Sheffield Hallam University

SHARPENS YOUR THINKING

SHEFFIELD HALLAM UNIVERSITY

ENGINEERING PROGRAM

A QUADCOPTER WITH AUTOMATED TAKE-OFF AND LANDING ON MOBILE ROBOT PLATFORM

Group Report

Group No: ML01/SM16

Submitted by

Sandaruwan B. A. S.

S.Mithun

Rathnayake R. M. K. M.

Supervised by

Dr. Migara Liyanage

M. ENG. (HONS) IN ELECTRONIC ENGINEERING

MODULE: 55-7618

December 2016

Preface

This report describes project work carried out within the engineering program at Sri Lanka Institute of Information Technology University between April and December 2016. The submission of the report is in accordance with the requirement of the degree of Electronic Engineering (MEng) under the guidance of the University.

Acknowledgement

At the end of a successful journey seeking to bring to light, a quadcopter with automated take-off and landing on mobile robot platform, based on concepts which were completely novel to us, it is with the utmost pleasure that we acknowledge the contribution of all individuals who has being supportive in making this project a success.

We would like to take this opportunity to express our deep sense of gratitude and profound feeling of admiration to our project supervisor Dr. Migara Liyanage. Many thanks go to him who helped us in this work and under whose supervision that we gained a clear concept of what we should do. A special thanks to our project coordinator Dr. Minhua Ding at Sri Lanka Institute of Information Technology for giving a good guideline for select this project throughout numerous consultations. Our special thanks to Sri Lanka Institute of Information Technology for giving an opportunity to carry out this individual project. We would like to gratefully acknowledge our Head of Electronic & Computer Engineering department, Dr. Lasantha Seneviratne, for his guidance throughout the course. We would also like to thank all the lecturers and staffs in SLIIT who gave their support. Finally we like to thank our college mates and friends for their ideas and help.

Abstract

In this thesis, a controller is designed for an off the shelf quadcopter to give it the ability to autonomously takeoff, hover at a given altitude, follow and land on a mobile robot platform. This is a small part of a much bigger system which is a quadcopter and a mobile robot combined fully autonomous surveillance system. This system has the ability to navigate and complete a given task without any human interaction. Different types of sensor are used to determine the position of the quadcopter in 3D space. A PID controller is implemented to keep the quadcopter at a given altitude.

Different types of sensors and technologies were used to achieve our target. A discrete PID controller will be used to hold the altitude of the quadcopter. Real-time image processing is used to determine the position of the quadcopter relative to the mobile robot platform. An ideal quadcopter simulation and a 3D simulation of the task is done to understand in detail how a quadcopter works and how to controller it the way we desire. Kalman filter is used to produce accurate and precious angular data of the quadcopter.

The project is separated into several parts and divided among all the members of the group. The simulation of the complete system and the implementation of the takeoff, altitude holding and landing algorithms for the test system are done by me. Determining the position of the quadcopter using image processing and design and implementation of the Mobile robot platform is done by Rathnayake R.M.K.M. Implementation of Kalman filter to be used with Gyro and accelerometer sensors and the simulation of an ideal quadcopter model in Matlab is done by S. Mithun.

Key words: Quadcopter, UAV, PID, Matlab, Simulink.

Table of Contents

Table of Figures

Table of Tables

Abbreviations

BLDC - Brushless Direct Current Motors

CX - Center of X axis

CY - Center of Y axis

FPV - First Person View

GPS - Global Positioning System

IMU - Inertia Measurement Unit

LQR – Linear Quadratic Regulator

MEMS - Micro-Electro-Mechanical Systems

OpenCV - Open Computer Vision

PID - Proportional-Integral-Derivative

RPM - Rotations per minute

UART - Universal Asynchronous Receiver Transmitter protocol

UAV - Unmanned Ariel Vehicle

1.0 Introduction

The field of unmanned aerial vehicle (UAV) has been growing for the last few years and it is has a very important role nowadays, since the application of UAV can apply to variety of area such military operations, public applications, and civil applications.

