People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

MASTER

In **Control Engineering**

Option: **Control Engineering**

Title:

Control Design and Visual Autonomous Navigation of Quadrotor

Presented by:

- **BOUGHELLABA Mouaad**
- **RABAH HAZILA Ramzi**

Supervisor:

Dr. BOUSHAKI Razika

Registration Number:………/2018

Acknowledgment

First and foremost, we are thankful to ALLAH almighty, for showing heavenly blessing upon us, as without that nothing would have been possible.

We would like to express our sincere gratitude and appreciation to our supervisor, Dr. R.BOUSHAKI for her considerable assistance, guidance and encouragement, as well as, Mr. K.Cherifi for his help, guidance and support throughout the year, Mr. T.BOUAMER for his guidance and for providing us with Ardrone2.0 to conduct the experiments.

Dedication

I would like to dedicate this work:

To my dear father and my lovely mother for their ultimate support

To my brother and my sisters

To my Grandparents, my uncles and all my family specially Rezak

To all my dear friends.

Ramzi

I would like to dedicate this work:

To my lovely mother and my father for their ultimate support

To my brother and my sisters

To all my family and friends.

Mouaad

Abstract

Starting from the fact that quadrotors are nonlinear MIMO system that operates in 3D space, the task of stabilizing and generating suitable control commands have been the interest of many researches. Another challenging task is the autonomous navigation as both the weight and the computation capacity are limited which constrains the type of sensors and algorithms.

In this project, an autonomous navigation and obstacle avoidance system based on monocular camera has been implemented which enables the quadrotor navigates in previously unknown GPS-denied environment. Moreover, four controllers have been designed and their performance were compared.

The mathematical model of a quadrotor has been derived using Newton's and Euler's laws, where a linear and nonlinear version of the model are presented, based on that various control strategies such as LQR, PID, Feedback Linearization with pole placement and Sliding Mode control Have been implemented in MATLAB/Simulink and discussed.

Sensor data and the camera video stream have been used by a Keyframe visual SLAM system to compute the location of the drone and generate the 3D map of the environment in the form of point cloud. This point cloud data is clustered and used for obstacle detection. Moreover a PRM algorithm has been used to generate a collision-free path that will be followed by the drone based on the PID controller designed.

We implemented our approach on a real Parrot ARDrone2.0, and our approach has been validated with experiments. All computations are performed on a ground station, which is connected to the drone via wireless LAN.

Contents

List of figures:

Introduction

With the huge technological development, the role of robots is becoming integral in humans life, and especially aerial robots. They have been employed in many key activities such as agriculture and environmental work, where they have been used to collect data of large surfaces of fields, investigating water consumption rates, security. It has also been used to detect gas leaks from oil pipelines.

Autonomous flight robots can assist human to complete various tasks, however they need to acquire the current pose and environment information for localization, motion planning and control

• Problem statement:

Control and autonomous obstacle avoidance are without doubt the most important components in the success of micro aerial vehicles (MAVs). The challenge to achieve that rise from the fact that MAVs are not equipped with a high accuracy sensors, deploying new sensors might not be suitable because of the high weight and power consumption of these sensors. Monocular cameras are the appropriate and attractive lightweight sensors that can provide enough data about the surrounding environment.

The objective of this thesis is to implement and compare a number of control methods then choosing the most suitable one to develop an obstacle avoidance system that is capable of navigating in an unknown environment using only onboard sensors, without markers or calibration objects.

• Previous works:

[1] Used a MAV equipped with a rotating laser range sensor in order to have an Omnidirectional sensing of the environment. Due to the accuracy of the sensing systems, the authors were able to obtain an accurate point cloud of the environment, to detect the obstacles and to successfully avoid them. In our work we have used monocular camera which is more challenging with respect of the use of the stereo cameras.

[2] Addresses this challenge by producing a dense depth map from the monocular camera using the hovering function of the Ardrone. The depth map is used to compute locally a new collision-free waypoint for the MAV to follow by calculating the furthest 3D point reachable from the MAV without collisions.

[3] Developed a small scale indoor collision avoidance system based on Tum_ardrone ROS package. However in our work a K-means clustering methods has been used for obstacle avoidance, Moreover an occupancy grid map has been used all along with graph-based collision avoidance algorithm instead of geometric algorithm.

• Quadrotors

A Quadrotor is an UAV with four rotors. Varying the speeds of the rotors helps to control the position and the orientation of the robot. The adjacent rotors have opposite sense of rotation. This is done to balance the total angular momentum of the craft, otherwise the UAV will start rotating about itself. The Quadrotor has 6 degrees of freedom but only four actuators (Rotors). Hence, Quadrotors are underactuated. The Rotors produce thrust, torque and drag force and the control input to the system is the angular velocity of the motors. A low level controller stabilizes the rotational speed of each blade. The Quadrotor can perform Vertical Take Off and Landing (VTOL), hover and make slow precise movements. The four rotors provide a higher payload capacity.

• Components of Autonomous Flight

In order to enable autonomous flight, there are a few components/modules that we need in any UAV. They are listed as below:

- State Estimation: Estimate the position and velocity (including rotation and angular velocity of the robot).
- Control: Command motors and produce desired actions in order to navigate to the desired state.
- Mapping: The vehicle must have some basic capability to map its environment. If it does not know what the surrounding environment looks like, then it’s incapable of reasoning about this environment and planning safe trajectories in this environment.
- Planning: Finally, the vehicle must be able to compute a trajectory, given a set of obstacles and a destination.

• Outline

The rest of this thesis is organized as follows:

- **Chapter 1:** This chapter presents the mathematical mode of a quadrotor UAV which has been derived based on the Newton-Euler laws, including the linear and nonlinear version of the model.
- **Chapter 2:** This chapter shows four developed control strategies to control the attitude, altitude and position of the quadrotor in space. The controllers have been verified and simulated in MATLAB/Simulink.
- **Chapter 3:** This chapter presents the theoretical background about the components used to implement the autonomous navigation system.
- **Chapter 4:** Finally in this chapter the results of controllers and the autonomous navigation system have been shown and discussed.

Chapter 1

System Modeling

1.1 Kinematic Model

In order to discuss the modeling of the quadrotor, we first need to define the coordinate frames that will be used. Figure 1-1 shows the Earth reference frame with N, E and D axes and the body frame with x, y and z axes. The Earth frame is an inertial frame fixed on a specific place at ground level as its name implies, it uses the N-E-D notation where the axes point to the North, East and downwards respectively. On the other hand, the body frame is at the center of the quadrotor body, with its x-axis pointing towards propeller 1, y-axis pointing towards propeller 2 and the z-axis is pointing to the ground.

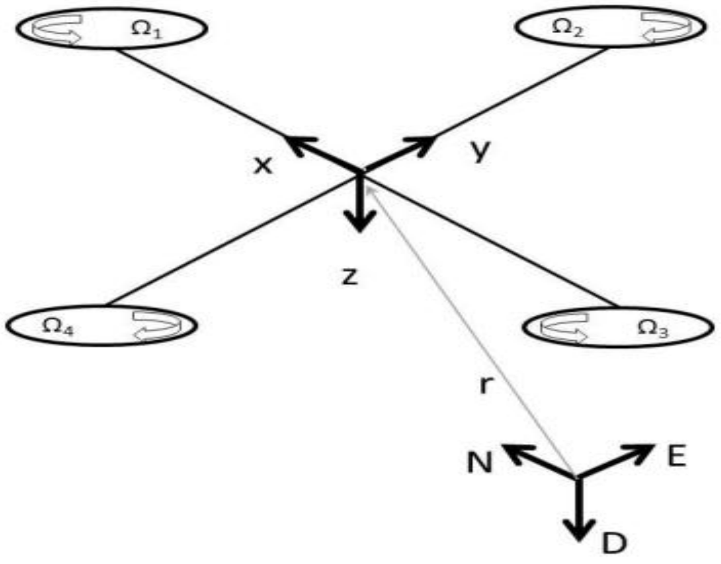


Figure 1-1: Quadrotor Reference Frames

The distance between the Earth frame and the body frame describes the absolute position of the center of mass of the quadrotor $r = [x\ y\ z]^T$. The rotation R from the body frame to the inertial frame describes the orientation of the quadrotor. The orientation of the quadrotor is described using roll, pitch and yaw angles (φ; θ and ψ) representing rotations about the X, Y and Z-axes respectively. Assuming the order of rotation to be roll (φ), pitch (θ) then yaw (ψ), the rotation matrix R which is derived based on the sequence of principle rotations is:

$$R = \begin{pmatrix} c\theta c\psi & s\varphi s\theta c\psi & c\varphi s\theta c\psi + s\varphi s\psi \\ c\theta s\psi & s\varphi s\theta s\psi + c\theta c\psi & c\varphi s\theta s\psi - s\theta c\psi \\ -s\theta & s\varphi c\theta & c\varphi c\theta \end{pmatrix} \tag{1.1}$$

Where c and s denote cos and sin respectively.

The rotation matrix R will be used in formulating the dynamics model of the quadrotor, its significance is due to the fact that some states are measured in the body frame (e.g. the thrust forces produced by the propellers) while some others are measured in the inertial frame (e.g. the gravitational forces and the quadrotor’s position). Thus, to have a relation between both types of states, a transformation from one frame to the other is needed.

To acquire information about the angular velocity of the quadrotor, typically an on-board Inertial Measurement Unit (IMU) is used which will in turn give the velocity in the body coordinate frame. To relate the Euler rates $\dot{\eta} = \left[\dot{\varphi}\ \dot{\theta}\ \dot{\psi}\right]^T$ that are measured in the inertial frame and angular body rates $\omega = [p\ q\ r]^T$, a transformation is needed as follows:

$$\omega = R_r \dot{\eta} \tag{1.2}$$

Where

$$R_r = \begin{pmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\varphi & \sin\varphi\cos\theta \\ 0 & -\sin\varphi & \cos\varphi\cos\theta \end{pmatrix}$$

Around the hover position, small angle assumption is made where cos $\varphi \equiv 1$, cos $\theta \equiv 1$, sin $\varphi = 0$ and sin $\theta = 0$. Thus R_r can be simplified to an identity matrix I [4].

1.2 Dynamics Model

The motion of the quadrotor can be divided into two subsystems, rotational subsystem (roll, pitch and yaw) and translational subsystem (altitude and *x* and *y* position), the rotational subsystem is fully while the translational subsystem is underactuated [4].

1.2.1 Rotational Equations of Motion

The rotational equations of motion are derived in the body frame using the Newton-Euler method with the following general formalism:

$$J\dot{\omega} + \omega \times J\omega = M_B \tag{1.3}$$

Where:

J	Quadrotor's diagonal inertia Matrix.
ω	Angular body rate.
M_B	Moments acting on the quadrotor in the body frame.

1.2.1.1 Inertia Matrix

The inertia matrix for the quadrotor is a diagonal matrix, the off-diagonal elements, which are the product of inertia, are zero due to the symmetry of the quadrotor.

$$J = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \tag{1.4}$$

Where I_{xx}, I_{yy} and I_{zz} are the area moments of inertia about the principle axes in the body frame.

1.2.1.2 Moments Acting on the Quadrotor (M_B)

For the last term of equation (1.3), there is a need to define two physical effects which are the aerodynamic forces and moments produced by a rotor. As an effect of rotation, there is a generated force called the aerodynamic force or the lift force and there is a generated moment called the aerodynamic moment. Equations (1.5) and (1.6) show the aerodynamic force F_i and moment M_i produced by the i_{th} rotor [5].

$$F_i = \frac{1}{2}\rho A C_T r^2 \Omega_i^2 \tag{1.5}$$

$$M_i = \frac{1}{2}\rho A C_D r^2 \Omega_i^2 \tag{1.6}$$

Where

ρ	Air density.
A	Blade area.
C_T, C_D	Aerodynamic coefficients.
r	Radius of blade.
Ω_i	Angular velocity of rotor *i*.

Clearly, the aerodynamic forces and moments depend on the geometry of the propeller and the air density. Since for the case of quad rotors, the maximum altitude is usually limited, thus the air density can be considered constant, Equations (1.5) and (1.6) can be simplified to [4]:

$$F_i = K_f \Omega_i^2 \tag{1.7}$$

$$M_i = K_M \Omega_i^2 \tag{1.8}$$

Where K_f and K_M are the aerodynamic force and moment constants respectively and Ω_i is the angular velocity of rotor i. The aerodynamic force and moment constants can be determined experimentally for each propeller type.

By identifying the forces and moments generated by the propellers, we can study the moments M_B acting on the quadrotor. Figure 1-2 shows the forces and moments acting on the quadrotor. Each rotor causes an upwards thrust force *Fi* and generates a moment *Mi* with direction opposite to the direction of rotation of the corresponding rotor *i*.

Starting with the moments about the body frame's x-axis, by using the right-hand-rule in association with the axes of the body frame, *F*2 multiplied by the moment arm *l* generates a negative moment about the x-axis, while in the same manner, *F*4 generates a positive moment.

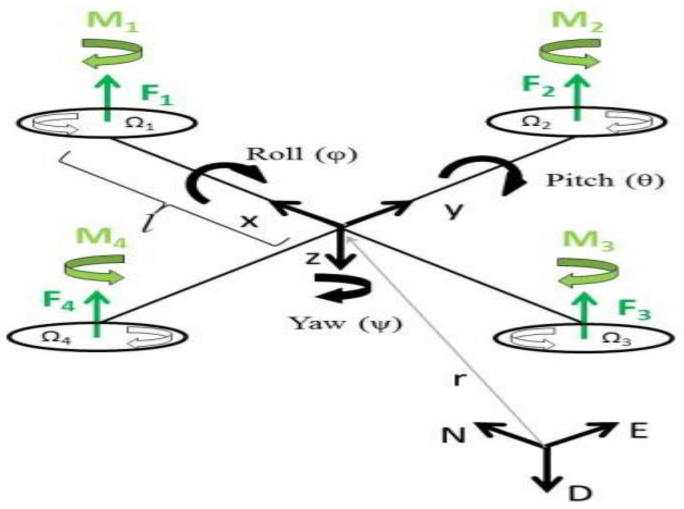


Figure 1-2: Forces and Moments acting on Quadrotor.

Thus the total moment about the x-axis can be expressed as

$$\begin{aligned} M_x &= -F_2 l + F_4 l \\ &= -\left(K_f \Omega_2^2\right) l + \left(K_f \Omega_4^2\right) l \\ &= l K_f \left(-\Omega_2^2 + \Omega_4^2\right) \end{aligned} \tag{1.9}$$

For the moments about the body frame's y-axis, also using the right-hand-rule, the thrust of rotor 1 generates a positive moment, while the thrust of rotor 3 generates a negative moment about the y-axis. The total moment can be expressed as:

$$\begin{aligned} M_y &= F_1 l - F_3 l \\ &= \left(K_f \Omega_1^2\right) l - \left(K_f \Omega_3^2\right) l \\ &= l K_f \left(\Omega_1^2 - \Omega_3^2\right) \end{aligned} \tag{1.10}$$

For the moments about the body frame's z-axis, the thrust of the rotors does not cause a moment. On the other hand, moment caused by the rotors' rotation as per Equation (1.6). By using the right-hand-rule, the moment about the body frame's z-axis can be expressed as:

$$\begin{aligned} M_z &= M_1 - M_2 + M_3 - M_4 \\ &= \left(K_M \Omega_1^2\right) - \left(K_M \Omega_2^2\right) + \left(K_M \Omega_3^2\right) - \left(K_M \Omega_4^2\right) \\ &= K_M \left(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) \end{aligned} \tag{1.11}$$

Combining equations (1.9), (.10) and (1.11) in vector form, we get:

$$M_B = \begin{pmatrix} lK_f\left(-\Omega_2^2 + \Omega_4^2\right) \\ lK_f\left(\Omega_1^2 - \Omega_2^2\right) \\ K_M\left(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) \end{pmatrix} \tag{1.12}$$

Where *l* is the arm length, which is the distance between the axis of rotation of each rotor to the origin of the body reference frame which should coincide with the center of the quadrotor.

1.2.2 Translational Equations of Motion

The translation equations of motion for the quadrotor are based on Newton's second law and they are derived in the Earth inertial frame [4]:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + RF_B \tag{1.13}$$

Where

m	Quadrotor's mass.
g	Gravitational acceleration.
F_B	Nongravitational forces acting on the quadrotor in the body frame.

1.2.2.1 Nongravitational Forces Acting on the Quadrotor

When the quadrotor is in a horizontal orientation (i.e. it is not rolling or pitching), the only nongravitational forces acting on it is the thrust produced by the rotation of the propellers which is proportional to the square of the angular velocity of the propeller as per Equation (1.7). Thus, the nogravitational forces acting on the quadrotor *FB*, can be expressed as:

$$F_B = \begin{bmatrix} 0 \\ 0 \\ -K_f\left(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2\right) \end{bmatrix} \tag{1.14}$$

The first two rows of the force vector are zeros as there is no forces in the X and Y directions, the last row is simply an addition of the thrust forces produced by the four propellers. The negative sign is due to the fact that the thrust is upwards while the positive z-axis in the body framed is pointing downwards. *FB* is multiplied by the rotation matrix *R* to transform the thrust forces of the rotors from the body frame to the inertial frame, so that the equation can be applied in any orientation of the quadrotor.

1.3 Aerodynamic Effects

In the previous dynamics formulation, the aerodynamic effects acting on the quadrotor body were neglected. However, in order to have an accurate and realistic model to be used in simulations, aerodynamic effects should be included. There are namely two types of aerodynamic effects, drag forces and drag moments [6].

1.3.1 Drag Forces

Due to the friction of the moving quadrotor body with air, a force acts on the body of the quadrotor resisting the motion. As the velocity of travel of the quadrotor increases, the drag forces in turn increase. The drag forces *Fa* can be approximated by:

$$F_a = K_t \dot{r} \tag{1.15}$$

Where K_t is a constant matrix called the aerodynamic translation coefficient matrix and $\dot{r}$ is the time derivative of the position vector r. This indicates that there is an extra force acting on the quadrotor body, the translational equation of motion Equation (1.13) should be rewritten to be:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + RF_B - F_a \tag{1.16}$$

1.3.2 Drag Moments

The same as the drag force, due to the air friction, there is a drag moment M_a acting on the quadrotor body which can be approximated by [5]:

$$M_a = K_r\dot{\eta} \tag{1.17}$$

Where K_r is a constant matrix called the aerodynamic rotation coefficient matrix and $\dot{\eta}$ is the Euler rates. Accordingly, the rotational equation of motion expressed by Equation (1.3) can be rewritten to as [5]:

$$J\dot{\omega} + \omega \times J\omega = M_B - M_a \tag{1.18}$$

1.4 State Space Model

Formulating the acquired mathematical model for the quadrotor into a state space model will help make the control problem easier to tackle.

1.4.1 State Vector X

Defining the state vector of the quadrotor to be:

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} \end{bmatrix}^T \tag{1.19}$$

Which is mapped to the degrees of freedom of the quadrotor in the following manner:

$$X = \begin{bmatrix} \varphi & \dot{\varphi} & \theta & \dot{\theta} & \psi & \dot{\psi} & z & \dot{z} & x & \dot{x} & y & \dot{y} \end{bmatrix}^T \tag{1.20}$$

The state vector defines the position of the quadrotor in space and its angular and linear velocities.

1.4.2 Control Input Vector U

A control input vector U consisting of four inputs, U_1 through U_4 is defined as:

$$U = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^T \tag{1.21}$$

Where

$$U_1 = K_f \left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \tag{1.22}$$

$$U_2 = K_f \left(-\Omega_2^2 + \Omega_4^2\right) \tag{1.23}$$

$$U_3 = K_f \left(\Omega_1^2 - \Omega_3^2\right) \tag{1.24}$$

$$U_4 = K_M \left(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) \tag{1.25}$$

U_1 is the resulting upwards force of the four rotors which is responsible for the altitude of the quadrotor and its rate of change ($z; \dot{z}$). U_2 is the difference in thrust between rotors 2 and 4 which is responsible for the roll rotation and its rate of change ($\varphi; \dot{\varphi}$). U_3 on the other hand represents the difference in thrust between rotors 1 and 3 thus generating the pitch rotation and its rate of change ($\theta; \dot{\theta}$). Finally U_4 is the difference in torque between the two clockwise turning rotors and the two counterclockwise turning rotors generating the yaw rotation and ultimately its rate of change ($\psi ; \dot{\psi}$). This choice of the control vector U decouples the rotational system, where U_1 will generate the desired altitude of the quadrotor, U_2 will generate the desired roll angle, the desired pitch angle will be generated by U_3 whereas U_4 will generate the desired heading.

