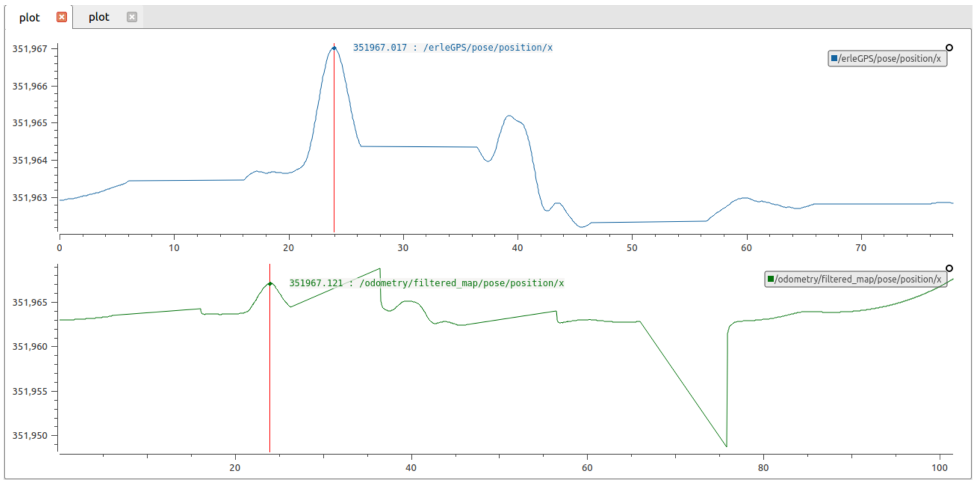


(b) Actual Pitch (Purple) vs Desired Pitch (Red)

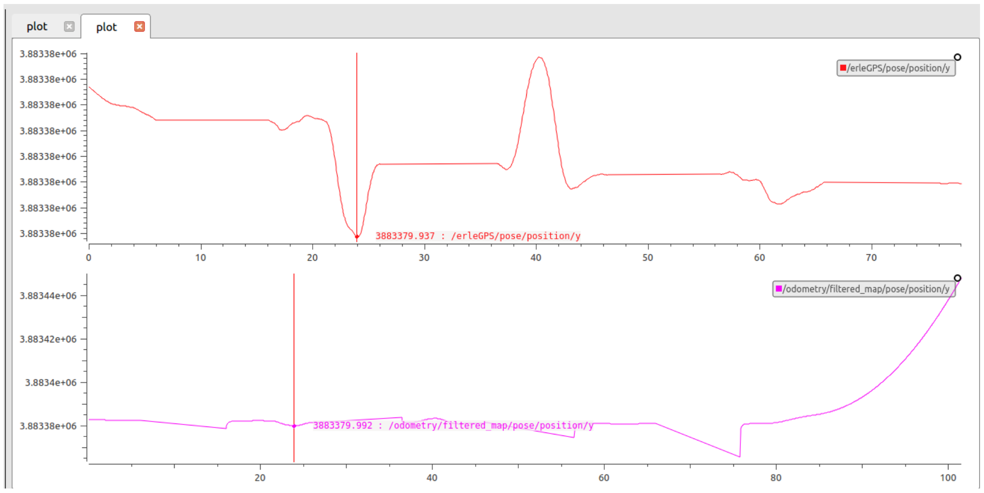
Figure 5.30: Roll and Pitch.

5.4.3.2 Poses estimation during tracking

The following graphs show the data (recorded in a rosbag during a tracking process). 5.31(a) presents the GPS data along with the output of the implemented kalman filter for the X-position meanwhile 5.31(b) shows these data for the Y-position. The small amount of errors (table 5.4) between the GPS and estimated data proves the effectiveness of the method.



(a) Estimated Position along the X-axis (Kalman Filter(Green) GPS data(Blue))



(b) Estimated Position along the Y-axis (Kalman Filter(Pink) GPS data(red))

Figure 5.31: Estimation.

Error at 24 seconds	
Axis	Error
X-axis	0.104
Y-axisS	0.055

Table 5.4: Errors between GPS data and estimated data.

Chapter 6

Conclusion

This thesis presented a method for the detection and tracking of a ground target using a quadrotor vehicle built with low-cost components and well known techniques such as kalman filtering and visual servoing. The thesis also provided a detailed analysis of the quadrotor dynamics and control using systems and control theories. A simulation and real-implementation was performed and showed good results. The autonomous behavior was guaranteed by the robot operating system, yet the lack of real-time capabilities of ROS made difficult the transition between the simulation and the real-implementation. A migration to ROS2, a new extension of ROS could solve for instance the problem of timestamps during the frames transform processes. Finally, we proved that using good knowledge of the quadrotor behavior, pose estimation methods and basic knowledge of computers vision and visual servoing control, a low-cost system based on quadrotor can be used for detection and tracking of moving objects.