- Military Operations
    - Border Security
    - Security Intelligence
    - Acquisition of Targets
    - Correction of Coordinates
    - Coast Guard
    - Cartography
    - Photography
- Public Applications
    - Search and Rescue
    - Pollution of Seas
    - Security of Petrol Pipe Line
    - Correction of Uncontrolled Trash Areas
    - Forest Fire
- Civil Applications
    - Control of Oil, Fuel and Connection Lines
    - Shooting Movie
    - Analysis of Fire Gas

A Quad-copter or quad-rotor is one of the UAVs that are major focuses of active researches in recent years. Quad-rotor is a helicopter with four rotors. The rotors are coordinated upwards and they are placed in a square shape with equal distance from the focal point of mass of the quad-copter [1] [2]. The movements of the quadcopter is controlled by adjusting the angular velocities of each motor separately [1].

The standard flight operations, for example, taking-off, landing and drifting are proposed for a quad-copter with indoor and outdoor flying capacities. This is accomplished by all the while controlling the speed of the four rotors all together for the quad-copter to achieve the right introduction. The aggregate thrust is resolved utilizing the contributions of elevation, pitch and

roll angles. Then the vertical thrust needed to keep the quadcopter hovering is calculated using the total thrust of the motors and the quadcopter tilt angles [3][6].

Like every other system available, quadcopters have some pros and cons. Pros are, quadcopter are versatile, fast, can easily reach places where normal robots or humans can't reach and so on. The cons are, quadcopters are not very reliable, can't carry bigger payloads, battery life is very short and so on.

The purpose of this project is to develop a proper control algorithm to autonomously take-off, hovering and land a quadcopter on the mobile robot platform. This is pursued with two aims. The first aim is modelling and simulate the quad-copter and mobile robot in matlab and SolidWorks. The second aim is to run and check the quad-copter and mobile robot in the real environment. This is actually a small but necessary part of a bigger system that we have planned to implement. That is a Surveillance system which is a combination of a Mobile robot and a quadcopter. The combined system will be fully autonomous. When a target position is given, the mobile robot which carries the quadcopter on-board will start to navigate to the target destination. If the mobile robot get stuck in a dead-end road, it can release the quadcopter which has the ability to find a new path for the mobile robot using its onboard camera. Also, this system can be used for area surveillance. The quadcopter can be used to cover a larger area with less time and faster. If the battery is low or any other problem, the quadcopter can return to the mobile robot platform for charging or repair. Meanwhile, the mobile robot can analyze small areas in detail. This kind of combined surveillance system can be used as a replacement for the Mar Rover robot system.

2.0 Literature survey

Air vehicles can be classified into two categories such as rotary wing and fixed wing aircrafts. Rotary wing air vehicles fly by the thrust force, which is created by propellers. Fixed wing aircrafts creates a thrust force in forward direction. The air, which passes through the wings, obtains a lift force. In recent years, popularity of unmanned air vehicles has been increased. First quadrotor is designed and proposed by De Bothezat and can be seen in Figure 1. The movement of the air vehicle was slow and at low altitudes. Its horizontal motion was affected by wind more than pilot control.

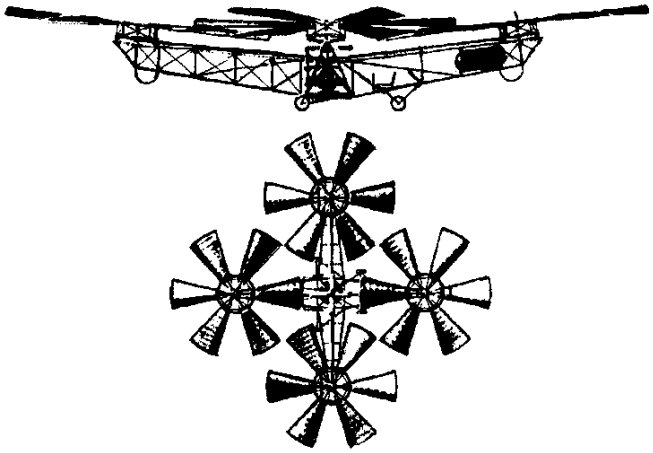


Figure 1. First quadcopter designed by De Bothezat

Different control strategies of rotary wing unmanned air vehicles such as quadrotors have been studied in commercial, academic, and military platforms. Four rotors increase the maneuverability of the vehicle. Having four rotors increases the ability to carry heavy loads. On the other hand, overall power consumption also increases [12]. There are different controller algorithms such as PID, back stepping control, inverse control and sliding mode control which can be applied to the quadrotors [13].