1.4.3 Rotational Equation of Motion

Substituting equations (1.21) through (1.24) in equation (1.12), the equation of the total moments acting on the quadrotor becomes:

$$M_B = \begin{bmatrix} lU_2 \\ lU_3 \\ U_4 \end{bmatrix} \tag{1.26}$$

Substituting (1.25) into the rotational equation of motion (1.3) and expanding each term with their prior definition, the following relation can be derived:

$$\begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \begin{bmatrix} \ddot{\varphi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} + \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} lU_2 \\ lU_3 \\ U_4 \end{bmatrix}$$

Expanding that, leads to:

$$\begin{bmatrix} I_{xx}\ddot{\varphi} \\ I_{yy}\ddot{\theta} \\ I_{zz}\ddot{\psi} \end{bmatrix} + \begin{bmatrix} \dot{\theta} I_{zz}\dot{\psi} - \dot{\psi} I_{yy}\dot{\theta} \\ \dot{\psi} I_{xx}\dot{\varphi} - \dot{\varphi} I_{zz}\dot{\psi} \\ \dot{\varphi} I_{yy}\dot{\theta} - \dot{\theta} I_{xx}\dot{\varphi} \end{bmatrix} = \begin{bmatrix} lU_2 \\ lU_3 \\ U_4 \end{bmatrix} \tag{1.27}$$

Rewriting the last equation to have the angular accelerations in terms of the other variables:

$$\ddot{\varphi} = \frac{l}{I_{xx}} U_2 + \frac{I_{yy}}{I_{xx}} \dot{\psi}\dot{\theta} - \frac{I_{zz}}{I_{xx}} \dot{\theta}\dot{\psi} \tag{1.28}$$

$$\ddot{\theta} = \frac{l}{I_{yy}} U_3 + \frac{I_{zz}}{I_{yy}} \dot{\varphi}\dot{\psi} - \frac{I_{xx}}{I_{yy}} \dot{\psi}\dot{\varphi} \tag{1.29}$$

$$\ddot{\psi} = \frac{l}{I_{zz}} U_4 + \frac{I_{xx}}{I_{zz}} \dot{\theta}\dot{\varphi} - \frac{I_{yy}}{I_{zz}} \dot{\varphi}\dot{\theta} \tag{1.30}$$

To simplify, define:

$$a_1 = \frac{I_{yy} - I_{zz}}{I_{xx}} \qquad b_1 = \frac{l}{I_{xx}}$$

$$a_2 = \frac{I_{zz} - I_{xx}}{I_{yy}} \qquad b_2 = \frac{l}{I_{yy}}$$

$$a_3 = \frac{I_{xx} - I_{yy}}{I_{zz}} \qquad b_3 = \frac{l}{I_{zz}}$$

Using the above definition of a_1 *to* a_3 and b_1 *to* b_3, equations (1.27) through (1.29) can then be rewritten in a simpler form in terms of the system states:

$$\ddot{\varphi} = b_1 U_2 + a_1 x_4 x_6 \tag{1.31}$$

$$\ddot{\theta} = b_2 U_3 + a_2 x_2 x_6 \tag{1.32}$$

$$\ddot{\psi} = b_3 U_4 + a_3 x_2 x_4 \tag{1.33}$$

With the choice of the control input vector *U*, it is clear that the rotational subsystem is fully-actuated, it is only dependent on the rotational state variables x_1 *to* x_6 that correspond $\varphi, \dot{\varphi}, \theta, \dot{\theta}, \psi$ *and* $\dot{\psi}$ respectively.

1.4.4 Translational Equation of Motion

Substituting equations (1.21) through (1.24) in equation (1.13), the equation of the total moments acting on the quadrotor becomes:

$$F_B = \begin{bmatrix} 0 \\ 0 \\ -U_1 \end{bmatrix} \tag{1.34}$$

Embedding that into the translational equation of motion (1.13) and expanding the terms, we get:

$$m\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{pmatrix} c\theta c\psi & s\varphi s\theta c\psi & c\varphi s\theta c\psi + s\varphi s\psi \\ c\theta s\psi & s\varphi s\theta s\psi + c\theta c\psi & c\varphi s\theta s\psi - s\theta c\psi \\ -s\theta & s\varphi c\theta & c\varphi c\theta \end{pmatrix} \begin{bmatrix} 0 \\ 0 \\ -U_1 \end{bmatrix}$$

$$m\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{bmatrix} (s\varphi s\psi + c\varphi c\psi s\theta)(-U_1) \\ (c\varphi s\psi s\theta - c\psi s\varphi)(-U_1) \\ (c\varphi c\theta)(-U_1) \end{bmatrix} \tag{1.35}$$

Rewriting Equation (1.34) to have the accelerations in terms of the other variables, we get:

$$\ddot{x} = \frac{-U_1}{m}\left(\sin\varphi\sin\psi + \cos\varphi\cos\psi\sin\theta\right) \tag{1.36}$$

$$\ddot{y} = \frac{-U_1}{m}\left(\cos\varphi\sin\psi\sin\theta - \cos\psi\sin\varphi\right) \tag{1.37}$$

$$\ddot{z} = g - \frac{U_1}{m}\left(\cos\varphi\cos\theta\right) \tag{1.38}$$

Rewriting in terms of the state variable X

$$\ddot{x} = \frac{-U_1}{m}\left(\sin x_1 \sin x_5 + \cos x_1 \cos x_5 \sin x_3\right) \tag{1.39}$$

$$\ddot{y} = \frac{-U_1}{m}\left(\cos x_1 \sin x_5 \sin x_3 - \cos x_5 \sin x_1\right) \tag{1.40}$$

$$\ddot{z} = g - \frac{U_1}{m}\left(\cos x_1 \cos x_3\right) \tag{1.41}$$

It is clear here that the translational subsystem is underactuated as it dependent on both the translational state variables and the rotational ones.

1.4.5 State Space Representation

Using the equations of the rotational angular acceleration. Equations (1.31) to (1.33), and those of translation, Equations (1.39) to (1.41), the complete mathematical model of the quadrotor can be written in a state space representation as follows:

$$\dot{x}_1 = \dot{\varphi} = x_2$$
$$\dot{x}_2 = \ddot{\varphi} = a_1 x_4 x_6 + b_1 U_2$$
$$\dot{x}_3 = \dot{\theta} = x_4$$
$$\dot{x}_4 = \ddot{\theta} = a_2 x_2 x_6 + b_2 U_3$$
$$\dot{x}_5 = \dot{\psi} = x_6$$
$$\dot{x}_6 = \ddot{\psi} = a_3 x_2 x_4 + b_3 U_4$$
$$\dot{x}_7 = \dot{z} = x_8$$
$$\dot{x}_8 = \ddot{z} = g - \frac{U_1}{m}\left(\cos x_1 \cos x_3\right)$$
$$\dot{x}_9 = \dot{x} = x_{10}$$
$$\dot{x}_{10} = \ddot{x} = \frac{-U_1}{m}\left(\sin x_1 \sin x_5 + \cos x_1 \cos x_5 \sin x_3\right)$$
$$\dot{x}_{11} = \dot{y} = x_{12}$$
$$\dot{x}_{12} = \ddot{y} = \frac{-U_1}{m}\left(\cos x_1 \sin x_5 \sin x_3 - \cos x_5 \sin x_1\right)$$

$$f\left(X,U\right)=\begin{bmatrix} x_2 \\ a_1x_4x_6+b_1U_2 \\ x_4 \\ a_2x_2x_6+b_2U_3 \\ x_6 \\ a_3x_2x_4+b_3U_4 \\ x_8 \\ g-\frac{U_1}{m}\left(\cos x_1\cos x_3\right) \\ x_{10} \\ \frac{-U_1}{m}\left(\sin x_1\sin x_5+\cos x_1\cos x_5\sin x_3\right) \\ x_{12} \\ \frac{-U_1}{m}\left(\cos x_1\sin x_5\sin x_3-\cos x_5\sin x_1\right) \end{bmatrix} \tag{1.42}$$

It is possible to rewrite the state space equation (1.42) in the following form:

$$\dot{\mathrm{X}}=f\left(\mathrm{X}\right)+\sum_{i=1}^{4}g_i\left(\mathrm{X}\right)u_i \tag{1.43}$$

Where

$$f\left(X\right)=\begin{bmatrix} x_2 \\ a_1x_4x_6 \\ x_4 \\ a_2x_2x_6 \\ x_6 \\ a_3x_2x_4 \\ x_8 \\ g \\ x_{10} \\ 0 \\ x_{12} \\ 0 \end{bmatrix} \qquad (1.44)$$

$$g_1=\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & G_1 & 0 & G_2 & 0 & G_3\end{bmatrix}^T \in \mathbb{R}^{12}$$
$$g_2=\begin{bmatrix}0 & b_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T \in \mathbb{R}^{12}$$
$$g_3=\begin{bmatrix}0 & 0 & 0 & b_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T \in \mathbb{R}^{12}$$
$$g_4=\begin{bmatrix}0 & 0 & 0 & 0 & 0 & b_3 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T \in \mathbb{R}^{12}$$

With

$$G_1=-\frac{1}{m}\left(\cos x_1 \cos x_3\right)$$
$$G_2=-\frac{1}{m}\left(\sin x_1 \sin x_5+\cos x_1 \cos x_5 \sin x_3\right)$$
$$G_3=-\frac{1}{m}\left(\cos x_1 \sin x_5 \sin x_3-\cos x_5 \sin x_1\right)$$

1.5 Linear model

Inorder to linearize the system (1.43), we need to find an equilibrium point $\bar{X}$ which for fixed input $\bar{U}$ is the solution of the algebraic system(1.42), or the value of state's vector, which on fixed constant input is the solution of algebraic system:

$$f\left(\bar{X},\bar{U}\right)=0 \tag{1.45}$$

Since the function f is nonlinear, problems related to the existence of a unique solution of system (1.45) arise. In particular, for the system in hand, the solution is difficult to find because of trigonometric functions related each other in no-elementary way. For this reason, the linearization is performed on a simplified model which considers only small oscillations. This simplification is made by approximating the sine function with its argument and the cosine function with unity. The approximation is valid if the argument is small. The resulting system is described by the following equations [8]:

$$\hat{f}\left(X,U\right)=\begin{bmatrix} x_2 \\ a_1x_4x_6+b_1U_2 \\ x_4 \\ a_2x_2x_6+b_2U_3 \\ x_6 \\ a_3x_2x_4+b_3U_4 \\ x_8 \\ g-\dfrac{U_1}{m} \\ x_{10} \\ \dfrac{-U_1}{m}\left(x_1x_5+x_3\right) \\ x_{12} \\ \dfrac{-U_1}{m}\left(x_5x_3-x_1\right) \end{bmatrix} \tag{1.46}$$

1.5.1 Linearization

As said above, in order to perform the linearization, an equilibrium point is needed. The resulting equilibrium point from the above equations:

$$\bar{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & z & 0 & x & 0 & y & 0 \end{bmatrix}^T \in \mathbb{R}^{12} \qquad (1.47)$$

From the equations, we can find that the equilibrium point (2.47) for the input is obtained by the constant input value:

$$\bar{U} = \begin{bmatrix} mg & 0 & 0 & 0 \end{bmatrix}^T \in \mathbb{R}^{4} \qquad (1.48)$$

Note that this particular value represents the force necessary to delete the quadrotor's weight and it consents its hovering. After determined the equilibrium point $\bar{X}$ and the corresponding nominal input $\bar{U}$, we have that the matrices associated to the linear system are given by relations:

$$A = \left.\frac{\partial f(X,U)}{\partial X}\right|_{\substack{X=\bar{X}\\ U=\bar{U}}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (1.49)$$

$$B = \left.\frac{\partial f\left(X,U\right)}{\partial U}\right|_{\substack{X=\bar{X}\\ U=\bar{U}}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & b_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & b_2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b_3 \\ 0 & 0 & 0 & 0 \\ \frac{-1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.50}$$

The linear model is in the form:

$$\dot{X} = A\ X + B\ U \tag{1.51}$$

1.5.2 Controllability of the linear system

Controllability represents major concept of modern control system theory. This concept was introduced by R. Kalman in 1960. If we consider the linear system (1.51) it can be roughly define the controllability as follow:

- Controllability: The pair (A, B) is said to be controllable if for any initial state $X(0)=X_0$ and any final state X_1, there exists an input that transfers X_0 to X_1 in a finite time. Otherwise (A, B) is said to be uncontrollable [8].

Inorder to check if the system (1.51) is controllable or not, we should first define the controllability matrix as follow:

$$c = \left[B \quad A\,B \quad A^2\,B \quad A^3\,B \quad A^4\,B \quad A^5\,B \quad A^6\,B \quad A^7\,B \quad A^8\,B \quad A^9\,B \quad A^{10}\,B \quad A^{11}\,B\right] \in \mathbb{R}^{12\times 48} \tag{1.52}$$

The system (1.51) is controllable if the matrix c (1.52) is full rank, otherwise is not controllable [8]. To check its controllability, we have used Matlab. The linear system (1.51) is found to be controllable.

Block controllability could have been used in this case but the block controllability matrix is not full rank.

Chapter 2

Control Design

In this chapter, the formulated quadrotor model has been used in control design. Four controllers have been developed: LQR, PID, Sliding Mode and Feedback Linearization with pole placement. Computer based simulations have been implemented on MATLAB/Simulink and have been used to assess the performance of the developed controllers.

2.1 Closed loop simulation for the PID and Sliding Mode control design

In this section we will discuss the closed loop configuration that has been used to design the PID and Sliding Mode controllers in MATLAB/Simulink.

2.1.1 Altitude Controller

The altitude controller takes an error signal *e* as an input which is the difference between the desired altitude Z_d and the actual altitude *Z* and produces a control signal U_1, as shown in the block diagram in Figure 2-1.

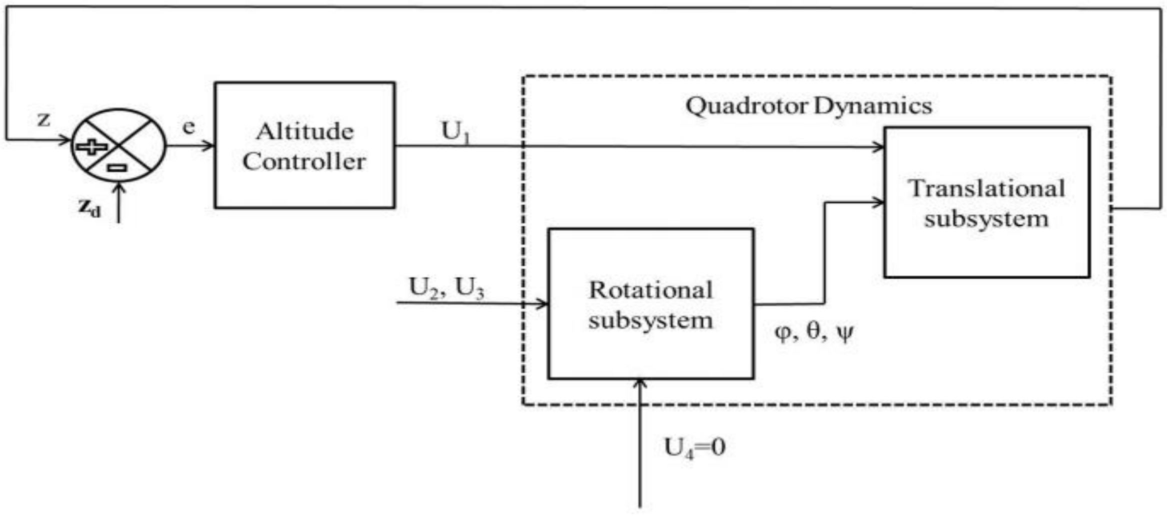


Figure 2-1: Block Diagram for Altitude Controller.

2.1.2 Attitude and Heading Controller

Similar to the altitude controller block, the attitude and heading controller take as an input an error signal *e* which is the difference between the desired roll φ_d , pitch θ_d and yaw ψ_d and their actual values φ, θ *and* ψ . The attitude and heading controller produces the output signals U_2, U_3 and U_4, as shown in Figure 2-2.

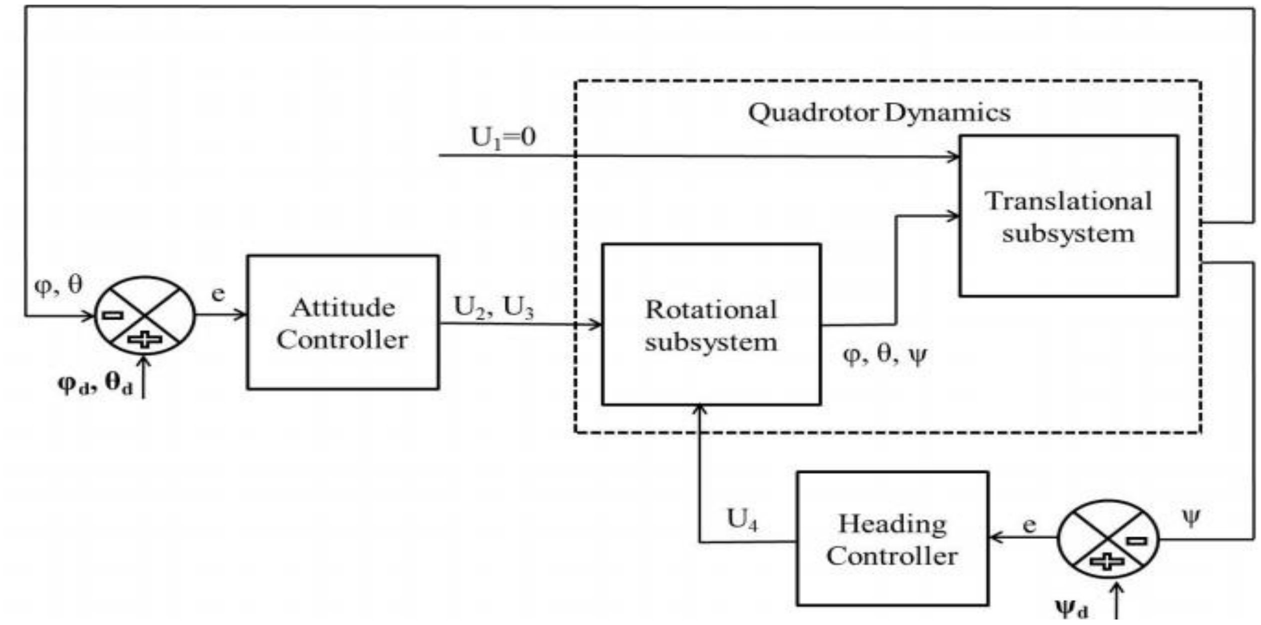


Figure 2-2: Block Diagram for Attitude and Heading Controller.

2.1.3 Position Controller

Unlike the altitude and orientation of the quadrotor, its *x* and *y* position is not decoupled and cannot be directly controlled using one of the four control laws U_1 through U_4. On the other hand, the *x* and *y* position can be controlled through the roll and pitch angles. The desired roll and pitch angles φ_d and θ_d can be calculated from the translational equations of motion, Equations (1.36) and (1.37) as follows:

$$\ddot{x} = \frac{-U_1}{m}\left(\sin\varphi\sin\psi + \cos\varphi\cos\psi\sin\theta\right)$$
$$\ddot{y} = \frac{-U_1}{m}\left(\cos\varphi\sin\psi\sin\theta - \cos\psi\sin\varphi\right)$$

Since the quadrotor is operating around hover, which means small values for the roll and pitch angles φ and θ, we can use the small angle assumption to simplify the above equations (small oscillation):

$$\ddot{x} = \frac{-U_1}{m}\left(\varphi_d\sin\psi + \theta_d\cos\psi\right) \tag{2.1}$$
$$\ddot{y} = \frac{-U_1}{m}\left(\theta_d\sin\psi - \varphi_d\cos\psi\right) \tag{2.2}$$

Which can be written in a matrix form as:

$$\begin{bmatrix} -\sin\psi & -\cos\psi \\ \cos\psi & -\sin\psi \end{bmatrix}\begin{bmatrix} \varphi_d \\ \theta_d \end{bmatrix} = \frac{m}{U_1}\begin{bmatrix} \ddot{x}_d \\ \ddot{y}_d \end{bmatrix} \tag{2.3}$$

Which can be inverted to get:

$$\begin{aligned}\begin{bmatrix} \varphi_d \\ \theta_d \end{bmatrix} &= \begin{bmatrix} -\sin\psi & -\cos\psi \\ \cos\psi & -\sin\psi \end{bmatrix}^{-1}\frac{m}{U_1}\begin{bmatrix} \ddot{x}_d \\ \ddot{y}_d \end{bmatrix} \\ &= \frac{m}{U_1}\begin{bmatrix} -\sin\psi & \cos\psi \\ -\cos\psi & -\sin\psi \end{bmatrix}\begin{bmatrix} \ddot{x}_d \\ \ddot{y}_d \end{bmatrix} \\ &= \frac{m}{U_1}\begin{bmatrix} -\ddot{x}_d\sin\psi + \ddot{y}_d\cos\psi \\ -\ddot{x}_d\cos\psi - \ddot{y}_d\sin\psi \end{bmatrix}\end{aligned} \tag{2.4}$$

The calculated φ_d and θ_d have to be limited to the range between -20^o and 20^o to fulfill the small angle assumption, this can be done via a saturation function in the simulation.

The closed loop simulation for the altitude and attitude controllers is further enhanced to include the position controller as shown in the block diagram in Figure 2-3.