References

[1] F. P. Rivara and C. D Mack, “Motor vehicle crash deaths related to police pursuits in the united states”, *INJURY PREVENTION*, no. 2, p. 93, 2004, ISSN: 1353-8047.

[2] (). UNM releases 2018 campus security and fire safety report, UNM Newsroom, [Online]. Available: `https://news.unm.edu/news/unm-releases-2018-campus-security-and-fire-safety-report` (visited on 03/12/2019).

[3] L. M. Argentim, W. C. Rezende, P. E. Santos, and R. A. Aguiar, “Pid, lqr and lqr-pid on a quadcopter platform”, in *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, 2013, pp. 1–6. DOI: `10.1109/ICIEV.2013.6572698`.

[4] I. Meslouli, R. Mesli, M. Kahouadji, A. Choukchou-Braham, and B. Cherki, “Quadrotor design procedure and pid control for outdoor free flight”, in *2018 International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM)*, 2018, pp. 1–6. DOI: `10.1109/CISTEM.2018.8613602`.

[5] D. Mellinger, N. Michael, and V. Kumar, “Trajectory generation and control for precise aggressive maneuvers with quadrotors”, *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012. DOI: `10.1177/0278364911434236`. eprint: `https://doi.org/10.1177/0278364911434236`. [Online]. Available: `https://doi.org/10.1177/0278364911434236`.

[6] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors”, in *2011 IEEE International Conference on Robotics and Automation*, Shanghai,

China: IEEE, May 2011, pp. 2520–2525, ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5980409. [Online]. Available: http://ieeexplore.ieee.org/document/5980409/ (visited on 03/10/2019).

[7] Z. Bodó and B. Lantos, "Modeling and control of outdoor quadrotor uavs", in *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, 2018, pp. 000 111–000 116. DOI: 10.1109/SISY.2018.8524697.

[8] C. C. Zhih, S. K. V. Ragavan, and M. Shanmugavel, "Development of a simple, low-cost autopilot system for multi-rotor uavs", in *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, 2015, pp. 285–289. DOI: 10.1109/RAICS.2015.7488429.

[9] Y. Li, Y. Yao, F. He, and X. Hu, "Autonomous control and target tracking algorithm design for a quadrotor", in *2017 36th Chinese Control Conference (CCC)*, 2017, pp. 6749–6754. DOI: 10.23919/ChiCC.2017.8028422.

[10] X. Wei, "Autonomous control system for the quadrotor unmanned aerial vehicle", in *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2016, pp. 796–799. DOI: 10.1109/URAI.2016.7733984.

[11] A.-S. Firas Abdullah Thweny and R. A. Sabar, "Design and implementation of autopilot system for quadcopter.", *International Journal of Computer Science Engineering Technology*, vol. 5, no. 6, pp. 190 –199, 2015, ISSN: 22310711. [Online]. Available: http://libproxy.unm.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=aci&AN=115923148&site=eds-live&scope=site.

[12] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches", *IEEE Robotics Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006, ISSN: 1070-9932. DOI: 10.1109/MRA.2006.250573.

[13] ——, “Visual servo control. ii. advanced approaches [tutorial]”, *IEEE Robotics Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007, ISSN: 1070-9932. DOI: `10.1109/MRA.2007.339609`.

[14] R. Fonseca and W. Creixell, “Tracking and following a moving object with a quadcopter”, in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2017, pp. 1–6. DOI: `10.1109/AVSS.2017.8078463`.

[15] A. G. Kendall, N. N. Salvapantula, and K. A. Stol, “On-board object tracking control of a quadcopter with monocular vision”, in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 404–411. DOI: `10.1109/ICUAS.2014.6842280`.

[16] J. Thomas, J. Welde, G. Loianno, K. Daniilidis, and V. Kumar, “Autonomous flight for detection, localization, and tracking of moving targets with a small quadrotor”, *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1762–1769, 2017, ISSN: 2377-3766. DOI: `10.1109/LRA.2017.2702198`.

[17] R. Mahony, V. Kumar, and P. Corke, “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor”, *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012, ISSN: 1070-9932. DOI: `10.1109/MRA.2012.2206474`.

[18] G. Andrew. (). Quadcopter dynamics, simulation, and control, [Online]. Available: `http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/`.

[19] P. Corke, *Robotics, vision and control: Fundamental algorithms in matlab*. Berlin: Springer, 2011, pp. 16–41.

[20] (). Understanding euler angles | CH robotics, [Online]. Available: `http://www.chrobotics.com/library/understanding-euler-angles` (visited on 03/14/2019).

[21] Y. Shtessel, C. Edwards, L. Fridman, and A. Levant, *Sliding mode control and observation*. Springer, 2014.