Within the last 5 years, vast amount of new commercial quadcopter have come to the market. Most of them are focused to Ariel photography and videography. What we are trying to implement is a Quadcopter and a mobile robot combined autonomous system. The currently commercially available systems doesn't the ability to land on a place which we desire. With our system, the quadcopter will have the ability to land on any place which the mobile robot can travel. Systems similar to our system seems to be very rare and all the available systems are in heavy research level.

When developing the test system different types of sensors can be used. To measure the altitude of the quadcopter a range of sensors can be used such as, camera, sonar sensors, optical flow sensors, barometers and LiDAR sensor. To determine the X, Y positions of the quadcopter a range of sensors can be used such as Camera, LiDAR, Infrared cameras and sonar. Each of these sensors has their own advantages and disadvantages. Sonar sensor has a very limited distance it can measure and the reflective surface should be smooth to get the most accurate readings. Cameras and optical flow sensors are very sensitive to sun light which means the accuracy will change according to the time of the day and environment. Barometers produce the altitude using the change of air pressure which means it doesn't have the ability to detect different height obstacles underneath the quadcopter when it's hovering. LiDAR sensor work very well in various environments but tend to be expensive.

2.1Previous work

As pointed out in [11] there are problems when using optical flow sensors to detect the landing target in the landing process. Our idea is that these problems can be overcame by using several combinations of sensors and algorithms. Previous work on visually guided takeoff, hovering and landing of UAVs is mainly focused on the landing problem, as it is the most difficult of the three phases. Two principle categories of visual landing frameworks exist for UAVs, which are being used both on large scale helicopters with a high payload and on MAVs with a very limited payload. The first one is for landing a UAV on a predefined target [19][18][17][21], which requires precise pose estimation of the UAV relative to the target. The second category is for landing on a suitable area [16][14], which uses vision for the detection of a good enough place for landing. Saripalli et al. [19] solved the landing task of a helicopter quite early by using image moments for object recognition and estimating the relative position to the landing pad with precise height of the helicopter provided by deferential GPS. But the proposed approach would hardly work in cluttered or GPS-denied environments. A special landing pad with five circle triplets' in different sizes was designed by Merz et al. [18]. And three ellipses in the image were used to estimate the relative pose in a coarse-to-fine process. Lange et al. [17] accomplished autonomous landing and position control of a MAV by evaluating the 3D position from a landing pad consisting of several concentric circles, assuming that the UAV is flying exactly parallel to the landing pad and the ground plane. Wenzel et al. [20] presented a low-cost solution for tracking a landing pad by using infrared LEDs and a Wii remote infrared camera. [21] has achieved takeoff, hovering and landing of an

Ariel vehicle on a moving mobile platform. Xu et al. [22] also used an infrared camera & a coordinated object to estimate the pose of UAVs, though only the yaw angle was calculated.

2.2 Technology Development

2.2.1 Quadcopter

For the control of UAVS there are various methodologies used. Some of the systems that are used for the control of a quad-rotor type flying machine are: reliable feedback controllers which uses fuzzy control, PID controllers, back-stepping controllers, Neural-Network Adaptive Flight Control. [7] studied on the 3-DOF attitude control free-flying vehicle. The trademark to be intensely combined with sources of info and yields, and the genuine nonlinearity show up in the flying vehicle and because of this non-straight control, multi variable control or ideal control for the disposition control of flying Quadcopter.

[8] developed of a non-linear control strategy and a non-linear model for a 6-DOF aerial robot quadcopter. Model deduction includes deciding equations of movement for the Quadcopter in 3 dimensions and searching to estimate actuation forces through modelling of the electric motor dynamics & aerodynamic coefficients.