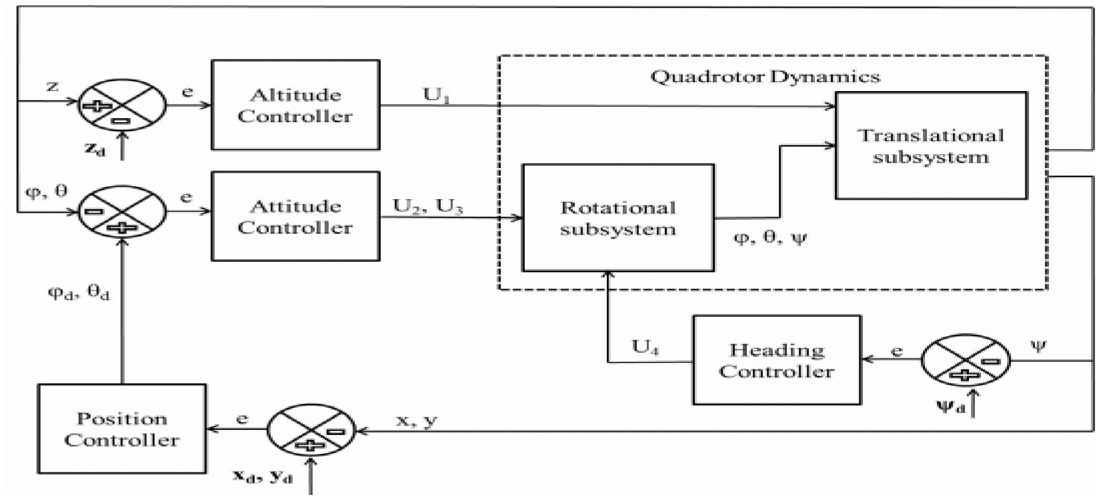


Figure 2-3: Position Controller Block Diagram (Complete System).

The controller blocks in the previous block diagrams have been used to implement the PID and the Sliding Mode controllers, where the controllers input(s) are the error related to some of the quadrotor's states and produce an output which is either one or several control inputs U_1 through U_4 or φ_d and θ_d if it is the position controller.

2.2 PID Control

After the mathematical model of the quadrotor, a PID controller was developed. The PID controller generates the desired control inputs for the quadrotor. The block diagram for a PID controller is shown in Figure 2-4.

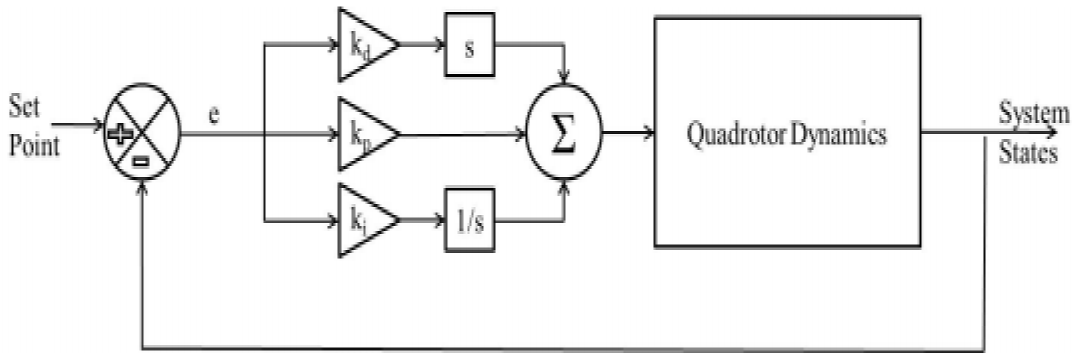


Figure 2-4: PID Controller Block Diagram

2.2.1 Altitude Control

A PID controller is developed to control the altitude of the quadrotor. It generates the control input U_1 which is responsible for the altitude for the quadrotor as per Equation (1.22). The derived control law is as follows:

$$U_1 = k_p\left(z - z_d\right) + k_d\left(\dot{z} - \dot{z}_d\right) + k_i \int \left(z - z_d\right) dt \tag{2.5}$$

Where

k_p	*Proportional gain*
z_d	*Desired altitude*
k_d	*Derivative gain*
$\dot{z}_d$	*Desired altitude rate of change*
k_i	*Integral gain*

2.2.2 Attitude and Heading Control

2.2.2.1 Roll Controller

Another PID controller is developed to control the roll angle φ of the quadrotor. The derived control law generates the input U_2 that controls the roll angle as follows:

$$U_2 = k_p\left(\varphi - \varphi_d\right) + k_d\left(\dot{\varphi} - \dot{\varphi}_d\right) + k_i \int \left(\varphi - \varphi_d\right) dt \tag{2.6}$$

Where

k_p	*Proportional gain*
φ_d	*Desired roll angle*
k_d	*Derivative gain*
$\dot{\varphi}_d$	*Desired roll angle rate of change*
k_i	*Integral gain*

2.2.2.2 Pitch Controller

A PID controller is developed to control the pitch angle θ of the quadrotor. The derived control law generates the input U_3 that controls the pitch angle as follows:

$$U_3 = k_p(\varphi - \varphi_d) + k_d(\dot{\varphi} - \dot{\varphi}_d) + k_i \int (\varphi - \varphi_d)\, dt \tag{2.7}$$

Where

k_p	*Proportional gain*
φ_d	*Desired roll angle*
k_d	*Derivative gain*
$\dot{\varphi}_d$	*Desired roll angle rate of change*
k_i	*Integral gain*

2.2.2.3 Yaw Controller

Similar to the pitch and roll controllers, a yaw controller was developed to generate the control input U_4 based on the following control law:

$$U_4 = k_p(\psi - \psi_d) + k_d(\dot{\psi} - \dot{\psi}_d) + k_i \int (\psi - \psi_d)\, dt \tag{2.8}$$

Where

k_p	*Proportional gain*
ψ_d	*Desired yaw angle*
k_d	*Derivative gain*
$\dot{\psi}_d$	*Desired yaw angle rate of change*
k_i	*Integral gain*

2.2.3 Position Controller

After acquiring stable controllers for the altitude and the attitude of the quadrotor, a complete position controller is developed. PID controllers are used to calculate the desired accelerations $\ddot{x}_d$ and $\ddot{y}_d$:

$$\ddot{x}_d = k_p\left(x_d - x\right) + k_d\left(\dot{x}_d - \dot{x}\right) + k_i \int \left(x_d - x\right) dt \qquad (2.9)$$

$$\ddot{y}_d = k_p\left(y_d - y\right) + k_d\left(\dot{y}_d - \dot{y}\right) + k_i \int \left(y_d - y\right) dt \qquad (2.10)$$

Where

k_p	*Proportional gain.*
x_d	*Desired x position.*
k_d	*Derivative gain.*
$\dot{x}_d$	*Desired x position rate of change.*
y_d	*Desired y position.*
$\dot{y}_d$	*Desired y position rate of change.*
k_i	*Integral gain.*

Plugging the values of the desired accelerations $\ddot{x}_d$ and $\ddot{y}_d$ into Equation (2.4), the desired roll and pitch angles φ_d and θ_d can be calculated which are in turn fed to the attitude controller previously expressed in Equations (2.6) and (2.7).

2.3 Sliding Mode Control

Since the quadrotor system is a nonlinear type system, we proposed using a Sliding Mode Controller (SMC) to control the states of the quadrotor.

2.3.1 Introduction to Sliding Mode Control

A Sliding Mode Control is a Variable Structure Control (VSC). Basically, VSC includes several different continuous functions that map plant state to a control surface. The switching among these functions is determined by plant state which is represented by a switching function [9].

Considering the system to be controlled described by state space equation:

$$x^{(n)} = f\left(x,t\right)+g\left(x,t\right)u \tag{2.11}$$

Where $x(t) = (x, x\ (1),.........., x\ (n\text{-}1)\ )$ is the vector of state variable $f\ (x, t)$ and $g(x, t)$ are both nonlinear functions present the system, u is the control part.

The design of the sliding mode control needed two steps. The choice of the sliding surface, and the design of the control law.

step1: the Choice of the Sliding Surface

The objective is the convergence of state variable x at its desired value .The general formulation of the sliding surface is given by the following equation [10]:

$$s(x)=\sum_{i=1}^{i=n}\lambda_i\, e_i = e_n+\sum_{i=1}^{n-1}\lambda_i\, e_i \tag{2.12}$$

When $\lambda_n = 1$, and λ_i $(i = 1...n\text{-}1)$ present the plan coefficients.

Generally the sliding surface is given by the following linear function:

$$S\left(x\right)=e+\lambda\dot{e} \tag{2.13}$$

Where λ is constant positive value, and $e = x - x_d$.

The idea in the tracking problem is to find a suitable control law that makes the error function e stay on the sliding surface $S(x,t)=0$ for $t \geq 0$. To achieve this, a positive Lyapunov function V is defined as:

$$V(s,x,t)=\frac{1}{2}s^2(x,\mathrm{t}) \tag{2.14}$$

The sufficient condition for the stability of the system is given by:

$$\dot{V}(s,x,t)=\dot{V}(s)=s.\dot{s}<-\eta.|s| \tag{2.15}$$

Where η is the positive value (η > 0).

Step2: the design of the control law

The sliding mode control contains two terms which are equivalent control term and switching control term:

$$U(t)=U_s(t)+U_{eq}(t) \tag{2.16}$$

$U_{eq}(t)$ is the equivalent part of the sliding mode control, i.e. the necessary known part of the control system when $\dot{s}=0$.

$U_s(t)$ is the Sliding control mode define as follow:

$$U_s(t)=-k\, sign(s) \tag{2.17}$$

A block diagram showing the SMC is shown in Figure 2-5.

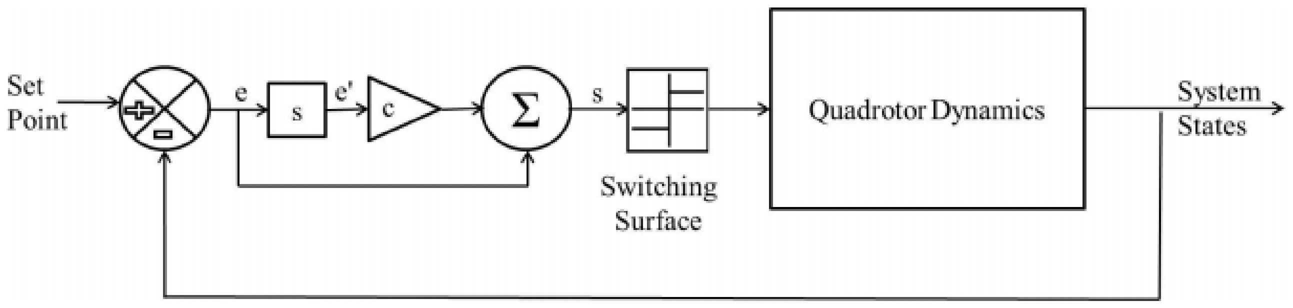


Figure 2-5: SMC Block Diagram

2.3.2 Attitude Control

Based on what we have discussed in the previous section, Sliding Mode controller for attitude has been developed.

2.3.2.1 Roll Controller

The SMC is used to track a reference trajectory for the roll angle. The error in the roll is defined as:

$$e = \varphi_d - \varphi \tag{2.18}$$

The sliding surface is defined as:

$$s = c_1 e + \dot{e} \tag{2.19}$$

Where c_1 is a constant that has to be greater than zero. The derivative of the sliding surface defined in Equation (2.14) with the substitution of Equation (2.13) is formulated as the following:

$$\begin{aligned} \dot{s} &= c_1 \dot{e} + \ddot{e} \\ &= c_1 \left(\dot{\varphi}_d - \dot{\varphi} \right) + \ddot{\varphi}_d - \ddot{\varphi} \end{aligned} \tag{2.20}$$

A Lyapunov function is then defined to be:

$$V(s) = \frac{1}{2} s^2(e) \tag{2.21}$$

Based on the Lyapunov function, the Sliding Mode controller define as follows:

$$\dot{s} = -k_1 \operatorname{sgn}(s) \tag{2.22}$$

Where

$$\operatorname{sgn}(s) = \begin{cases} -1 & if \ s < 0 \\ 1 & if \ s > 0 \end{cases}$$

And k_1 is design constants. To satisfy the sliding mode condition $s\dot{s} < 0$, limits has to be set on k_1 such as $k_1 > 0$. By equating the reaching law (2.17) to the derivative of the sliding surface in Equation (2.15) and substituting $\ddot{\varphi}$ by its definition from Equation (1.31), the control input U_2 is calculated to be:

$$U_2 = \frac{1}{b_1} \left[k_1 \operatorname{sgn}(s) + c_1 (\dot{\varphi}_d - \dot{\varphi}) + \ddot{\varphi}_d - a_1 \dot{\theta} \dot{\psi} \right] \tag{2.23}$$

2.3.2.2 Pitch Controller

Following exactly the same steps as the roll controller, the control input U_3 responsible of generating the pitch rotation θ is calculated to be:

$$U_3 = \frac{1}{b_2}\left[k_1 \operatorname{sgn}(s) + c_1\left(\dot{\theta}_d - \dot{\theta}\right) + \ddot{\theta}_d - a_2\dot{\varphi}\dot{\psi}\right] \qquad (2.24)$$

2.3.2.3 Yaw Controller

Following the same steps as the roll and pitch controller, the control input U_4 responsible of producing the yaw rotation is calculated to be:

$$U_4 = \frac{1}{b_3}\left[k_1 \operatorname{sgn}(s) + c_1\left(\dot{\psi}_d - \dot{\psi}\right) + \ddot{\psi}_d - a_3\dot{\varphi}\dot{\theta}\right] \qquad (2.25)$$

2.3.3 Altitude control

Following the same steps as the roll, pitch and yaw controller, the control input U_1 responsible of producing the altitude is calculated to be:

$$U_1 = \frac{m}{\cos\varphi\cos\theta}\left[k_1 \operatorname{sgn}(s) + c_1\left(\dot{z}_d - \dot{z}\right) + \ddot{z}_d - g\right] \qquad (2.26)$$

2.4 Feedback Linearization Control

Feedback linearization is an approach to nonlinear control design that has attracted lots of research in recent years. The central idea is to algebraically transform nonlinear systems dynamics into (fully or partly) linear ones, so that linear control techniques can be applied. This differs entirely from conventional (Jacobian) linearization, because feedback linearization is achieved by exact state transformation and feedback, rather than by linear approximations of the dynamics [11].

2.4.1 The different linearization approaches

Most feedback linearization approaches are based on input-output linearization or input-state linearization. In the input-output linearization approach, the objective is to linearize the map between the transformed input v and the actual output y. A controller is then designed for the linearized input-output model. In the input-state linearization approach, the goal is to linearize the map between the transformed inputs and the entire vector of transformed state variables. A linear controller is then synthesized for the linear input-state model. However, this approach may lead to a complex controller design task because the map between the transformed inputs and the original outputs y is generally nonlinear. Feedback linearization produces a linear model by the use of nonlinear coordinate transformations and nonlinear state feedback [7].

In this work, input-output linearization approach have been used in order to avoid the complexity of the input-state linearization approach.

2.4.2 Input-output Linearization of MIMO systems

The purpose of this technique is to transform the nonlinear multivariable system using a linearizing state feedback with input-output decoupling. From there, we can apply the theory of linear systems. So we seek a static state feedback of the form $U(\mathrm{X},\mathrm{V})=\alpha(\mathrm{X})+\beta(\mathrm{X}).\mathrm{V}$ so the input-output behavior and/or properties of the system after feedback are linear and decoupled. The input-output linearization technique of a multivariable systems is applied to nonlinear square plant that can be written in the following form:

$$\begin{aligned} \dot{X} &= f(X) + g(X)U \\ y &= h(X) \end{aligned} \qquad (2.27)$$

Where $X \in \mathbb{R}^n$ state vector, $y \in \mathbb{R}^p$ output vector, $U \in \mathbb{R}^m$ input vector and *f; g; h* smooth nonlinear functions.

Before going any further, let's introduce some mathematical tools that have been used in this approach.

2.4.2.1 The Lie Derivative

Consider $f : \mathbb{R}^n \to \mathbb{R}^n$ a vector field and $h : \mathbb{R}^n \to \mathbb{R}$ a scalar function, the Lie derivative can be introduced as a new scalar function, denoted $L_f h(x)$, giving the derivative of *h*(**x**) in the direction *f*(**x**), as [11]:

$$L_f h = \nabla h \times f = \left[\frac{\partial h}{\partial x_1} \frac{\partial h}{\partial x_2} \cdots \frac{\partial h}{\partial x_n} \right] \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \qquad (2.28)$$

For any order, we have:

$$L_f{}^i h = L_f \left(L_f{}^{i-1} h \right) h \qquad i > 1 \qquad (2.29)$$

2.4.2.2 The relative degree

The relative degree ([$r_1, \cdots, r_p$]) of a non-linear MIMO system is a vector of which each component r_i represents the minimum number of times as necessary to differentiate with respect to time the expression of the corresponding output (y_i) explicitly appear to see at least a component of the entry (u_j) [5]:

$$y_i^{(r_i)} = L_f^{r_i} h_i + \sum_{j=0}^{m} L_{g_j} L_f^{(r_i - 1)} h_i u_j \qquad (2.30)$$

If the total relative degree ($r = r_1 + \cdot \cdot \cdot + r_p$) equals the system degree (n) then the system is linearized by static feedback.

2.4.3 Linearization by static feedback

In some applications, the control objectives can be achieved with a nonlinear static feedback control law of the form:

$$U = \alpha(X) + \beta(X).V \tag{2.31}$$

Where V is an external reference input to be defined later, $\alpha(X) \in \mathbb{R}^4$ and $\beta(X) \in \mathbb{R}^{4\times4}$.

In order to find the expression of the linearization control law $U(X,V)$ which allows to make the linear relation between the input and output, we rewrite expression (2.31) in matrix form [11]:

$$\left[y_1^{(r_1)} \cdots y_p^{(r_p)}\right] = \Delta_0(X) + \Delta(X).U = V \tag{2.32}$$

Where

$$\Delta_0(X) = \left[L_f^{r_1} h_1(X) \cdots L_f^{r_p} h_p(X)\right]^T \tag{2.33}$$

$$\Delta(X) = \begin{bmatrix} L_{g1} L_f^{r_1-1} h_1(X) \cdots L_{gm} L_f^{r_1-1} h_1(X) \\ \vdots \qquad \ddots \qquad \vdots \\ L_{g1} L_f^{r_p-1} h_p(X) \cdots L_{gm} L_f^{r_p-1} h_p(X) \end{bmatrix} \tag{2.34}$$

$\Delta(X)$ is the decoupling matrix.

So the linearizing control law has the form:

$$U = \Delta(X)^{-1}\left(-\Delta_0(X) + V\right) \tag{2.35}$$

By identification the two equations (2.31) and (2.35) the terms of static feedback can be deduced as follow: $\alpha(X) = -\Delta(X)^{-1}.\Delta_0(X)$ and $\beta(X) = \Delta(X)^{-1}$

Substituting (2.35) in (2.27) the equivalent system becomes linear and decoupled in the form:

$$y_i^{(r_i)} = v_i \tag{2.36}$$

2.4.4 Linearization by dynamic compensation

Previous linearization is only applicable under the condition (*r* = *n*), (the decoupling matrix is invertible). The dynamic extension applies to systems such as that condition is not met and the matrix $\Delta(X)$ is singular. This extension is to delay an input by means of an additional states in the system. That is to add an integrator at the input [11]. This operation results on the one hand to increase the number of times it must differentiate an output to see the expression of this new entry appear. On the other hand, the proper degree of the system is increased since it has a more state variables [11].

There is therefore an interest in delaying an input when its expression appears in the derivative of order equal to the relative degree of several outputs. Indeed, the total relative degree will be increased by several units while the proper degree will be augmented. This operation will therefore tend to approaches the system condition (*r* = *n*) [11].

2.4.5 Feedback Linearization control design for the Quadrotor

To linearize the Quadrotor model, we first differentiate the outputs with respect to time:

$$\begin{aligned}
\ddot{y}_1 &= \frac{-U_1}{m}\left(\sin x_1 \sin x_5 + \cos x_1 \cos x_5 \sin x_3\right) \\
\ddot{y}_2 &= \frac{-U_1}{m}\left(\cos x_1 \sin x_5 \sin x_3 - \cos x_5 \sin x_1\right) \\
\ddot{y}_3 &= g - \frac{U_1}{m}\left(\cos x_1 \cos x_3\right) \\
\ddot{y}_4 &= b_3 U_4 + a_3 x_2 x_4
\end{aligned}$$

The derivative of the outputs explicitly depend on inputs U_1 and U_4, so we can deduce the decoupling matrix $\Delta(X)$:

$$\Delta(X)=\begin{pmatrix} \frac{-1}{m}(\sin x_1 \sin x_5+\cos x_1 \cos x_5 \sin x_3) & 0 & 0 & 0 \\ \frac{-1}{m}(\cos x_1 \sin x_5 \sin x_3-\cos x_5 \sin x_1) & 0 & 0 & 0 \\ \frac{1}{m}(\cos x_1 \cos x_3) & 0 & 0 & 0 \\ 0 & 0 & 0 & b_3 \end{pmatrix}$$

It is clear that, the total relative degree ($r=8$) is less than the degree of the system (1.42) (n=12). So the decoupling matrix is singular. To have a non-singular decoupling matrix we need a dynamic compensator.

As the second derivative of the outputs $\ddot{y}_2$ and $\ddot{y}_3$ system involves the input u_1, with inputs u_2 and u_3 acting on the Quadrotor do not appear explicitly, then one is compelled to add an integrators on u_1 (Figure 2-6), so as to delay its action and thus enable the inputs u_2 and u_3 act on the system [11].