[22] F. Bensalah and F. Chaumette, “Compensation of abrupt motion changes in target tracking by visual servoing”, in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 1, 1995, 181–187 vol.1. DOI: `10.1109/IROS.1995.525794`.

[23] (). Introduction | erle robotics docs, [Online]. Available: `http://docs.erlerobotics.com/` (visited on 04/04/2019).

[24] (). Ar_track_alvar - ROS wiki, [Online]. Available: `http://wiki.ros.org/ar_track_alvar` (visited on 07/22/2019).

[25] (). Erle-brain 3 | erle robotics docs, [Online]. Available: `http://docs.erlerobotics.com/brains/erle-brain-3` (visited on 04/10/2019).

[26] (). Augmented reality / 3d tracking | ALVAR, [Online]. Available: `http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html` (visited on 07/22/2019).

[27] P. Corke, *Robotics, vision and control: Fundamental algorithms in matlab*. Berlin: Springer, 2011, pp. 28–32.

[28] D. Knuth. (). Knuth: Computers and typesetting, [Online]. Available: `http://www-cs-faculty.stanford.edu/~uno/abcde.html`.

[29] C. KAYMAK and A. UCAR, “Implementation of object detection and recognition algorithms on a robotic arm platform using raspberry pi”, in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, Sep. 2018, pp. 1–8. DOI: `10.1109/IDAP.2018.8620916`.

[30] I. Kalutskiy, S. Spevakova, and I. Matiushin, “Method of moving object detection from mobile vision system”, in *2019 International Russian Automation Conference (RusAutoCon)*, Sep. 2019, pp. 1–5. DOI: `10.1109/RUSAUTOCON.2019.8867632`.

[31] P. Shang, Y. Gao, and Z. Song, "Low-cost quadrotor attitude solution based on improved complementary filtering", in *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Dec. 2018, pp. 1037–1041. DOI: `10.1109/ITOEC.2018.8740394`.

[32] M. S Arifin, Y. Y. Nazaruddin, T. A. Tamba, R. A. Santosa, and A. Widyotriatmo, "Experimental modeling of a quadrotor UAV using an indoor local positioning system", in *2018 5th International Conference on Electric Vehicular Technology (ICEVT)*, Oct. 2018, pp. 25–30. DOI: `10.1109/ICEVT.2018.8628424`.

[33] K. K. Deveerasetty and Y. Zhou, "PID with derivative filter and integral sliding-mode controller techniques applied to an indoor micro quadrotor", in *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, Oct. 2018, pp. 439–444.

[34] Z. Bodó and B. Lantos, "Modeling and control of outdoor quadrotor UAVs", in *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, ISSN: 1949-0488, 1949-047X, Sep. 2018, pp. 000 111–000 116. DOI: `10.1109/SISY.2018.8524697`.

[35] D. Kucherov, A. Kozub, and A. Rasstrygin, "Setting the PID controller for controlling quadrotor flight: A gradient approach", in *2018 IEEE 5th International Conference on Methods and Systems of Navigation and Motion Control (MSNMC)*, Oct. 2018, pp. 90–93. DOI: `10.1109/MSNMC.2018.8576294`.

[36] A. Mashood, A. Dirir, M. Hussein, H. Noura, and F. Awwad, "Quadrotor object tracking using real-time motion sensing", in *2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, ISSN: 2159-2055, Dec. 2016, pp. 1–4. DOI: `10.1109/ICEDSA.2016.7818504`.

[37] Y. Liu and Z. Meng, "Visual object tracking for a nano-scale quadrotor", in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Nov. 2018, pp. 843–847. DOI: `10.1109/ICARCV.2018.8581319`.

[38] (). IEEE xplore full-text PDF: [Online]. Available: `https://ieeexplore-ieee-org.libproxy.unm.edu/stamp/stamp.jsp?tp=&arnumber=7991411` (visited on 11/11/2019).

[39] Y. Ma, P. Pei, C. Xiang, S. Yao, and Y. Gao, "KCF based 3d object tracking via RGB-d camera of a quadrotor", in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2017, pp. 939–944. DOI: `10.1109/ICUAS.2017.7991411`.

[40] G. Coelho, E. Kougianos, S. P. Mohanty, P. Sundaravadivel, and U. Albalawi, "An IoT-enabled modular quadrotor architecture for real-time aerial object tracking", in *2015 IEEE International Symposium on Nanoelectronic and Information Systems*, Dec. 2015, pp. 197–202. DOI: `10.1109/iNIS.2015.10`.

[41] C.-T. Dang, H.-T. Pham, T.-B. Pham, and N.-V. Truong, "Vision based ground object tracking using AR.drone quadrotor", in *2013 International Conference on Control, Automation and Information Sciences (ICCAIS)*, Nov. 2013, pp. 146–151. DOI: `10.1109/ICCAIS.2013.6720545`.