[9] works on an intelligent fuzzy controller of quadcopter. A fuzzy controller was designed and implemented to test & control a simulation model of the quadcopter. The inputs are the given values of the roll, pitch, yaw and height. The outputs are the power of each of the four motors. Simulation results proves that the efficiency of the intelligent control strategy is acceptable. [10] worked on a research to analyze the dynamic characteristics and PID controller performance of a quadcopter. This paper is describe the construction of a quadcopter and analyzes the dynamic model of it. Besides that, this paper also designs a controller which aim to regulate the aspect of the 6-DOF quadcopter. In our case we are also going to use the PID controller for the matlab simulation.

2.2.2 Mobile Robot

Model-based design is used in various fields of robotics as well. In a Simulink library for the model-based development of robotic manipulators is presented. This Simulink library provides blocks and functions to model kinematics of a robotic manipulator as well as code generation support and verification. In authors use Simulink to simulate motion control loop of a mobile

two-wheeled robot. The results of the simulation helped them identify the proper parameter values for the control system using parameter tuning in the simulation.

[9] proposed a design for a servo motor controller in discrete-time system to determine the transfer function of the PID controller design. MATLAB / Simulink has been used to confirm the effectiveness of this new design method, which provides a simple and powerful way to design a speed controller for servo motor. It also extracted a mathematical model & equations of a DC servo motor and three different motion controllers were designed and simulated to control the velocity of the motor. [29] did a comparison between Fuzzy & PID controllers that were used in mobile robot control. Because a lot of complex operations like requiring fuzzification, inference, and defuzzification were used in the fuzzy controller, it needed more computing time compared to PID controller.

Several mobility configurations can be found in different applications [11]. The most common for single-body robots are differential drive and synchronic drive tricycle or car-like drive and omnidirectional steering [11].

A MATLAB robot test system is utilized to execute the route control calculation and the individual control calculations were recreated utilizing Simulink models. For the route control calculation, the robot test system can move to an objective within the sight of arched and non-curved deterrents. Additionally, a few analyses are performed utilizing a ground robot as a part of a commonplace genuine environment to check the route design calculation modified in MATLAB.

3.0 Statement of the problem

Surveillance robotic systems can be a huge advantage in urban emergency situations, militarily war zones and so on. Currently most of the robotics systems are either mobile vehicle only or Ariel vehicle only. Ground vehicles and Ariel vehicles are operated by separate sets of teams. Since UAV technology has become much more reliable, robust and advanced, they have the ability to take-off, land and take on a completely autonomous mission on a desired path without any human interference. But no matter what there will always be places where the mobile robot can't reach and the Ariel vehicle can't reach if they were to work separately.

But if both of these system can be linked together, the combined system will be much more reliable, robust and advanced compared to the currently available systems. The combined system will get much done during a small period of time. The mobile robot will act as a base station for the Ariel vehicle. The Ariel vehicle can charge its batteries on the mobile platform. It can cover vast area of surveillance with short amount of time and if the mobile robot reach a dead end path, the Ariel vehicle can provide Ariel video which can be used to find a new path for the mobile robot. Even though the Ariel vehicle can cover more ground faster, it won't be able to examine and analyze the ground in detail. But the mobile robot platform will be perfect for examining and analyzing the ground in much more detail and accuracy.

4.0 Aims and objectives of the study

4.1 Aims

The main aim is, a controller is designed for an off the shelf quadcopter to give it the ability to autonomously takeoff, hover at a given altitude, follow and land on a mobile robot platform without any human interface.

4.2 Objectives

I. Complete simulation of the system in Matlab Simulink
II. Implementation of an altitude hold PID controller
III. Implementation of an autonomous takeoff and landing procedures
IV. X, Y coordinates and yaw angle extraction using image processing
V. Implementation of mobile robot platform

5.0 Approach / Methodology

5.1 Approach

The project is divided into two main parts. Those are quadcopter and mobile robot. The mobile robot part is approached by another member of the group. The quadcopter part is also divided into two main parts. Those are quadcopter simulation and real-world implementation. Using Matlab & Simulink software, the altitude hold PID controller algorithm of the quadcopter will be tested.