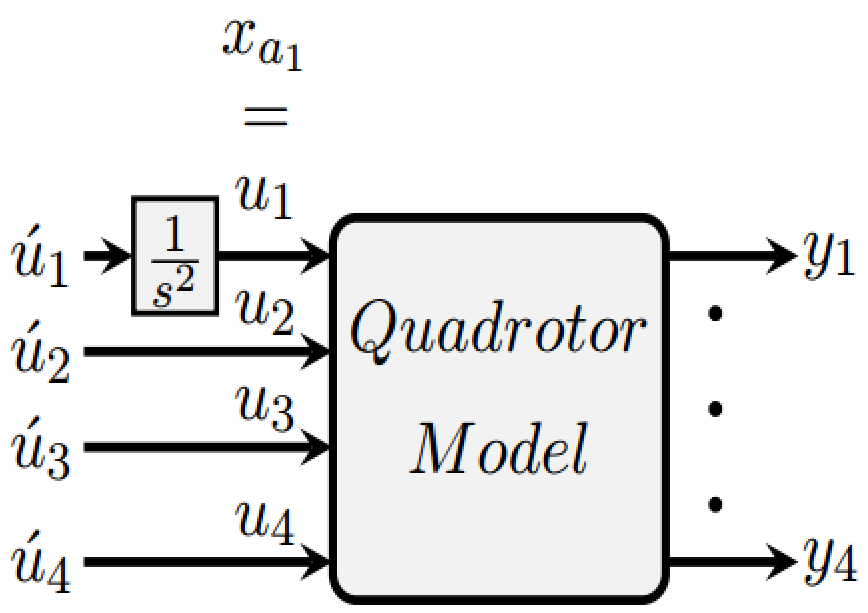


Figure 2-6: The Quadrotor model after dynamic

The real control signals (u_1, u_2, u_3 *and* u_4) have been replaced by (u'_1, u'_2, u'_3 *and* u'_4) to avoid singularity in Lie transformation matrices when using exact linearization. In that case u_1 has been delayed by double integrator. The other control signals will remain unchanged.

$$\begin{cases} u'_1 = \dot{x}_{a2} \\ \dot{x}_{a1} = x_{a2} \\ u_1 = x_{a1} \\ u'_2 = u_2 \\ u'_3 = u_3 \\ u'_4 = u_4 \end{cases} \qquad (2.37)$$

(u'_1, u'_2, u'_3 *and* u'_4) are the new entries after the dynamic compensation of the system, and (x_{a1}, x_{a2}) are auxiliary states. The state space representation of Quadrotor after the dynamic compensation is of the form:

$$\frac{d(\mathrm{X}')}{dt} = f(\mathrm{X}') + g(\mathrm{X}')\mathrm{U}' \qquad (2.38)$$

Where

$$f(X')=\begin{bmatrix} x_2 \\ a_1x_4x_6 \\ x_4 \\ a_2x_2x_6 \\ x_6 \\ a_3x_2x_4 \\ x_8 \\ g-\frac{x_{a1}}{m}(\cos x_1\cos x_3) \\ x_{10} \\ \frac{-x_{a1}}{m}(\sin x_1\sin x_5+\cos x_1\cos x_5\sin x_3) \\ x_{12} \\ \frac{-x_{a1}}{m}(\cos x_1\sin x_5\sin x_3-\cos x_5\sin x_1) \\ x_{a2} \\ 0 \end{bmatrix}, \quad g(X')=\begin{bmatrix} 0&0&0&0 \\ 0&b_1&0&0 \\ 0&0&0&0 \\ 0&0&b_2&0 \\ 0&0&0&0 \\ 0&0&0&b_3 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 1&0&0&0 \end{bmatrix}$$

$$\mathrm{U}'=[u_1',u_2',u_3',u_4']^T$$
$$\mathrm{X}'=\left[\mathrm{X}^T,x_{a1},x_{a2}\right]^T$$
$$y=[y_1,y_2,y_3,y_4]^T=[x_9,x_{11},x_7,x_5]^T$$

The input-output decoupling problem is solvable for the nonlinear system (2.27) by means of a dynamic feedback control law if it is solvable via a static feedback for the extended system (2.38). For the nonlinear system, the relative degree vector $[r_1,r_2,r_3,r_4]$ is given by:

$$r_1=r_2=r_3=4, \qquad r_4=2$$

And we have

$$\begin{bmatrix} y_1^{(r_1)} & y_2^{(r_2)} & y_3^{(r_3)} & y_4^{(r_4)} \end{bmatrix}^T=\Delta_0(\mathrm{X}')+\Delta(\mathrm{X}')\mathrm{U} \qquad (2.39)$$

Passing again through the steps of static feedback control design we can deduce the decoupling matrix $\Delta(\mathrm{X}')$ and the vector $\Delta_0(\mathrm{X}')$.The matrix $\Delta(\mathrm{X}')$ is nonsingular at any point characterized by

$x_{a1} \neq 0, -\frac{\pi}{2} < \varphi < \frac{\pi}{2}, -\frac{\pi}{2} < \theta < \frac{\pi}{2}$. Therefore, the input-output decoupling problem is solvable for the system (2.27) by means of a dynamic feedback control law of the form:

$$\mathrm{U}' = \alpha(\mathrm{X}') + \beta(\mathrm{X}').\mathrm{V} \tag{2.40}$$

The figure 2-7 shows the structure for the control law of the original system (2.27).

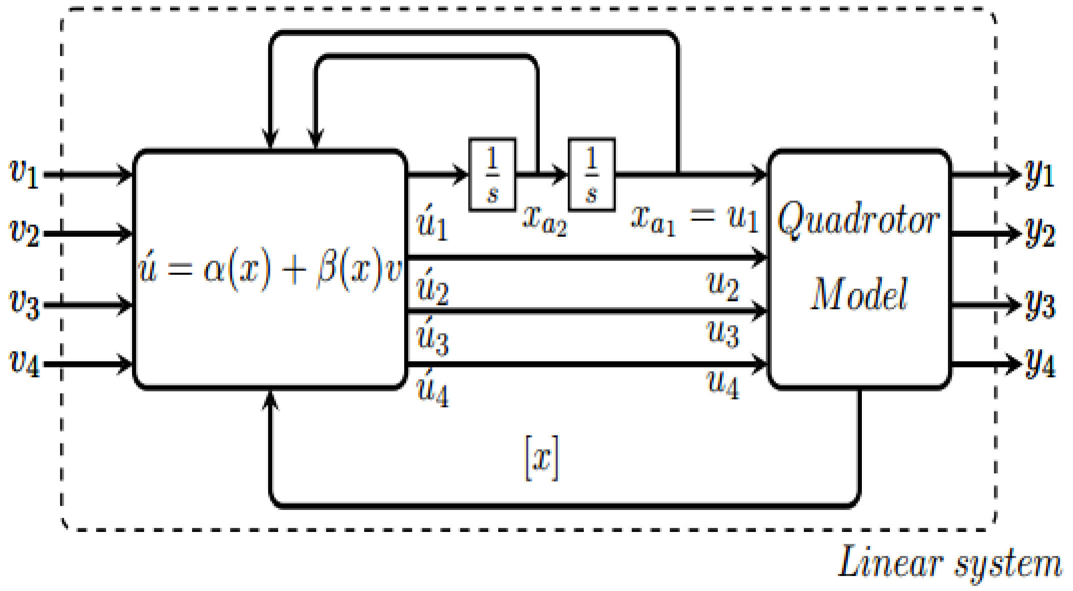


Figure 2-7: Block diagram of the control law.

Since the extended system (2.38) has dimension n=14 and the condition $r_1 + r_2 + r_3 + r_4 = n$ is hold, now, the system can be transformed via a dynamic feedback into a system which is fully linear and controllable. The change of coordinates $z = \Phi(\mathrm{X}')$ is given by:

$$\begin{cases} z_1 = h_1(\mathrm{X}) = x \\ z_2 = L_f h_1(\mathrm{X}) = \dot{x} \\ z_3 = L_f^2 h_1(\mathrm{X}) = \ddot{x} \\ z_4 = L_f^3 h_1(\mathrm{X}) = x^{(3)} \\ z_5 = h_2(\mathrm{X}) = y \\ z_6 = L_f h_2(\mathrm{X}) = \dot{y} \\ z_7 = L_f^2 h_2(\mathrm{X}) = \ddot{y} \\ z_8 = L_f^3 h_2(\mathrm{X}) = y^{(3)} \\ z_9 = h_3(\mathrm{X}) = z \\ z_{10} = L_f h_3(\mathrm{X}) = \dot{z} \\ z_{11} = L_f^2 h_3(\mathrm{X}) = \ddot{z} \\ z_{12} = L_f^3 h_3(\mathrm{X}) = z^{(3)} \\ z_{13} = h_4(\mathrm{X}) = \psi \\ z_{14} = L_f h_4(\mathrm{X}) = \dot{\psi} \end{cases} \tag{2.41}$$

The linear system with new coordinates is given as follow:

$$\begin{cases} \dot{z} = \mathrm{A} \cdot z + \mathrm{B} \cdot \mathrm{v} \\ y = \mathrm{C} \cdot \mathrm{z} \end{cases} \tag{2.42}$$

Where

$$z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 & z_9 & z_{10} & z_{11} & z_{12} & z_{13} & z_{14} \end{bmatrix}^T \in \mathbb{R}^{14}$$
$$\mathrm{V} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}^T \in \mathbb{R}^{14}$$

$$\mathrm{A} = \begin{bmatrix} \mathrm{A}_1 & 0 & 0 & 0 \\ 0 & \mathrm{A}_1 & 0 & 0 \\ 0 & 0 & \mathrm{A}_1 & 0 \\ 0 & 0 & 0 & \mathrm{A}_2 \end{bmatrix} \in \mathbb{R}^{14\times14}, \quad \mathrm{B} = \begin{bmatrix} \mathrm{B}_1 \\ \mathrm{B}_2 \\ \mathrm{B}_3 \\ \mathrm{B}_4 \end{bmatrix} \in \mathbb{R}^{14\times4}, \quad \mathrm{C} = \begin{bmatrix} c_1^T & 0 & 0 & 0 \\ 0 & c_1^T & 0 & 0 \\ 0 & 0 & c_1^T & 0 \\ 0 & 0 & 0 & c_2^T \end{bmatrix} \in \mathbb{R}^{4\times14}$$

With

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$B_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad B_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad c_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T, \quad c_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$$

The figure 2-8 shows the linear relation between the input v and the output y.

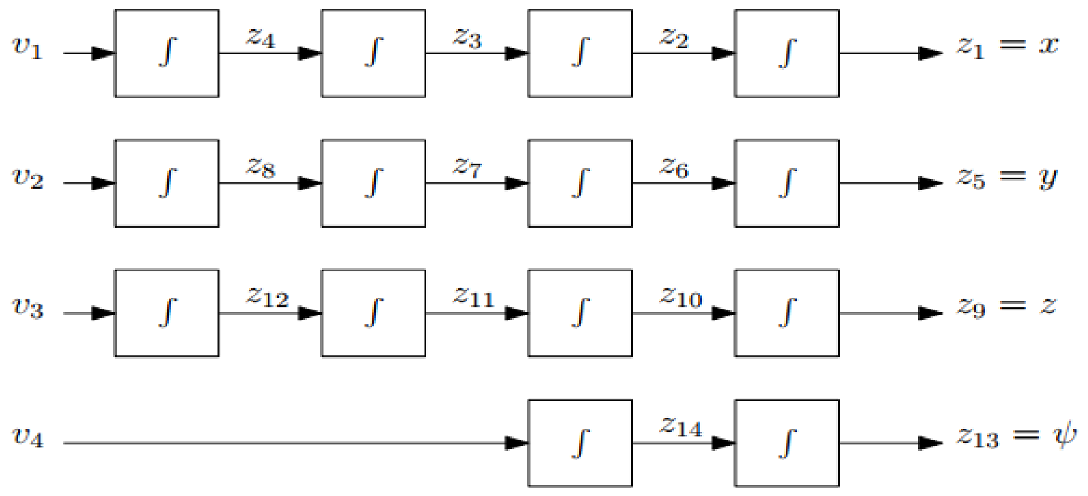


Figure 2-8: The relation between the input v and the output y.

Note that the resulting system (2.42) with the new controls (**v**) is linear, in addition to that is decoupled. So it is easy to control the system using conventional linear techniques. In our case, pole placement technique has been used. The results are shown in chapter 4.

2.5 Linear Quadratic Regulator control

The objective of the optimal control is to determine control signal so that the system to be controlled can meet physical constraints and minimize/maximize a cost/performance function. Namely, the solution of an optimization problem is supposed to bring the system's state $\mathrm{X}(t)$ to the desired trajectory X_d minimizing some cost. Furthermore, it minimizes the use of the control inputs, thus reducing the use of actuators[7].

Considering the continuous linear system shown in Equation (2.43), a cost function can be defined as Equation (2.44).

$$\begin{cases} \dot{\mathrm{X}} = \mathrm{A}\,\mathrm{X} + \mathrm{B}\,\mathrm{U} \\ y = \mathrm{C}\,\mathrm{X} \end{cases} \tag{2.43}$$

$$J = \int_{t_0}^{\infty} \left[\mathrm{U}(t)^T . \mathrm{R} . \mathrm{U}(t) + \left(\mathrm{X}(t) - \mathrm{X}_d(t)\right)^T . \mathrm{Q} . \left(\mathrm{X}(t) - \mathrm{X}_d(t)\right) \right] dt \tag{2.44}$$

Where

R is the cost of actuators ($\mathrm{R} = \mathrm{R}^T$ positive definite matrix; $\mathrm{R} \in \mathbb{R}^{m\times m}$).

Q is the cost of the state ($\mathrm{Q} = \mathrm{Q}^T$ positive semi-definite matrix; $\mathrm{Q} \in \mathbb{R}^{n\times n}$).

It is possible to find the control input $\mathrm{U}(t)$ which minimizes the cost function formed as:

$$\mathrm{U}(t) = -\mathrm{K}.\left[\mathrm{X}(t) - \mathrm{X}_d(t)\right] \tag{2.45}$$

Where

$$\mathrm{K} = \mathrm{R}^{-1}.\mathrm{B}^T.\mathrm{P} \tag{2.46}$$

The P matrix is a solution of the Riccati's algebraic equation:

$$\mathrm{P}.\mathrm{A} + \mathrm{A}^T.\mathrm{P} - \mathrm{P}.\mathrm{B}\,\mathrm{R}^{-1}.\mathrm{B}^T.\mathrm{P} + \mathrm{C}.\mathrm{Q}.\mathrm{C} = 0 \tag{2.47}$$

Where P is a positive definite matrix. The Figure 2-9 shows a scheme of the implemented system:

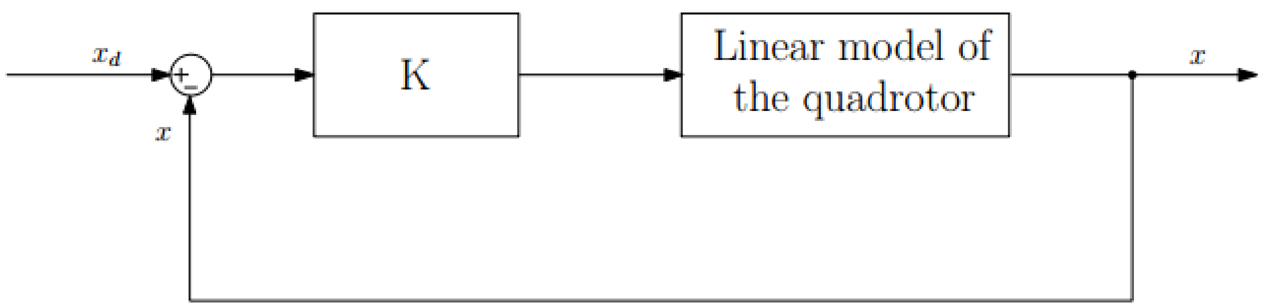


Figure 2-9: LQR control.

The algebraic equation can be solved through Riccati's method, performed by Matlab through LQR function:

$$K = LQR(A, B, Q, R).$$

We consider the linear system (1.51). We choose the matrices Q and R taking into account A (1.49) and B (1.50). We apply the LQR control using the LQR function from Matlab/Simulink.

Chapter 3

Autonomous navigation

3.1 SLAM System

Simultaneous localization and mapping (SLAM) is the problem of concurrently estimating in real time the structure of the surrounding world (the map), while simultaneously localizing the robot in it, and it is the major component to build any system capable of autonomous navigation. It uses sensor data to build a map of the environment incrementally and estimates the position of the robot at the same time.

There are two categories of sensors that can be used with SLAM systems

- Non-visual :
  - Ultrasonic range sensors
  - High-resolution laser range scanner: very powerful and accurate providing a full depth images, however it's so expensive.
- Visual :
  - RGBD and stereo cameras: provide depth data of the environment can greatly simplify the process and reduce the computational cost but they are not suitable for drones because of their size, weight, high power consumption and limitation of range.
  - Monocular camera: it's available with the drone so no additional sensors are required, however it does not provide depth information.

On this thesis we are going to use Visual monocular SLAM system where the map of the environment is typically represented by a number of landmarks, which are points in three-dimensional space that can be recognized and Localized in the camera image, typically appearing as small, distinctive regions or patches (keypoints). The position of the camera is estimated Based on the locations of these keypoints in the image. As new parts of the environment become visible, additional landmarks are identified, added to the map and can then be integrated into the pose- estimation process [12].

3.1.1 Monocular, Keyframe-Based SLAM Algorithm Outline

Monocular SLAM algorithms (PTAM for this thesis) are based on two distinct parts, running independently in parallel threads: tracking and mapping. A separate initialization procedure is needed to generate an initial map for the system to launch.

• **Initialization** is performed once after the algorithm is started and requires a certain type of camera-motion.

input: initial video frames

output: initial map

• **Mapping** runs continuously to optimize the map and integrate new keyframes when instructed to by the tracking component.

input: the old map, new keyframes.

output: updated map

• **Tracking** continuously evaluating the camera pose for each new video frame.

input: new video frame, landmark positions.

output: the camera pose

3.1.2 Keypoints generation

Keypoints or local feature points are widely used in many algorithms of computer vision field, the idea behind them is that processing the image as a whole is computationally unfeasible, instead a small set of particularly "interesting" and distinguishable image segments is used for tasks such as object recognition, detection and tracking, pose estimation and many more. Keypoints mainly occur on strongly textured objects, a keypoint is a two-dimensional location in the camera image, while the corresponding three-dimensional position will be referred to as a landmark. Keypoints are hence the two-dimensional projections of landmarks onto the image plane [12].

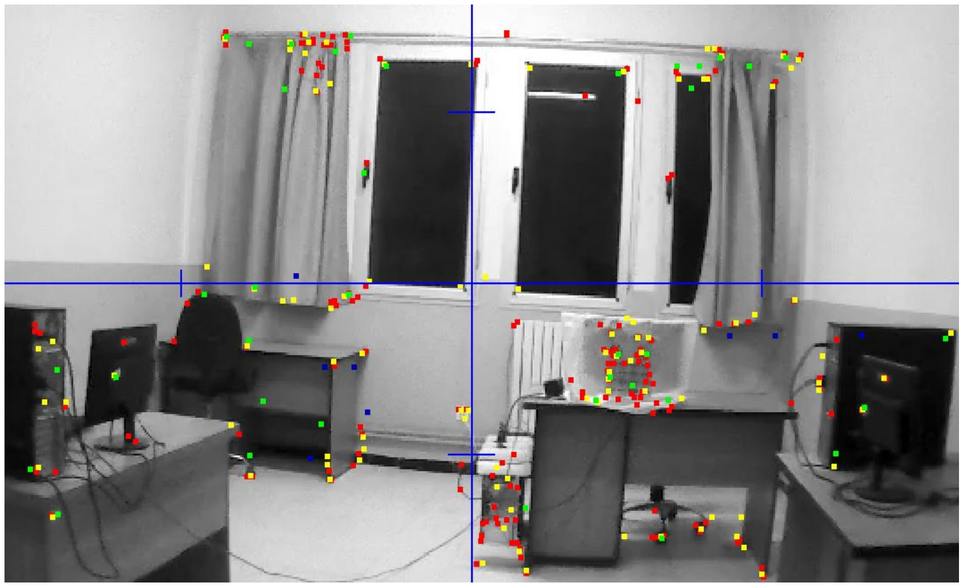


Figure 3-1: Keypoints generated by PTAM system

3.1.3 The FAST Corner Detector

SLAM system presented on this thesis uses FAST (Features from Accelerated Segment Test) a keypoint detector which was presented by Rosten and Drummond in 2006 [13], As the acronym already tells the FAST corner detector has the advantage of being significantly faster than other methods, achieving a speedup of factor 20 to 30 [12], compared to Harris corner detector [14], making it the method of choice for time-critical applications such as real-time SLAM systems

The key idea behind the segment test criterion is to consider a circle of sixteen pixels around the corner candidate P, and classifies P as a corner if there exists a set of n contiguous pixels in the circle which are all brighter than the intensity of the candidate pixel Ip plus a threshold t, or all darker than Ip - t, as illustrated in Figure 3.2, n was chosen to be twelve because it admits a high-speed test which can be used to exclude a very large number of non-corners: the test examines only the four pixels at 1, 5, 9 and 13 (the four compass directions). If p is a corner then at least three of these must all be brighter than Ip + t or darker than Ip - t. If neither of these is the case, then p cannot be a corner. The full segment test criterion can then be applied to the remaining candidates by examining all pixels in the circle,then creates a decision tree which can classify all corners, this decision tree is then converted into C-code, creating a long string of nested if-then-else statements which is compiled and used as a corner detector [13].

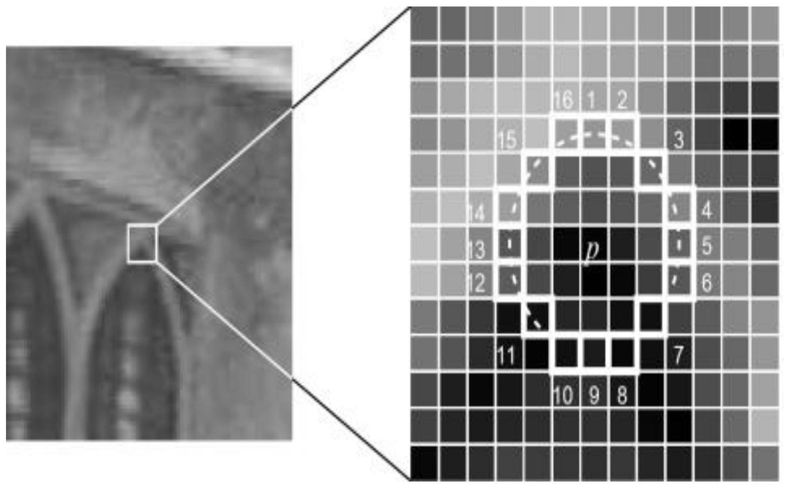


Figure 3-2: 12 point segment test corner detection in an Image patch. P is the center of a candidate corner.

3.1.4 Tracking a Keypoint

Tracking a keypoint is the task of finding the exact position (and possibly other parameters such as scale and orientation) of a keypoint in an image. Assuming that the displacement between two consecutive frames is small. A general formulation of tracking is to find parameters **p** of a warp function $f(\mathrm{x}, \mathrm{y}; \mathrm{p}) : \mathbb{R}^2 \times \mathbb{R}^d \rightarrow \mathbb{R}^2$ such that the difference between the original patch $T(x, y)$ and the transformed image $I(f(x, y; \mathbf{p}))$ becomes minimal, that is minimizing the sum of squared differences[12] (SSD) :

$$p^* = \arg\min E_{SSD}(\mathrm{p}) \tag{3.1}$$

with

$$E_{SSD}(p) := \sum_{x,y} (\mathrm{I}(f(\mathrm{x}, \mathrm{y}; \mathrm{p})) - \mathrm{T}(\mathrm{x}, \mathrm{y}))^2 \tag{3.2}$$

The warp function $f(x, y; \mathbf{p})$ can take different forms, for tracking a two-dimensional image patch two important warp functions are:

Pure Translation: It is often sufficient to consider only translation for frame-to-frame tracking. The resulting transformation has two degrees of freedom, the displacement in two dimensions.

$$f(x,y;\delta x,\delta y)=\begin{pmatrix} x+\delta x \\ y+\delta y \end{pmatrix} \tag{3.3}$$

Which is used for the PTAM system used on this thesis.

Affine Transformation: An affine transformation allows for displacement, non-uniform scaling and rotation, leading to 6 degrees of freedom.

$$f(x,y;\mathrm{p})=\begin{pmatrix} p_1 & p_2 \\ p_3 & p_4 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}+\begin{pmatrix} p_5 \\ p_6 \end{pmatrix} \tag{3.4}$$

3.1.5 Initialization

The difficulty in Visual SLAM systems comes from the fact that to build a map it is required to be able to track the camera position, which in turn requires the existence of a map. This is due to absence of the depth information, this issue does not exist with stereo-SLAM or RGBD-SLAM where the initial map can be built simply from the first image, the approach used to solve this is to apply a separate initialization procedure which can be summarized as follow:

1. Analyze the first keyframe *K*1 and detect promising keypoints **p**1 . . . **p***n* using FAST corner detector.
2. Track keypoints using a simple frame-to-frame tracking approach as described in the previous Section.
3. Extract new keypoint positions **p’**1 . . . **p**’n from the second keyframe *K*2.
4. Generate the initial map from these point-correspondences.

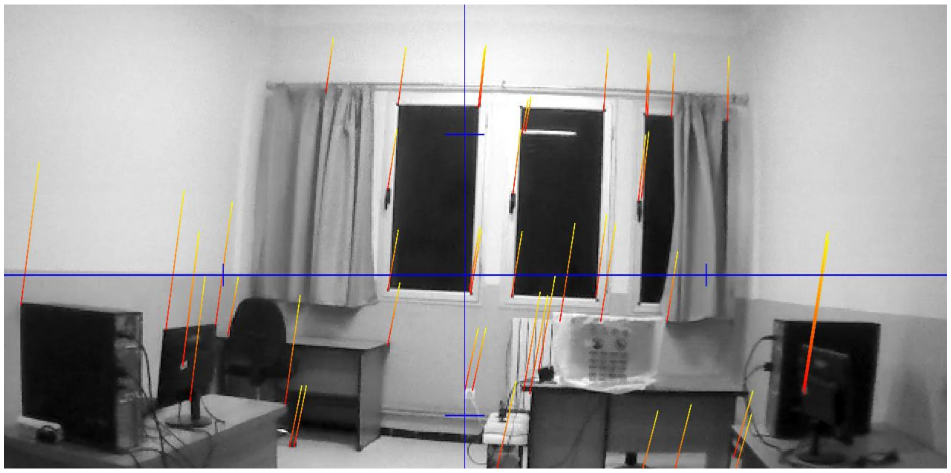


Figure 3-3: Initialization procedure of PTAM. Every line corresponds to a successfully tracked keypoint.

3.1.6 Mapping

The mapping loop continuously optimizes the map and extends it by incorporating new keyframes and landmarks.

Map Optimization

Given all observations p_{ij}, the goal is to refine the keyframe and landmark positions K_j and x_i, such that they best coincide with these observations. The solution to this problem is obtained by minimizing the **total reprojection error** *E*rep: Let the reprojection error of a single observation **p** of landmark x_w, and from a camera-position *C* be defined as [12]

$$e = (p, x_w, c) := \bar{p} - proj(\mathrm{K}_{cam}\, proj(Ec\,\tilde{x}_w)) \tag{3.5}$$

$\tilde{x}_w$: The homogeneous representation of a point x_w.
Kcam: camera projection matrix.
$\bar{p}$ **:** Pixel-coordinates of an observation of landmark.

Which corresponds to the distance in pixel between the location where the landmark actually was observed and its projection onto the image. The total reprojection error is now given by:

$$E_{rep}(x_1 \ldots x_n) := \sum_{\substack{j=1\ldots m \\ i \in \varsigma_j}} Obj\left(\frac{\left\| e_{ij} \right\|^2}{\sigma_{ij}^2}\right) \tag{3.6}$$

where Obj: $\mathbb{R} \rightarrow \mathbb{R}$ is a robust kernel function and ζ_j the set of indices of all landmarks observed in keyframe *j*. Minimizing this error function, using an iterative method is referred to as global bundle adjustment (BA).

For a growing number of landmarks and keyframes, optimizing this error function as a whole each time a new keyframe or landmark is added quickly becomes computationally unfeasible. This gives rise to the concept of local bundle adjustments. Optimization of (3.6) is performed by only considering a small subset of keyframes and a corresponding subset of landmarks, keeping everything else fixed, after adding a new keyframe, optimizing only over the most recently added keyframes and a corresponding set of landmarks may be sufficient [12].

3.1.7 Tracking

The tracking loop is executed once for each new video-frame *I*, and calculates the corresponding camera position *C*, based on the known landmark positions $\mathbf{x}_1 \dots \mathbf{x}_n$. It requires an initial guess of the camera pose C_0, for example the pose in the previous frame [12].

Pose Estimation

First, all potentially visible landmarks are projected into the image based on the expected camera position C_0, and for each such landmark, a warped template of its expected appearance is generated from a keyframe it was observed in. Using a tracking approach, the exact location of the landmark in the image is then computed to subpixel accuracy. The result of this stage is a set of *k* 3D-to-2D point correspondences, $\mathbf{x}_1 \dots \mathbf{x}_k$ and $\mathbf{p}_1 \dots \mathbf{p}_k$.

Based on these 3D-to-2D point correspondences, the camera position *C* is to be estimated. This is called the perspective *n*-point (P*n*P) problem, it is a known problem in computer vision and robotics. There are various ways to solve this problem, including iterative and non-iterative methods, a good overview is given in [15]. For a SLAM system, it can generally be assumed that a good initialization is available as the camera movement is small in between two frames - applying a gradient-descend based, iterative method, minimizing the reprojection error as defined in the previous Sections Therefore the preferred method[12]:

$$C^* = \arg\min \sum_{i=1}^{k} Obj\left(\frac{\left\|e(\mathrm{p}_i, \mathrm{x}_i, C)\right\|^2}{\sigma_i^2}\right) \tag{3.7}$$

The tracking part also decides if a frame will be added as new keyframe based on heuristic criteria such as:

- Tracking quality is good (a high fraction of landmarks has been found).
- No keyframe was added for some time.
- No keyframe was taken from a point close to the current camera position.

3.2 K-Means Clustering

k-means clustering is an iterative, data-partitioning algorithm that assigns n observations to exactly one of k clusters defined by centroids, where k is chosen before the algorithm starts. The objective of K-Means clustering is to minimize the "within-cluster sum of squares".

Inputs :

- K: the number of clusters.
- X: $n \times p$ data set where n is number of points, and p is the number of features.

Outputs :

- C : $k \times p$ matrix of cluster centroid locations

At the initialization phase the algorithm gives an initial estimates for the K centroids which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

- **Data assignment step**

  Each centroid defines one of the clusters. In this step, each data point is assigned to Its nearest centroid, based on the squared Euclidean distance. More formally, if c_i is the collection of centroids in set C, then each data point x is assigned to a cluster based on

$$\operatorname{argmin} dist(c_i, x)^2 \quad c_i \in C \tag{3.8}$$

  Let the set of data point assignments for each i^{th} cluster centroid be S_i.

- **Centroid update step**

  The centroids are then recomputed by taking the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i \tag{3.9}$$

The algorithm iterates between these steps until the centroids are no longer changing or some maximum number of iterations is reached.This algorithm is guaranteed to converge to a result, the result may be a local optimum.

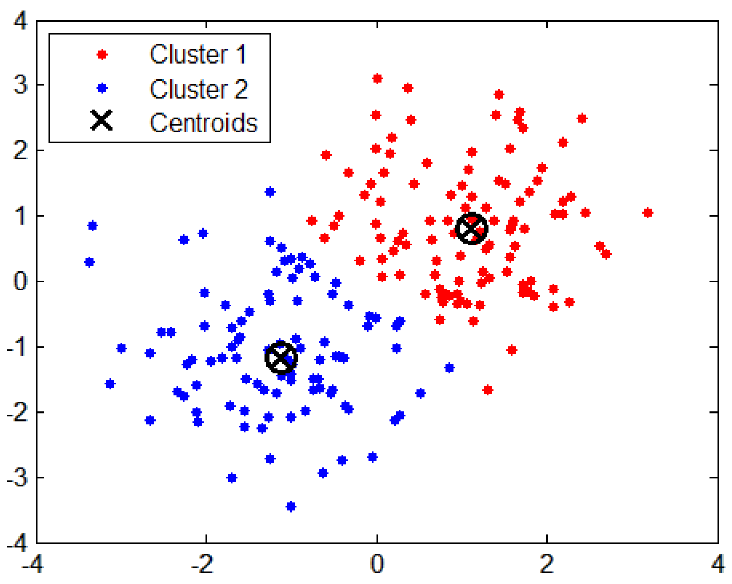


Figure 3-4: K-means clustering on a data set

3.3 Occupancy Grids

After the obstacle detection phase through clustering methods the next step is to create a visual map that represents the robot all along with the obstacles in the surrounding environment**,** which enables us to perform path planning to find collision-free paths.

Occupancy grids are a represent of the robot workspace as a matrix of grids with specified resolution Depending On the quality of the sensor data used to create the map. Each cell gives information weather its location is occupied with an obstacle.

There are two types of occupancy grids :

- Binary occupancy grid: cells with occupied workspace (obstacles) hold true values, Whereas cells with false values represent the free workspace. This grid shows where obstacles are and whether a robot can move through that space. It is a light map and does not require lot of memory.
- Probability occupancy grid: uses probability values to create a more detailed map representation. Each cell in the occupancy grid has a value representing the probability of the occupancy of that cell.Values close to 1 represent a high certainty that the cell contains an obstacle. Values close to 0 represent certainty that the cell is not occupied and obstacle free. The probabilistic values can give better fidelity of objects and improve performance of certain algorithm applications.

The two coordinate systems supported are world and grid coordinates.

- **world coordinates:** which is used most in robotics applications, the origin is the bottom-left corner of the map
- **grid coordinates:** the first grid location with index (1,1) begins in the top-left corner of the grid.

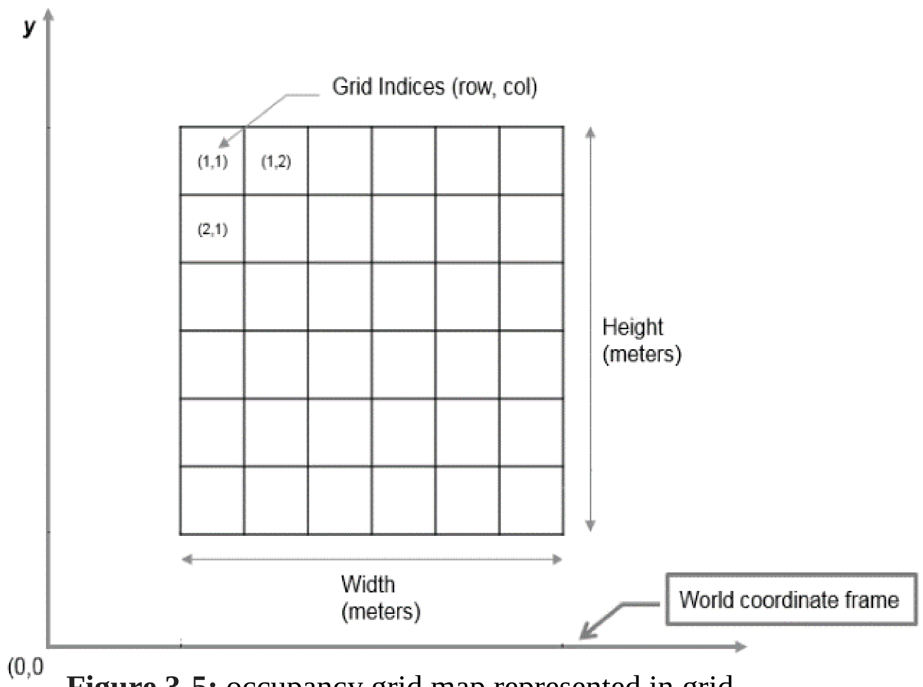


Figure 3-5: occupancy grid map represented in grid coordinates starting from top left corner and world coordinates starting from bottom left corner.

3.4 Probabilistic roadmap

with an occupancy grid map in hands the next step is to generate a collisions-free path for our robot, path planning has been the area of interest of many researches throughout the years and this yield in many algorithms and approaches such as the Rapidly-Exploring Random Trees algorithm, the Probabilistic Roadmap (PRM) method and the Artificial Potential Field (APF). PRM approach was used in this thesis to realize the path planning of the robot.

The basic idea behind PRM is to take random samples from the configuration space of the robot, testing them for whether they are in the free space, then use a local planner to connect these configurations to other nearby configurations. The starting and goal configurations are added in, and a graph search algorithm is applied to the resulting graph to determine a path between the starting and goal configurations.

The framework of PRM planning algorithm consists of two phases

- **roadmap construction (learning) phase**

  On this phase a roadmap data structure is constructed in a probabilistic way by repeatedly generating random free configurations of the robot and connecting these configurations using a local planner, the roadmap formed is stored as an undirected graph R = (N, E).The nodes in N are a set of configurations of the robot appropriately chosen over the free C-space. The edges in E correspond to (simple) paths.

  in more details at the beginning of the learning algorithm the graph R is empty, Then repeatedly adding a random free configuration to N, For every new node c a set N_c of candidate neighbors is chosen from N based on a distance function, Then we pick nodes from N, in order of increasing distance from c. We try to connect c to each of the selected nodes. whenever this planner succeeds to compute a feasible path between c and a selected node n, the edge (c, n) is added to E, then an expansion step is performed to improve the connectivity of the graph R [16].

- **query phase**

  In this phase we use the roadmap (graph R) constructed in the learning phase to find the shortest path connecting two desired configurations namely a starting configuration s and a goal configuration g. The first step to achieve this is to connect s and g to some two nodes s' and g' in N with feasible paths P_s, and P_g, If successful, it then searches *R* for a sequence of edges in *E* connecting s' to g' , A feasible path from s to g is eventually constructed by concatenating P_s and P_g.

  Another question to answer is how to compute the paths P_s, and P_g using no expensive algorithms. The strategy for connecting s to *R* is to consider the nodes in *R* in order of increasing distance from s according to a distance function and try to connect s to each of them with the local planner, until one connection succeeds.

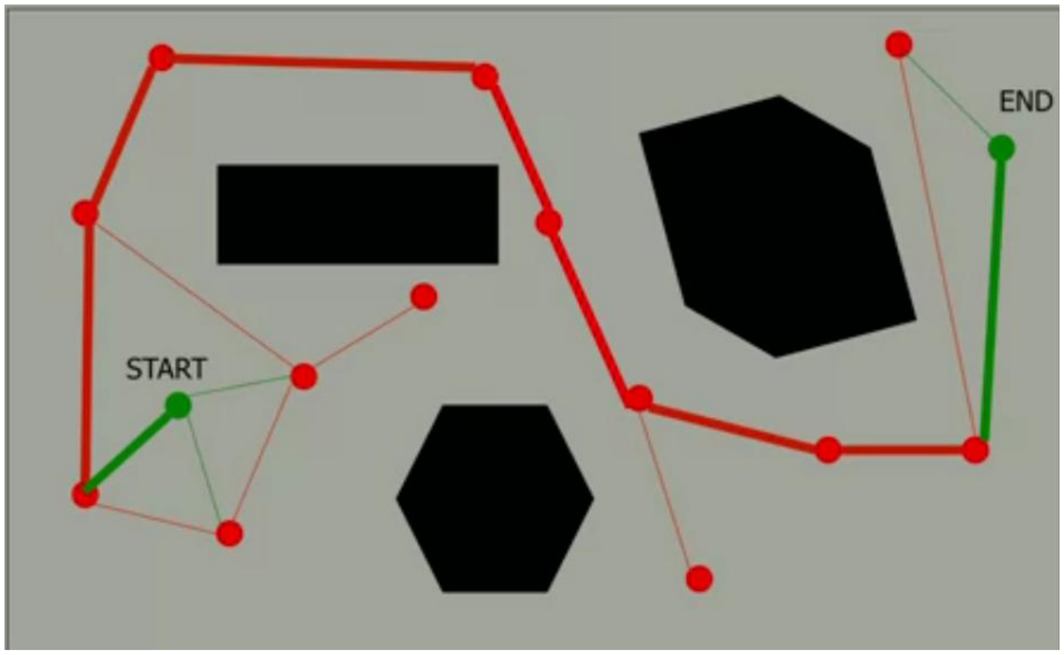


Figure 3-6: a path generated by PRM algorithm in bold red, P_S and P_g in bold green, and local path segments in red

Chapter 4

Implementation and results

4.1 Controller's simulation and results

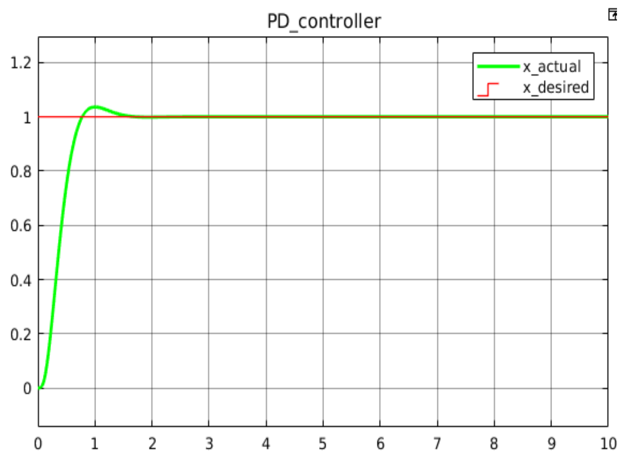
In this section, we show the results obtained in MATLAB/Simulink and we discuss the differences between the several controllers illustrated above. For each control, we show the step response of the output variable x, y, z, ψ . Table 4.1 presents the parameters of our real model ardrone2.0 used in the simulation.

Parameter	**Value**	**Units**
Body Mass	103.8	g
Battery Mass	119.2	g
Case Mass	62.0	g
Engine Mass	37.8	g
Ixx	4.50	gm^2
Iyy	5.10	gm^2
Izz	9.50	gm^2

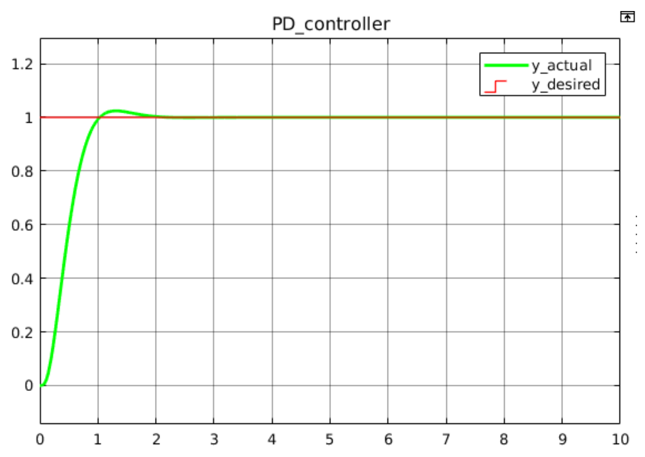
Table 4.1: Model parameters (AR drone2.0).