When implementing the real world system, a sonar sensor will be used to measure the altitude of the quadcopter. These measurements will be used for the altitude hold PID controller. The PID controller is implemented on the secondary controller. To follow the mobile robot, an FPV camera fitted under the quadcopter facing downward is used. The video stream is then sent to a computer. An image processing algorithms running on the PC will determine the position of the mobile robot relative to the camera and the position coordinates are then send to the quadcopter via a wireless RF link. Using those position data, quadcopter follow the mobile robot

Figure 2 describes the overall flow of the project and how all separate parts are connected to each other.

Figure 2. Project flow diagram

5.2 Methodology

The quadcopter real world implementation, an off the shelf Flight controller (KK2) is used as the primary controller for the quadcopter. This primary controller will take care of balancing the quadcopter in mid-air. Using a secondary (Arduino) controller, the radio controller signals will be emulated and inserted to the primary controller. The Mobile Robot will communicate with the secondary (Arduino) controller of the quadcopter. The taking off algorithm is implemented for 80% of height from mobile robot to the altitude hold position it shown as ‘A’ in the Figure 3. At ‘B’ altitude hold controller used to keep the quadcopter at a constant height. For this implementation sonar sensors used to get height feedback.

An axis movement controller implemented to move the quadcopter in either x or y direction from point L1 to point L2 as shown in Figure 3. At ‘C’ using image processing technique for quadcopter to follow the mobile robot in x or y direction. Finally at ‘D’, a landing procedure is implemented to safely land the quadcopter on the mobile robot.

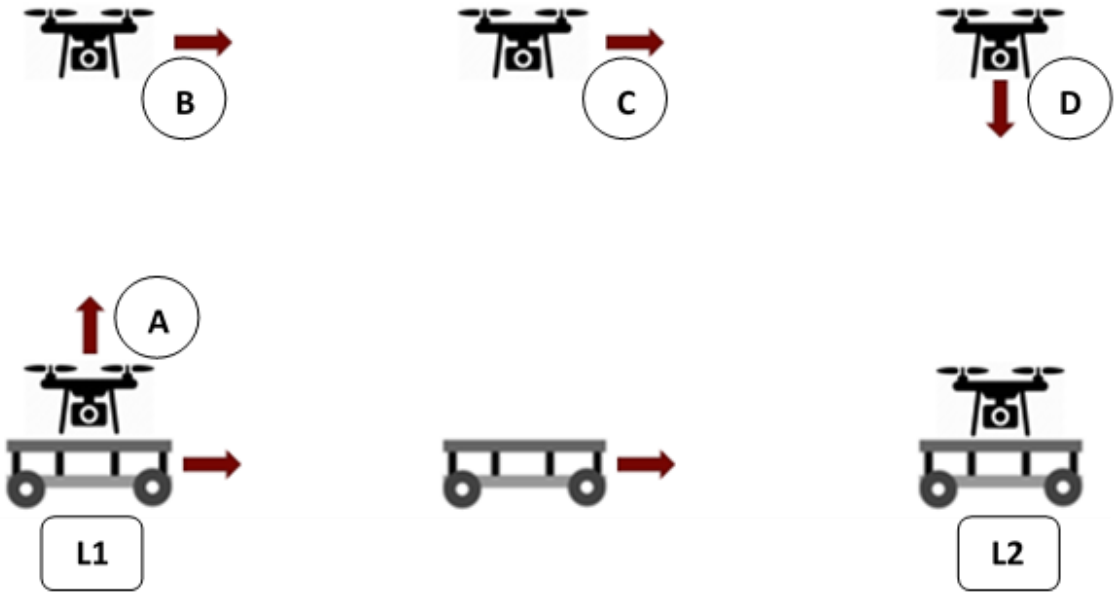


Figure 3: Overview of the project

5.2.1 Matlab Simulation

The matlab based simulation done by two methods. Those methods are mathematical model and simulink model.