4.1.1 PD control results

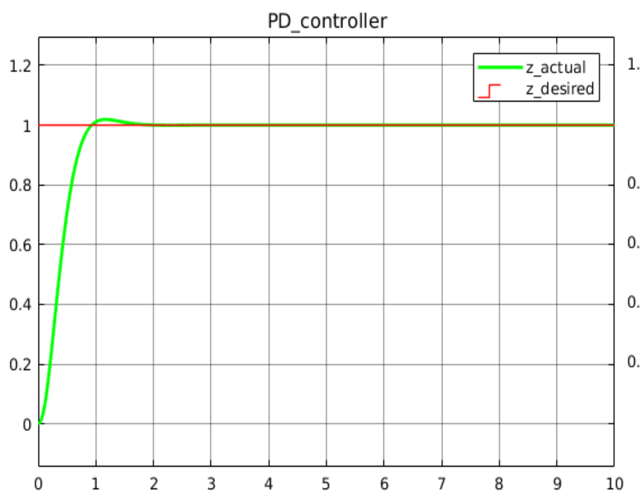
Figure 4.1 shows the step response of the output variables *x; y; z;* when the PD controller is used



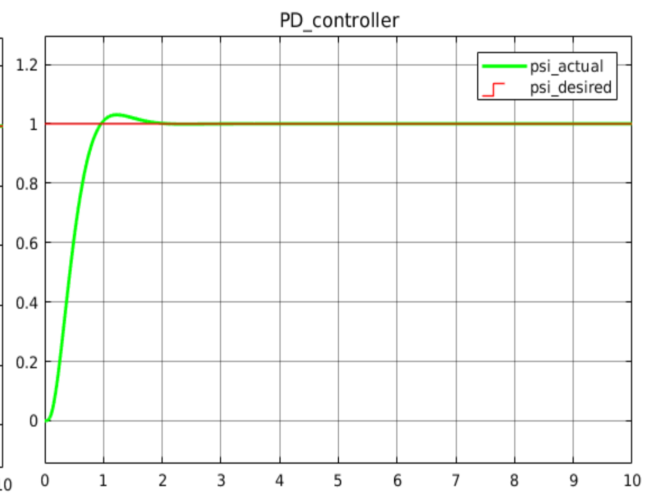
(a)



(b)



(c)



(d)

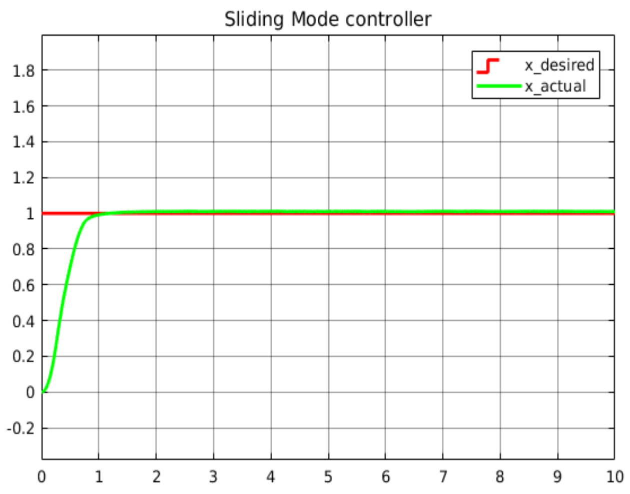
Figure 4.1: step response (PD controller) of (a): $x(t)$, (b): $y(t)$, (c): $z(t)$, (d): $\psi(t)$

	$x(t)$	$y(t)$	$z(t)$	$\psi(t)$
Rise time [s]	0.453	0.595	0.548	0.567
Overshoot [%]	3.646	2.597	1.531	2.577
Settling time [s]	0.702	0.856	0.770	0.837

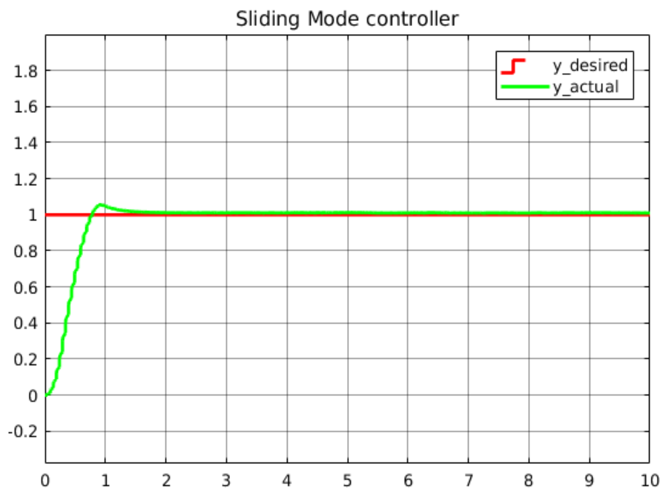
Table 4.2: Characteristic parameters to a step input.

4.1.2 Sliding Mode control results

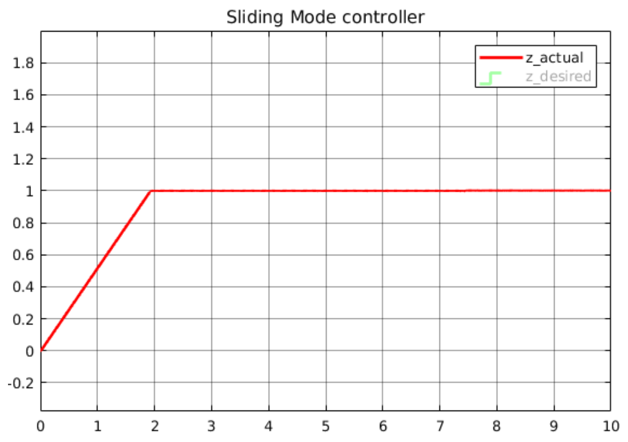
Figure 4.2 shows the step response of the output variables *x; y; z;* when the Sliding Mode controller is used.



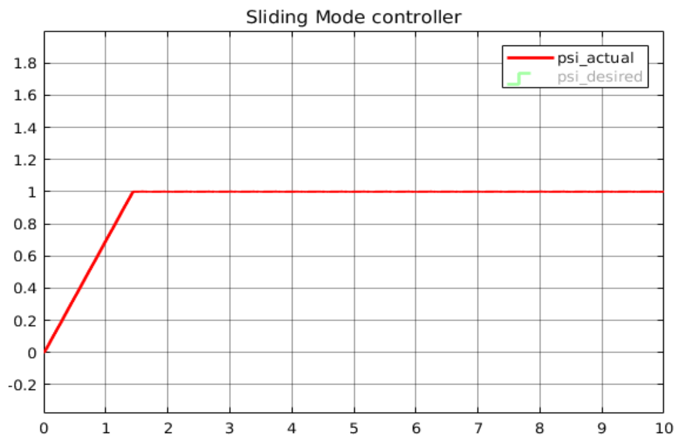
(a)



(b)



(c)



(d)

Figure 4.2: step response of Sliding Mode controller (a): $x(t)$, (b): $y(t)$, (c): $z(t)$, (d): $\psi(t)$

Table 4.3 shows some characteristic parameters of the step response when Sliding Mode controller is used.

	$x(t)$	$y(t)$	$z(t)$	$\psi(t)$
Rise time [s]	0.501	0.487	1.462	1.121
Overshoot [%]	0	2.43	0	0
Settling time [s]	0.751	0.977	1.746	1.463

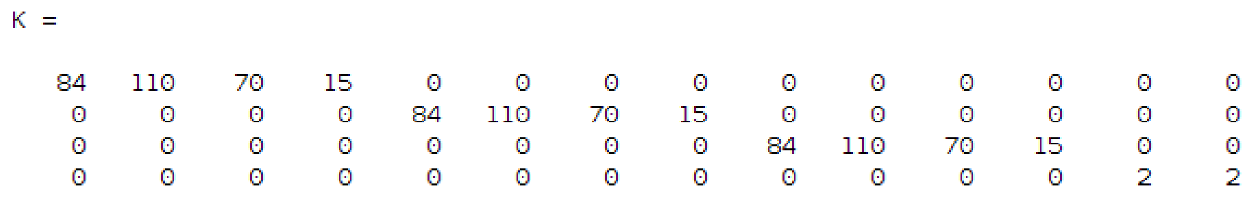
Table 4.3: Characteristic parameters to a step input.

4.1.3 Feedback Linearization with Pole Placement control results

After the system (1.43) has been linearized using Feedback Linearization technique, the obtained system was not stable with degree 14, so the pole placement technique has been used to stabilize the system, and the following desired poles have been chosen to improve the system response:

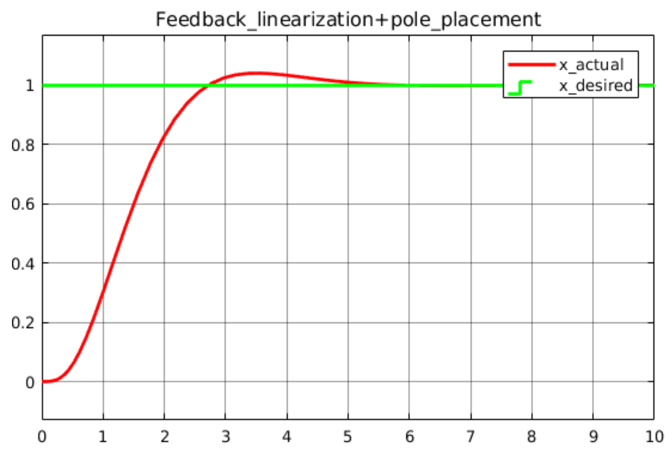
[-1-i -1+i -6 -7 -1-i -1+i -6 -7 -1-i -1+i -6 -7 -1-i -1+i]

With control gain matrix:

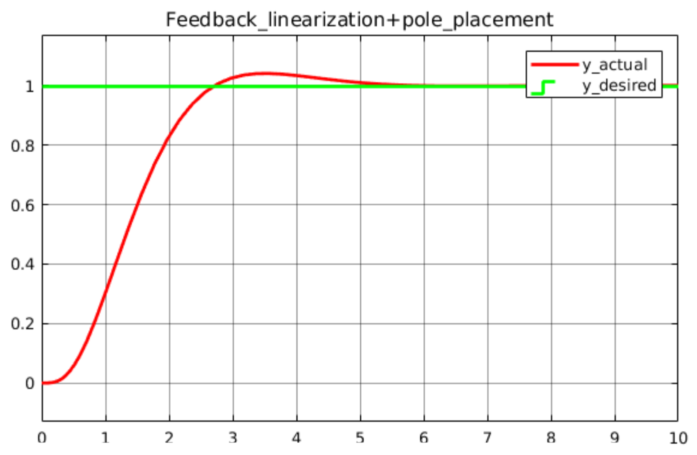


```
K =

    84   110    70    15     0     0     0     0     0     0     0     0     0     0
     0     0     0     0    84   110    70    15     0     0     0     0     0     0
     0     0     0     0     0     0     0     0    84   110    70    15     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     2     2
```

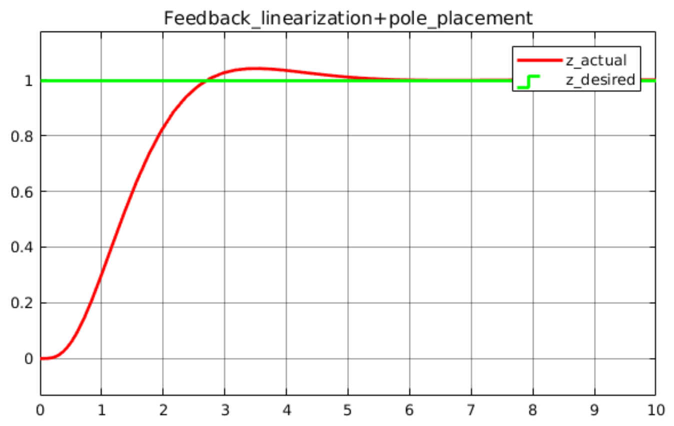
Figure 4.3 shows the step response of the output variables *x; y; z;* when the Feedback linearization with pole placement controller is used.

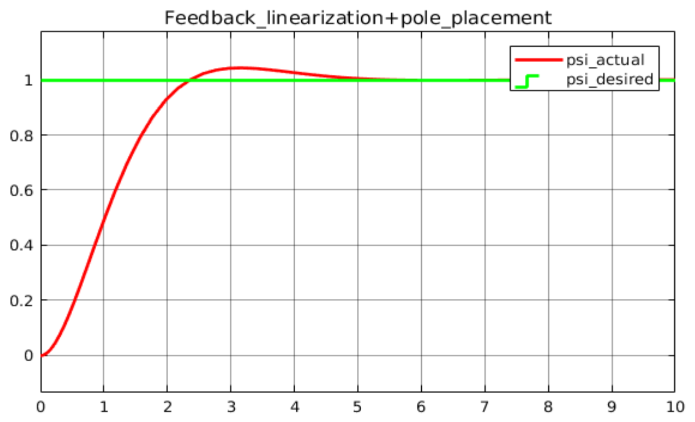


(a)



(b)





(c) (d)

Figure 4.3: step response of feedback linearization with Pole placement (a): $x(t)$, (b): $y(t)$, (c): $z(t)$, (d): $\psi(t)$

Table 4.4 shows some characteristic parameters of the step response when Feedback linearization with pole placement controller is used.

	$x(t)$	$y(t)$	$z(t)$	$\psi(t)$
Rise time [s]	1.612	1.612	1.612	1.507
Overshoot [%]	3.646	3.646	3.646	4.737
Settling time [s]	2.431	2.431	2.431	2.084

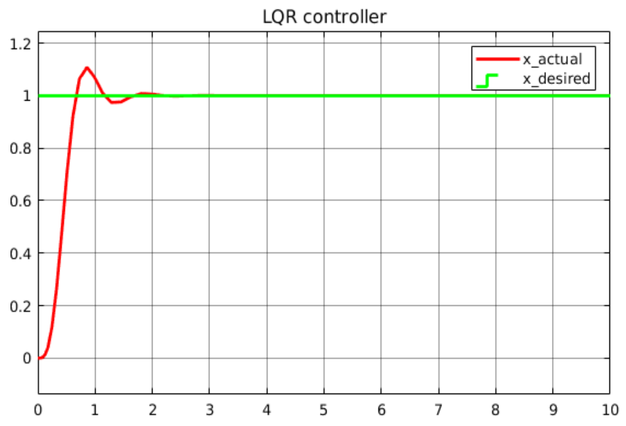
Table 4.4: Characteristic parameters to a step input.

4.1.4 LQR control results

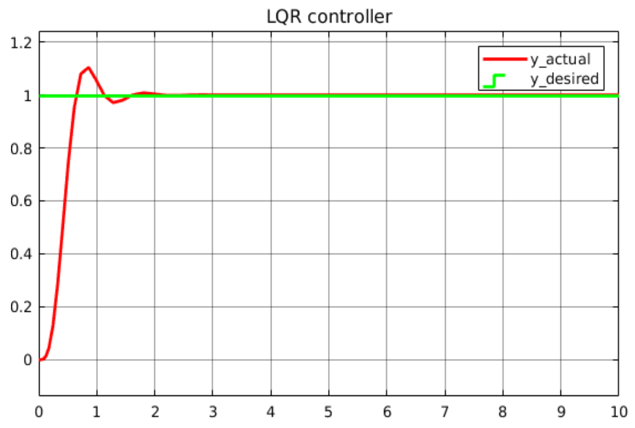
Based on linear mode (1.51), the LQR has been implemented with $R = \text{diag}\{1,1,1,1\}$ and $Q = \text{diag}\{1,1,100000,1,1,1,1,1,1,100000,100000,100000\}$. The obtained gain matrix is:

```
K_lqr =

   1.0e+03 *

    0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    1.4288    0.0000    0.0000    3.1623
    0.7182   -0.0000   -0.0000    0.0805   -0.0000   -0.0000    0.0000    0.3827    0.0000    0.0000    1.0000    0.0000
   -0.0000    0.7644    0.0000   -0.0000    0.0884   -0.0000   -0.3948   -0.0000   -0.0000   -1.0000   -0.0000    0.0000
   -0.0000   -0.0000    1.0000   -0.0000   -0.0000    0.1379    0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000
```

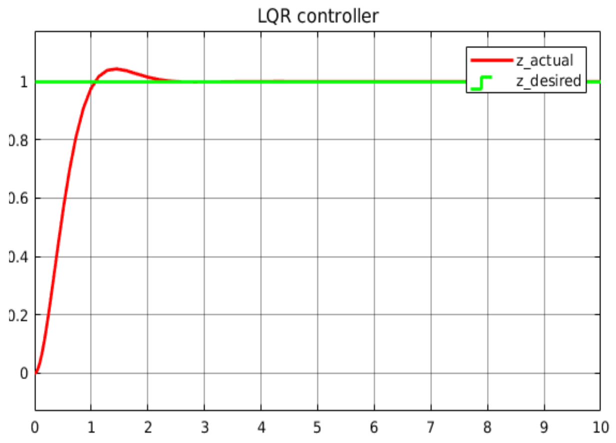
Figure 4.4 shows the step response of the output variables *x; y; z;* when the LQR controller is used.



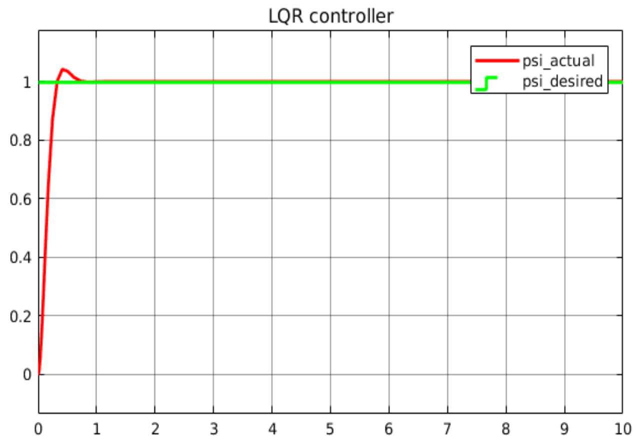
(a)



(b)



(c)



(d)

Figure 4.4: step response of LQR controller (a): $x(t)$, (b): $y(t)$, (c): $z(t)$, (d): $y(t)$

Table 4.5 shows some characteristic parameters of the step response when Feedback linearization with pole placement controller is used.

	z(t)	$y(t)$	$z(t)$	$\psi(t)$
Rise time [s]	0.369	0.360	0.681	0.212
Overshoot [%]	10.56	10.49	4.737	4.831
Settling time [s]	1.039	1.006	1.680	0.298

Table 4.5: Characteristic parameters to a step input.

4.1.5 Discussion:

Based on the results we have obtained above (Table 4.2, Table 4.3, Table 4.4 and Table 4.5), the four controllers gave comparable dynamic performance in term of rising time, settling time and overshoot. However, we can make the following observation for each controllers:

- The PD is faster than SM and FL, but it has a lower value of overshoot than FL and LQR.
- The SM is slower than PD and LQR, but it has the smallest value of overshoot. In other hand, SM produce the chattering effect due to its high frequency switching nature which is can adversely affect the actuators.
- The FL is the slowest and has a higher value of overshooting than PD and SM.
- The LQR is the fastest, but has the highest value of overshooting and a huge control gain.

For the implementation of our autonomous navigation system, we have used PID controller due to its performance and also because our system is working around the hovering point (small oscillation assumption). In addition to that, the notable advantage of the PID is that its control law is not a function of the system parameters, it is only a function of the state error which makes it easy to implement.

4.2 Autonomous navigation system implementation and results

In our work we have chosen to work with Tum_ardrone ROS package that would help us achieve our aim of autonomous navigation. Tum_ardrone's system consists of three components: a monocular SLAM system, an extended Kalman filter for data fusion and state estimation and a PID controller to generate steering commands.

- **Monocular SLAM:**
  The SLAM system used is based on Parallel Tracking and Mapping (PTAM). Using the pose estimates from the EKF falsely tracked frames are identified and rejected. A novel, closed-form solution to estimate the absolute scale of the generated visual map from inertial and altitude measurements was implemented on this package as well.
- **Extended Kalman Filter:**
  An extended Kalman filter is employed to fuse all available data, it is also used to compensate for the different time delays in the system, arising from wireless LAN communication and computationally complex visual tracking. Full motion model of the quadrocopter's flight dynamics and reaction to control commands was derived and calibrated as well.
- **PID Control:**
  Based on the position and velocity estimates from the EKF, PID control is applied to steer the quadrocopter towards the desired goal location $P = (\hat{x}, \hat{y}, \hat{z}, \hat{\psi}) \in \mathbb{R}^4$ in a global coordinate system. For each of the four degrees-of freedom, a separate PID controller is employed for which we designed suitable controller gains.

Tum_ardrone is composed of three nodes:

- **drone_stateestimation:** its main role is to estimate the drone's position based on sent sensors data, PTAM data and sent control commands.
  Subscribed topics**:**
  - /ardrone/navdata
  - /ardrone /image_raw
  - /cmd_vel

  Published topics:
  - /ardrone/predictedPose

- **drone_autopilot:** It used the implemented PID controller to follow the desired way-points sent using **Drone_gui** node.

  Subscribed topics:
  - /ardrone/predictedPose

  Published topics:
  - /cmd_vel
- **Drone_gui:** This node offers a simple QT GUI to send control commands to the drone either via the **drone_autopilot** node, or manually via keyboard.

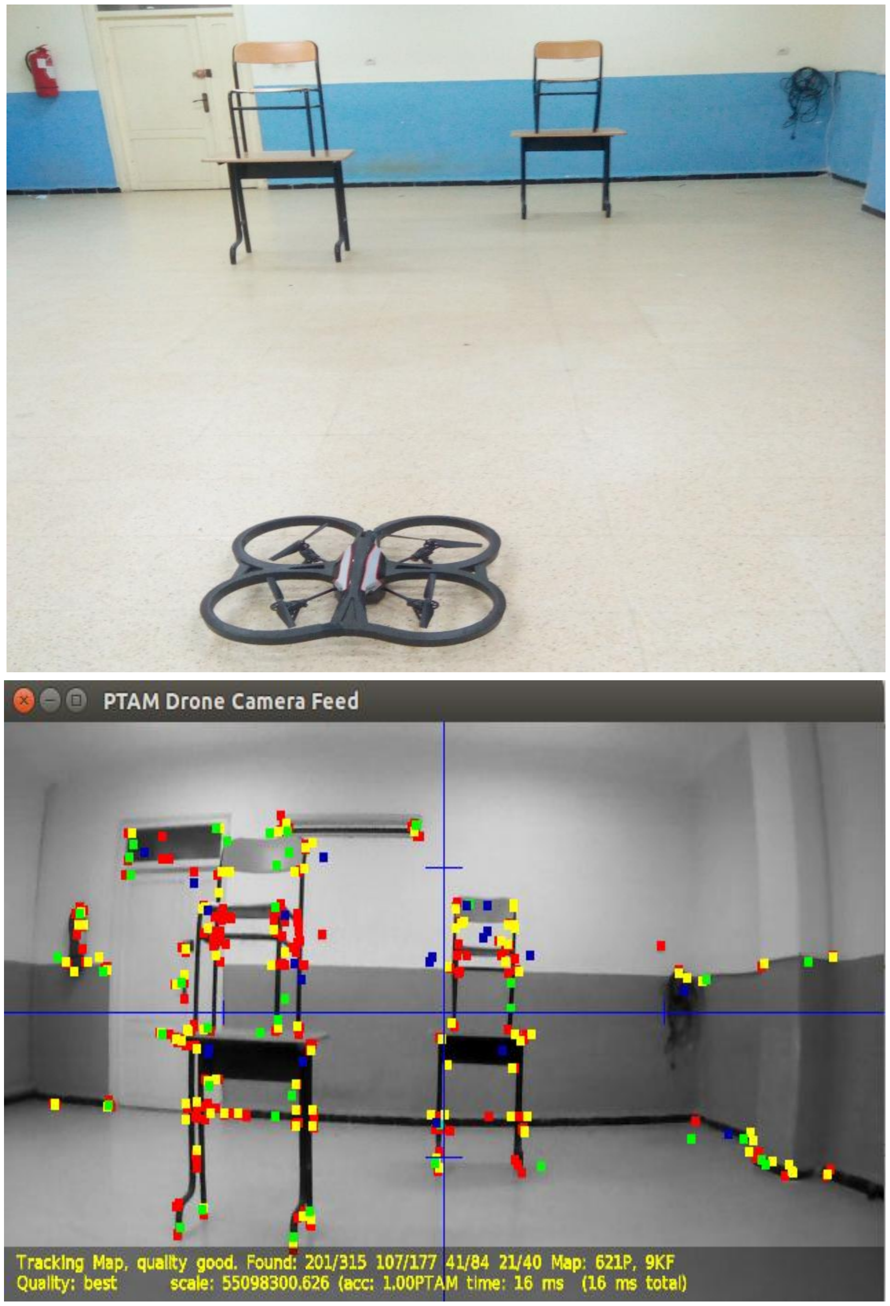
  Subscribed topics:
  - /ardrone/predictedPose
  - /cmd_vel
  - /ardrone/navdata
  - /joy

  Published topics:
  - /cmd_vel

Our approach was to use Tum_ardrone package to localize the robot and generate 3D point cloud of the environment, this point cloud is then sent to a MATLAB algorithm that will segment and cluster it. The centroids of the segments are computed and handled as obstacles. Furthermore a binary occupancy grid map has been built using the detected obstacles. A path planning algorithm based on PRM has been used to generate an obstacle-free optimal path for the drone. This path is then sent back to ROS through a topic to be followed using the **drone_autopilot** node.
To test our approach we have conducted many experiments the process can be summarized as follow:

4.2.1 Point cloud generation:

Tum_ardrone package does not originally publish key points or point cloud data of the environment so we had to modify the code in order to integrate this functionality.
We have used The Point Cloud Library (**PCL**) which is a standalone, large scale, open project for 2D/3D image and point cloud processing In order to retrieve and store and publish the point cloud data through a ROS topic.
Robotics System Toolbox™ provides an interface between MATLAB and the Robot Operating System (ROS) that enables them to communicate and interactively exchange messages through topics.



```
Tracking Map, quality good. Found: 201/315 107/177 41/84 21/40 Map: 621P, 9KF
Quality: best       scale: 55098300.626 (acc: 1.00PTAM time: 16 ms  (16 ms total)
```

Figure 4-5: real world environment with 2 obstacles

As shown in the figure 4-5 our environment consists of 2 obstacles, the built map of the environment is shown in figure 4-6, gave a good representation of the surroundings. The next step after sending this point cloud data to MATLAB is to Cluster it.

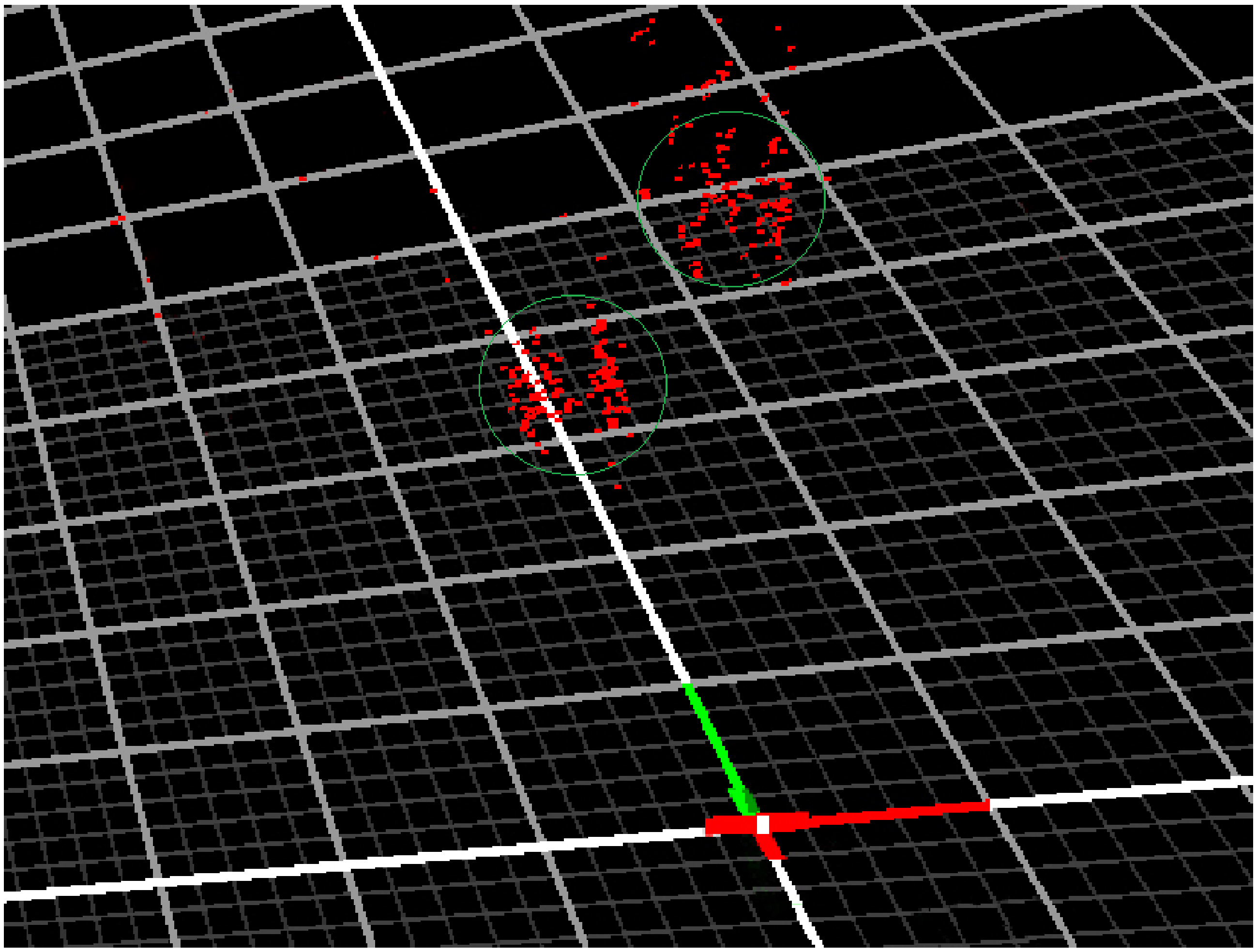


Figure 4-6: The map of the environment generated by PTAM in the form of Point cloud data

4.2.2 Clustering and segmentation:

After subscribing to the point cloud topic sent from ROS the coordinates of each point has been extracted and saved it in a matrix where Rows correspond to points and columns correspond to features (x, y, z), This matrix is passed as an argument to the clustering algorithm, we have chosen the number of clusters to be 2 same as the number of the obstacles. The output of this step is the location of the obstacles.

4.2.3 Occupancy grid building:

An occupancy grid has been created with a specific height, width and resolution. The locations of the centroids returned from the previous step are set as occupied in the map, and it has been inflated with a specific radius to insure there will be no collision with the robot. Because the occupancy grid implementation does not accept negative values the obstacles have been shifted up on the x-axis.

4.2.4 Path generation:

The path planning algorithm takes the map, starting and the goal locations as input and returns the way points the robot should take in order to reach the goal destination.

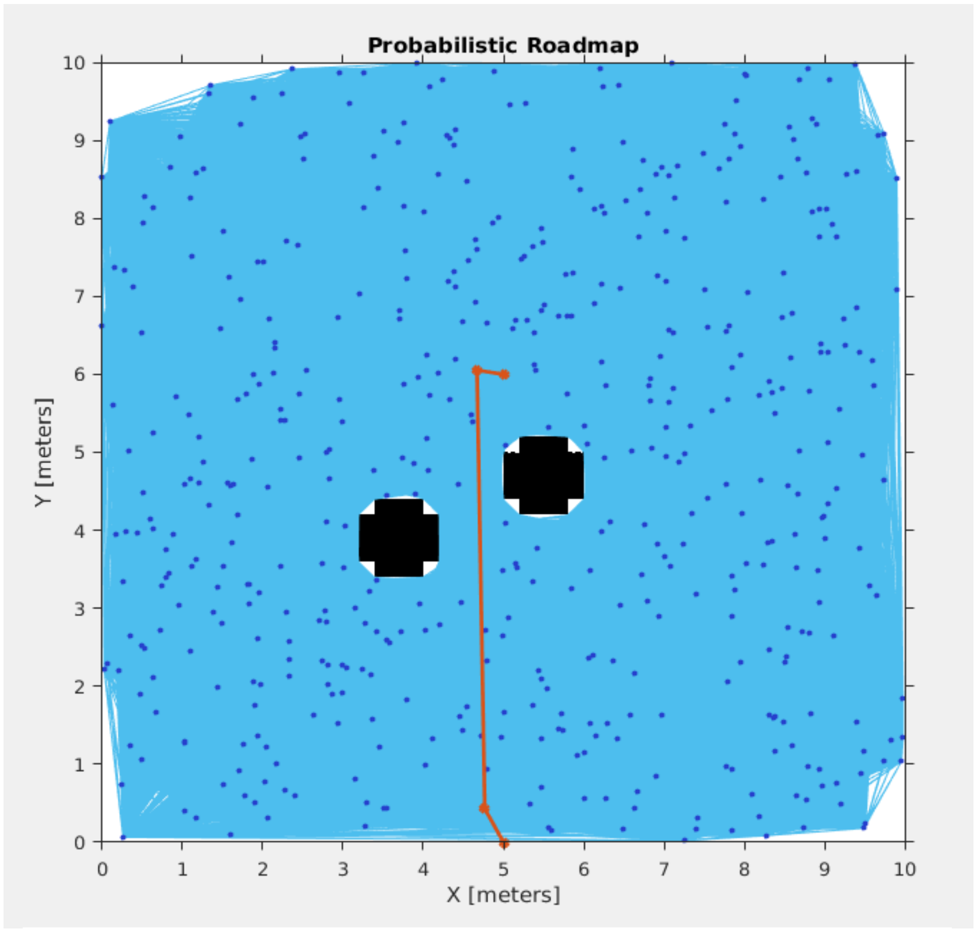


Figure 4-7: occupancy grid map with the detected two obstacles and the generated path.

The waypoint composing the trajectory are sent back to Tum_ardrone package, which has been modified to receive these way points and sends the steering command to the real drone.

Many experiments has been conducted to test the approach in various environments. It has been noticed that the algorithm works better in indoor textured environments with a good lighting, because the SLAM system is able to detect bigger number of keypoints. In general the experiments proved the applicability of the approach in terms of obstacles detections and avoidance.

Conclusion

On this thesis a control design for ardrone2.0 has been presented, as well as collision avoidance system that allows the quadrotor to navigate in previously unknown environment based on the monocular camera.

The mathematical model has been developed and used to design a number of controllers for namely: LQR, PID, Sliding mode and feedback linearization with pole placement. Furthermore they have been compared based on their performance. PID controller has been used in an autonomous navigation system. Tum_ardrone ROS package were used for simultaneous Localization of the robot and mapping of the surrounding environment as well as an accurate state estimation. The environment was published in the form of point cloud which has been clustered in MATLAB and used for obstacle detection. The location of these obstacles has been used to construct a binary occupancy grid. Finally a PRM algorithm has been used to generate an obstacle free path that is followed by the drone.

Our results have been validated with experiments using real Ardrone2.0. as a future work we propose an adaptive PID controller ,that is able to tune its coefficients based on a learning algorithm, to overcome the uncertainties and the disturbance of the system. On the other hand we propose the use of ORB-SLAM system for an accurate map building and obstacle detection.

Appendix-A: Quadrocopter

The Parrot AR.Drone2.0 was introduced in January 2010, originally designed as a sophisticated toy for augmented reality games. It is meant to be controlled by a pilot using a smart phone or a tablet PC. In spite of the original design as a high-tech toy, the drone quickly caught attention of universities and research institutions, and today is used in several research projects in the fields of Robotics, Artificial Intelligence and Computer Vision in contrast to many other available remote controlled aerial vehicles, the drone with a retail price of only 300 dollars is inexpensive, robust, and easy to use and fly.

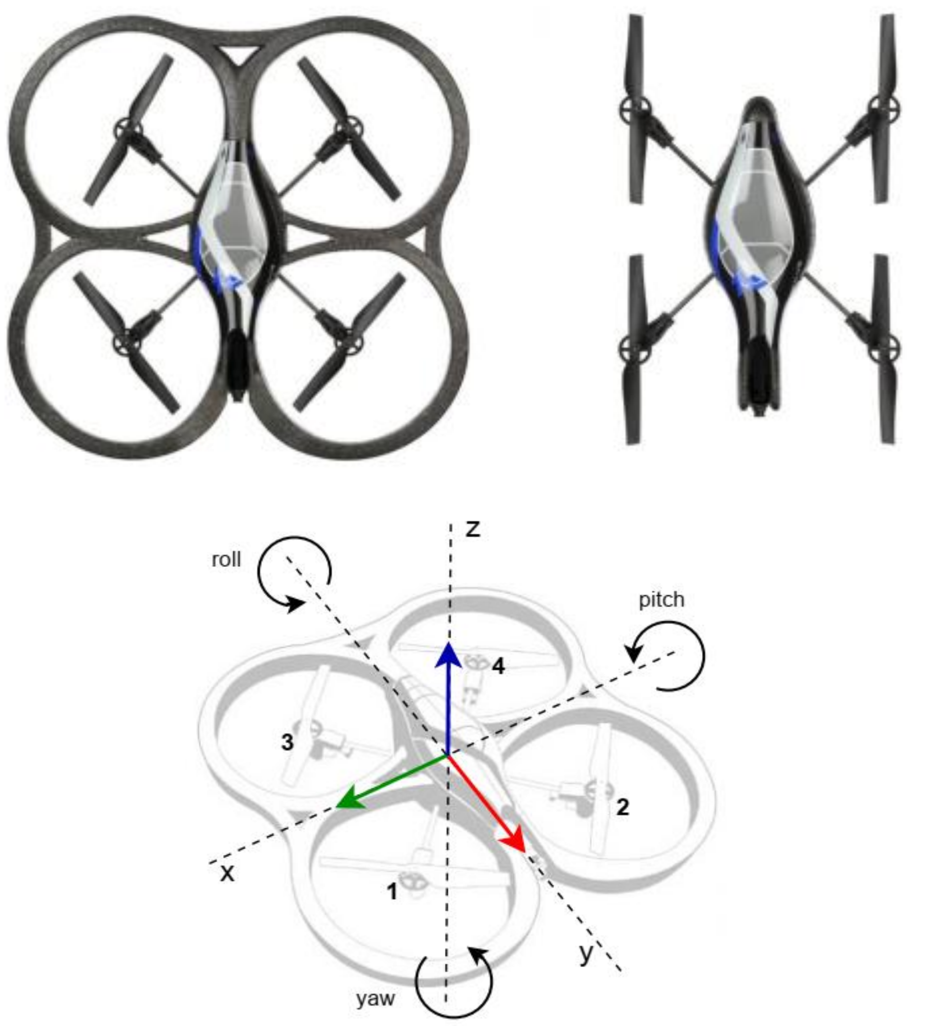


Figure A-1 Schematic of the Parrot AR.Drone2.0

A-1 Hardware

A-1.1 Basic Quadrocopter Mechanics

A quadrocopter is a heicopter, which is lifted and maneuvered by four rotors. It can be maneuvered in three-dimensional space solely by adjusting the individual engine speeds while all four rotors contribute to the upwards thrust, two opposite ones are rotating clockwise (rotors 2 and 3) while the other two (rotors 1 and 4) are rotating counter-clockwise, canceling out their respective torques. Ignoring mechanical inaccuracies and external influences, running all engines at equal speed - precisely nullifying gravity - allows a quadrocopter to stay in the air without moving. The following actions can be taken to maneuver the quadrocopter:

- Vertical acceleration is achieved by increasing or decreasing the speed of all four rotors equally.
- Yaw rotation can be achieved by increasing the speed of engines 1 and 4, while decreasing the speed of engines 2 and 3 (or vice-versa) - resulting in an overall clockwise (or counter-clockwise) torque, without changing overall upwards thrust or balance.
- Horizontal movement can be achieved by increasing the speed of one engine, while decreasing the speed of the opposing one, resulting in a change of the roll or pitch angle, and thereby inducing horizontal acceleration. The fine tuning of the relative engine speeds is very sensible to small changes, making it difficult to control a quadrocopter without advanced controlling routines and accurate sensors.

A-1.2 The Parrot AR.Drone2.0

The Parrot AR.Drone2.0 has dimensions of 52.5 cm×51.5 cm with, and 45 cm×29 cm without hull. It has four rotors with a 20 cm diameter, fastened to a robust carbon-fiber skeleton cross providing stability. A removable styrofoam hull protects the drone and particularly the rotors during indoor-flights, allowing the drone to survive minor and not-so-minor crashes such as flying into various types of room furniture, doors and walls – making it well suited for experimental flying and development. An alternative outdoor-hull - missing the rotor protection and hence offering less protection against collisions - is also provided and allows for better maneuverability and higher speeds. The drone weights 380 g with the outdoor-hull, and 420 g with the indoor-hull. Although not officially supported, in our tests the drone was able to fly with an additional payload of up to 120 g using the indoor hull - stability, maneuverability and battery life however suffered significantly, making the drone hardly controllable with that kind of additional weight. The drone is equipped with two cameras (one directed forward and one directed downward), an ultrasound altimeter, a 3-axis accelerometer (measuring acceleration), a 2-axis gyroscope (measuring pitch and roll angle) and a one-axis yaw precision gyroscope. The onboard controller is composed of an ARM9 468 MHz processor with 128 Mb DDR Ram, on which a BusyBox based GNU/Linux distribution is running. It has an USB service port and is controlled via wireless LAN.

Cameras

The AR.Drone has two on-board cameras, one pointing forward and one pointing downward. The camera pointing forward runs at 18 fps with a resolution of 640 × 480 pixels, covering a field of view of 73.5◦ × 58.5◦. Due to the used fish eye lens, the image is subject to significant radial distortion. Furthermore rapid drone movements produce strong motion blur, as well as linear distortion due to the camera’s rolling shutter (the time between capturing the first and the last line is approximately 40 ms). The camera pointing downwards runs at 60 fps with a resolution of 176 × 144 pixels, covering a field of view of only 47.5◦ × 36.5◦, but is afflicted only by negligible radial distortion, motion blur or rolling shutter effects. Both cameras are subject to an automatic brightness and contrast adjustment.

Gyroscopes and altimeter

The measured roll and pitch angles are, with a deviation of only up to 0.5◦, surprisingly accurate and not subject to drift over time. The yaw measurements however drift significantly over. Furthermore an ultrasound based altimeter with a maximal range of 6 m is installed on the drone.

A-2 Software

The Parrot AR.Drone comes with all software required to fly the quadcopter. Due to the drone being a commercial product which is primarily sold as high-tech toy and not as a tool for research, accessing more than this basic functionality however turns out not to be so easy. The first and most important drawback is, that the software running onboard is not accessible: while some basic communication via a telnet shell is possible, the control software is neither open-source nor documented in any way - while custom changes including starting additional processes are possible, this would require massive trial and error and is connected with a risk of permanently damaging the drone.

A-2.1 Communication Channels

As soon as the battery is connected, the drone sets up an ad-hoc wireless LAN network to which any device may connect. Upon connect, the drone immediately starts to communicate (sending data and receiving navigational commands) on four separate channels:

- navigation channel (UDP port 5554)
- video channel (UDP port 5555)
- command channel (UDP port 5556)
- control port (TCP port 5559, optional)

Note that the three major communication channels are UDP channels, hence packages may get lost or be received in the wrong order. Also note the complete lack of any security measures - anybody may connect to and control a running drone at any time, no password protection or encryption is possible.

Navigation Channel

While in normal mode the drone only broadcasts basic navigational data every 30 ms, after switching to debug mode it starts sending large amounts of sensor measurements every 5 ms. The most important parameters and sensor values - and the ones used in our approach - are the following:

- Drone orientation as roll, pitch and yaw angles: as mentioned in the previous section, roll and pitch values are drift-free and very accurate, while the measured yaw-angle is subject to significant drift over time.
- Horizontal velocity: in order to enable the drone to keep its position in spite of wind, an optical-flow based motion estimation algorithm utilizing the full 60 fps from the floor camera is performed onboard, estimating the drone’s horizontal speed. The exact way these values are determined however is not documented.

Experiments have shown that the accuracy of these values strongly depends on whether the ground below the drone is textured or not: when flying above a textured surface (or, for example, a cluttered desk) these values are extremely accurate

- when flying above a poorly textured surface however, the quality of these speed estimates is very poor, deviating from the true value by up to 1 m/s above completely untextured surfaces.

- Drone height in millimeter: this value is based solely on the ultrasound altimeter measurements. As long as the drone is flying over a flat, reflecting surface, this value is quite accurate, with (at a height of 1 m) a mean error of only 8, 5 cm. As this sensor measures relative height, when flying over uneven surfaces or close to walls strong fluctuations will occur. This often induces sudden and undesired vertical acceleration, as the drone tries to keep its relative height as supposed to its absolute height. This value is measured only every 40 ms.
- Battery state as an integer between 100 and 0.
- The control state as a 32-bit bit field, indicating the drone’s current internal status. This might for example be “LANDED”, “HOVERING”, “ERROR”, “TAKEOF” etc.,
- The drone’s internal timestamp, at which the respective data was sent, in microseconds. This is not necessarily the time at which the sensor values were taken, experiments have shown that within the same package, some parameters are up to 60 ms older than others.

Video Channel

The drone continuously transmits one video stream, which can be one of four different channels - switching between channels can be accomplished by sending a control command to the drone. The four available channels are depicted in Figure 2.3. As can be seen, neither of the available cameras can be accessed fully: for the downwards facing camera the available frame rate is - with only 18 fps - significantly lower than the original 60 fps. Furthermore the maximal supported resolution is 320 × 240, halving the forward camera’s original resolution1. The video stream is encoded using a proprietary format, based on a simplified version of the H.263 UVLC codec [41]. Images are encoded in YCBCR color space, 4:2:0 type2, using 8 bit values. More details can be found in [32]. While the achieved compression is fairly good (in practice around 10 kB per frame, resulting in a bandwidth required of only 180 kBps), this encoding produces significant artifacts in the decoded picture.

Command Channel

The Drone is navigated by broadcasting a stream of command packages, each defining the following parameters:

1. Desired roll and pitch angle, yaw rotational speed as well as vertical speed, each as fraction of the allowed maximum, i.e. as value between -1 and 1.

2. One bit switching between hover-mode (the drone tries to keep its position, ignoring any other control commands) and manual control mode,
3. One bit indicating whether the drone is supposed to enter or exit an error-state, immediately switching off all engines,
4. One bit indicating whether the drone is supposed to take off or land.

Being sent over an UDP channel, reception of any one command package cannot be guaranteed. In our implementation the command is therefore re-sent approximately every 10 ms, allowing for smoothly controlling the drone.

Control Port

Control commands can be used to change internal settings of the drone, for example for switching between the four available video channels. In general a control command is transmitted as a string of the format “[attribute]=[value]”.

Appendix-B: ROS

B-1 Introduction to Robot Operating System ROS

Robot Operating System (ROS) is a trending robot application development platform that provides various features such as message passing, distributed computing, code reusing, and so on. The ROS project was started in 2007 with the name Switchyard by Morgan Quigley as part of the Stanford STAIR robot project. The main development of ROS happened at Willow Garage. Here are some of the reasons why people choose ROS over other robotic platforms such as Player,YARP, Orocos, MRPT, and so on :

- ***High-end capabilities***: ROS comes with ready to use capabilities, for example, SLAM (Simultaneous Localization and Mapping) and AMCL (Adaptive Monte Carlo Localization) packages in ROS can be used for performing autonomous navigation in mobile robots and the MoveIt package for motion planning of robot manipulators.
- ***Tons of tools***: ROS is packed with tons of tools for debugging, visualizing, and performing simulation. The tools such as rqt_gui, RViz and Gazebo are some of the strong open source tools for debugging, visualization, and simulation. The software framework that has these many tools is very rare.
- ***Support high-end sensors and actuators***: ROS is packed with device drivers and interface packages of various sensors and actuators in robotics. The high-end sensors include Velodyne-LIDAR, Laser scanners, Kinect, and so on and actuators such as Dynamixel servos. We can interface these components to ROS without any hassle.
- ***Inter-platform operability***: the ROS message-passing middleware allows communicating between different nodes. These nodes can be programmed in any language that has ROS client libraries. We can write high performance nodes in C++ or C and other nodes in Python or Java. This kind of flexibility is not available in other frameworks.
- ***Modularity:*** One of the issues that can occur in most of the standalone robotic applications are, if any of the threads of main code crash, the entire robot application can stop. In ROS, the situation is different, we are writing different nodes for each

process and if one node crashes, the system can still work. Also, ROS provides robust methods to resume operation even if any sensors or motors are.

- ***Concurrent resource handling***: Handling a hardware resource by more than two processes is always a headache. Imagine, we want to process an image from a camera for face detection and motion detection, we can either write the code as a single entity that can do both, or we can write a single threaded code for concurrency. If we want to add more than two features in threads, the application behavior will get complex and will be difficult to debug. But in ROS, we can access the devices using ROS topics from the ROS drivers. Any number of ROS nodes can subscribe to the image message from the ROS camera driver and each node can perform different functionalities. It reduce the complexity in computation and also increase the debug-ability of the entire system.
- ***Active community***: When we choose a library or software framework, especially from an open source community, one of the main factors that needs to be checked before using it is its software support and developer community. There is no guarantee of support from an open source tool. Some tools provide good support and some tools don't. In ROS, the support community is active. The ROS community has a steady growth in developers worldwide.

B-2 Understanding the ROS file system level

Similar to an operating system, ROS files are also organized on the hard disk in a particular fashion. In this level, we can see how these files are organized on the disk. The following graph shows how ROS files and folder are organized on the disk:

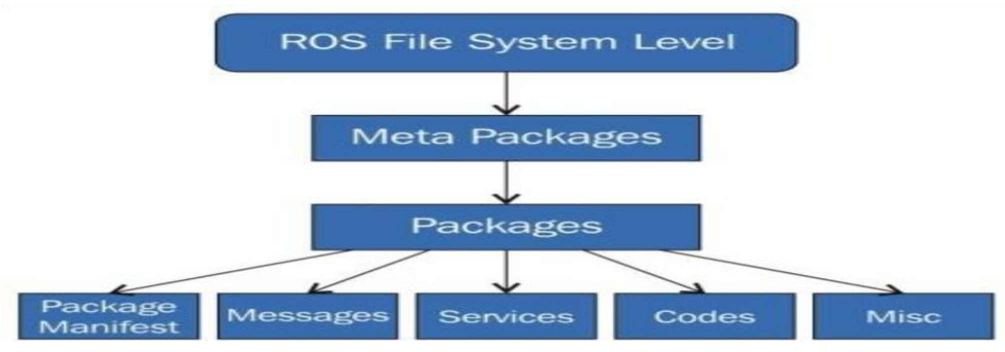


Figure B-1. The ROS file system level

Similar to an operating system, an ROS program is divided into folders, and these

- ***Packages:*** Packages form the atomic level of ROS. A package has the minimum structure and content to create a program within ROS. It may have ROS Runtime process (nodes), configuration files; and so on.
- ***Manifests:*** Manifests provide information about a package, license information, dependencies, compiler flags, and so on. Manifest are managed with a file called *manifests.xml*.
- ***Stacks***: When you gather several packages with some functionality, you will obtain a stack. In ROS, there exists a lot of these stacks with different uses, for example, the navigation
- ***Stack manifests***: Stack manifests (stack.xml) provide data about a stack, including its license information and its dependencies on other stacks.
- ***Message*** (msg) types: A message is the information that a process sends to other processes. ROS has a lot of standard types of messages. Message descriptions are stored in my_package/msg/MyMessageType.msg.
- ***Service (srv) types***: Service descriptions, stored in my_package /srv/ My Service Type.srv, define the request and response data structures for services in ROS.

B-3 ROS computational Graph level

ROS creates a network where all the processes are connected. Any node in the system can access this network, interact with other nodes, see the information that they are sending, and transmit data to the network

- ***Nodes***: ROS nodes are a process that perform computation using ROS client libraries such as roscpp and rospy. One node can communicate with other nodes using ROS Topics, Services, and Parameters.
- ***Topic***: Chanel between two or more nodes, nodes communicate by publishing and/or subscribing to the appropriate topics
- ***Services***: ROS uses a simplified service description language for describing ROS service types. This builds directly upon the ROS msg format to enable request/response communication between nodes. Service descriptions are stored in .srv file in the srv/ subdirectory of a package.

- ***Parameters:*** The Parameter Server gives us the possibility to have data stored using keys in a central location. With this parameter, it is possible to configure a nodes while it's running or to change the working of the nodes.

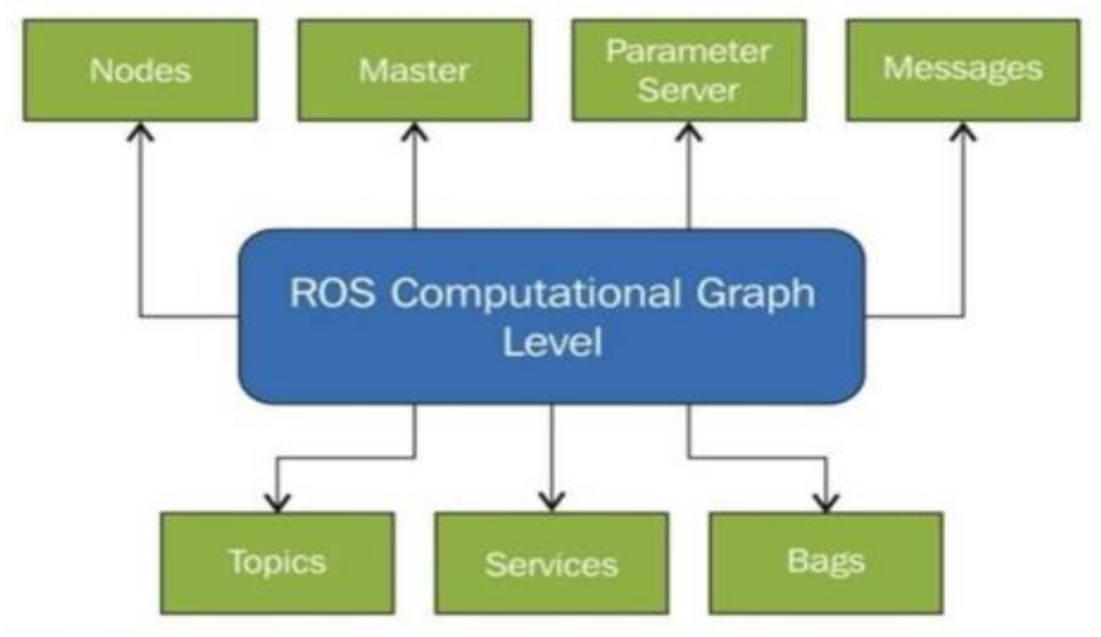


Figure B-2. ROS computational Graph level

Appenix-C: History and development of SLAM systems

The SLAM problem has been formulated first in 1986 [17], First solutions proposed were based on the extended Kalman filter (EKF), these methods are called **EKF-SLAM**. The map is a large state vector **x** that consists of both the Position of landmarks as well as the current position of the robot.

$$x = \begin{bmatrix} R \\ M \end{bmatrix} = \begin{bmatrix} R \\ L1 \\ \\ Ln \end{bmatrix} \qquad \text{(C.1)}$$

R : is the robot state.

M: is the set of landmark states, with *n* the current number of mapped landmarks.

$$\bar{x} = \begin{bmatrix} \bar{R} \\ \bar{M} \end{bmatrix} = \begin{bmatrix} \bar{R} \\ \bar{L}1 \\ \\ \bar{L}n \end{bmatrix} \qquad P = \begin{bmatrix} P_{RR} & P_{RM} \\ P_{MR} & P_{MM} \end{bmatrix} = \begin{bmatrix} P_{RR} & P_{RL_1} & & P_{RL_n} \\ P_{L_1R} & P_{L_1L_1} & & P_{L_1L_n} \\ & & & \\ P_{L_nR} & P_{L_nL_1} & & P_{L_nL_n} \end{bmatrix} \qquad \text{(C.2)}$$

The goal of EKF-SLAM, therefore, is to keep the map $\{\bar{x}, P\}$ up to date at all times.

From the fact that a full update of the maintained covariance matrix of **x** is required for each new pose-estimate the computational complexity of this approach scales quadratically, which limits the number of landmarks to a few hundred in practice, making it impossible to navigate in larger environments.

This limitation was addressed by **FastSLAM** by Montemerlo et al. [18]: The observation that conditioned on the robot's path, the positions of the individual landmarks become independent allows for each landmark's position to be estimated independently. Using a particle filter instead of a Kalman filter (each particle representing one possible path taken and maintaining its own estimate of all landmark positions), leads to a naive complexity for each update of $O(kn)$, k being the number of particles and n the number of landmarks.

Using a tree-based data structure, this can be reduced to $O(k \log n)$, hence logarithmic instead of quadratic complexity in the number of landmarks - rendering maps containing tens of thousands of landmarks computationally feasible [12].

The emergence of keyframe-based methods such as **parallel tracking and mapping** (PTAM) by G. Klein and D. Murray in 2007 [19] was a major development of monocular SLAM methods. Keyframe-based approaches are different in many ways from filtering-based approaches such as EKF-SLAM. Instead of using probability distribution to marginalize out previous poses and summarize all information, keyframe-based approaches retain a selected subset of previous observations - called keyframes - explicitly representing past knowledge gained.

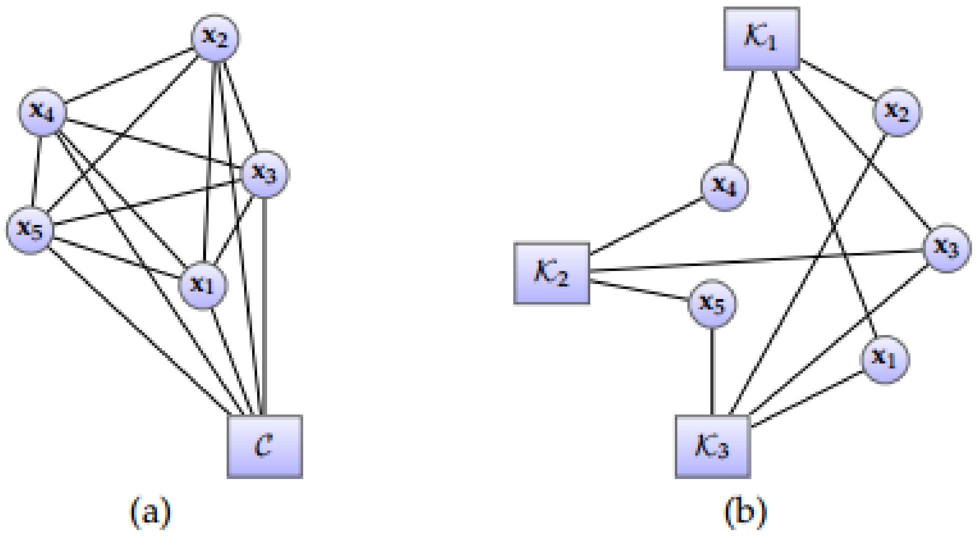


Figure C.1: (a) Visual representation of the map for filtering-based approaches, **(b)** Visual representation of the map for keyframe-based approaches.

As illustrated by Figure C.1 in the filtering-based approaches correlations between landmark positions are explicitly represented as covariance, and the current camera position is an integral part of the map. Whereas in keyframe-based approaches correlations between landmarks are not explicitly represented and observations serve as link between landmarks and keyframes.

The process of Keyframe-based SLAM can be split into two major tasks:

1. **Tracking:** estimating the position of the camera C based on a fixed map, using only the landmark positions $\mathbf{x}i$.

2. **Mapping:** optimizing and integrating new keyframes and landmarks into the map, as well as performing other modifications such as removing invalid landmarks, observations or even keyframes.

References

[1] N. Nieuwenhuisen, D. Droeschel, M. Beul, S. Behnke, Obstacle detection and navigation planning for autonomous micro aerial vehicles, Proc. 2014 IEEE Int. Conf. Unmanned Aircraft Systems.

[2] H. Alvarez, L.M. Paz, J. Sturm, D. Cremers, Collision Avoidance for Quadrotors with a Monocular Camera, in Experimental Robotics, Springer international Publishing, 2016, pp. 195 209.

[3]Francesco D'apolito,Christoph Sulzbachner, Obstacle Avoidance System Development for the Ardrone 2.0 using the tum_ardrone Package, Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS) Linköping, Sweden, October 3-5, 2017

[4] Amr Nagaty, Sajad Saeedi, Carl Thibault, Mae Seto, and Howard Li. Control and navigation framework for quadrotor helicopters. Journal of Intelligent and Robotic Systems, 70(1-4):1{12, 2013. ISSN 0921-0296. doi: 10.1007/s10846-012-9789-z. URL http://dx.doi.org/10.1007/s10846-012-9789-z.

[5] A. Azzam and Xinhua Wang. Quad rotor arial robot dynamic modeling and configuration stabilization, In Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on, volume 1, pages 438{444, 2010. doi: 10.1109/CAR.2010.5456804.

[6] Gergely Regula, Formation control of autonomous aerial vehicles, Phd thesis, Budapest University of Technology and Economics, 2013.

[7]F. Sabatino, "Quadrotor control : modeling, nonlinear control design, and simulation," no. June, 2015.

[8] C.-T. Chen, Linear System Theory And Design, Third edition. New York: OXFORD university press, 1999.

[9] F. Xiang, "Block-Oriented Nonlinear Control of Pneumatic Actuator Systems," Doctoral thesis, Sweden university 2001.

[10] Samir Bouabdallah and Roland Siegwart "Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor" Proceedings of the 2005 IEEE International Conference on Robotics and Automation Barcelona, Spain,
April 2005

[11] B. Bekhiti, K. Hariche, A. Dahimene, and B. Nail, "Intelligent block spectral factors relocation in a quadrotor unmanned aerial vehicle," 2017.

[12] Jakob Julian Engel. Autonomous Camera-Based Navigation of a Quadrocopter, 2011. Master's Thesis

[13] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In Proc. of the European Conference on Computer Vision (ECCV), 2006.

[14] C. Harris and M. Stephens. A combined corner and edge detector. In Proc. of the Alvey Vision Conference, 1988.

[15] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An accurate O(n) solution to the PnP problem. International Journal of Computer Vision (IJCV), 81(2):155 – 166, 2009.

[16] L. Kavraki, P. Svestka, .I.-C. Lato:mbe, M. Overmars, “Probabilistic roadmaps for fast path planning in high dimensional configuration spaces”, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 12, NO. 4, AUGUST 1996

[17] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part I. Robotics & Automation Magazine, 13(2):99 – 110, 2006.

[18] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In Proc. of the AAAI National Conference on Artificial Intelligence, 2002.

[19] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In Proc. of the International Symposium on Mixed and Augmented Reality (ISMAR), 2007.