5.2.1.1 Mathematical model

Mathematical model portrays quadcopter movement and behavior with the regard to the input information of the model and external impacts on quadcopter. Mathematical model can be seen as a capacity that is mapping inputs on outputs. By utilizing mathematical model, it is conceivable to predict position and attitude of quadcopter by knowing the four angular velocities of propellers and also it empowers the computer simulation of quadcopter behavior in various conditions. Computer simulation is moderately straightforward, cheap and safe technique for control algorithm check.

The Figure 3 shows that the steps of how the mathematical modelling was model in matlab. The first step of the mathematical modelling is derive the equation of the motion then modeling the quadcopter dynamic model. After modeling the dynamic model the output of dynamic model is feedback through PID controller and store in matrix and plot in 3D animation.

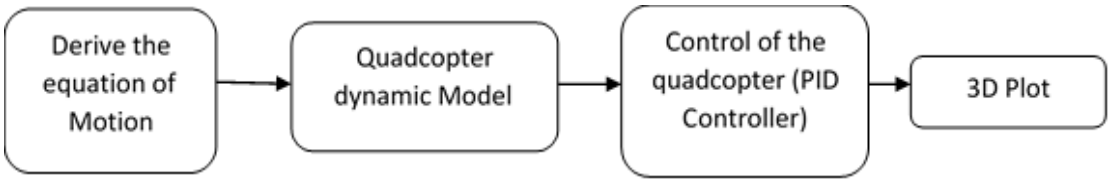


Figure 4: Flow of mathematical modelling

5.2.1.2 Matlab simulink model

The quadcopter used in the project has a DJI F330 frame. The propellers were 8x4.5 inch size and the motors were 1000 kV brushless motors. These parts were first 3D drawn in Solid Works software and then the STL file is imported to the matlab Simulink software for the simulation. Importing each part separating to the Simulink allowed us to model and control the system easily and preciously. The Figure 5 shows the full system simulation block diagram.

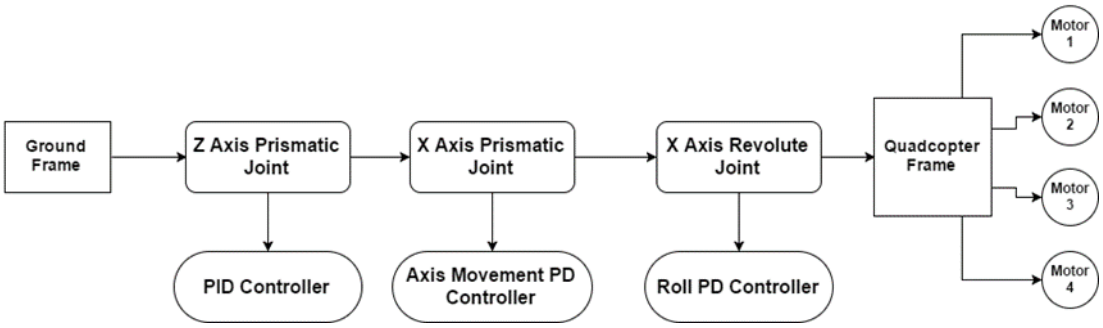


Figure 5: Full System simulation block diagram

5.2.2 Experimental System

5.2.2.1 Quadcopter

The flight controller (KK2) is used as the primary controller for the quadcopter. This primary controller will take care of balancing the quadcopter in mid-air. The secondary controller consists with an Arduino Mega 2560 development platform. All the custom implementations will be done with this platform. A quadcopter radio receiver is connected to the secondary controller to take control of the quadcopter at emergency situations. A sonar sensor is used to measure the altitude of the quadcopter at all times with accuracy. Using the PID controller, throttle values will be calculated to move up, down and hover around the set altitude. The axis movement controller will work with the FPV camera fitted underneath the quadcopter. The camera feed will be received by a PC which is running an image processing algorithm which will calculate the position of the quadcopter relative to a 'red' square on the ground. Using the position data from the image processing algorithm, the secondary controller will try to keep the quadcopter centered above the 'red' square. When the 'red' square is fitted to the mobile robot, the quadcopter will move along with the mobile robot.

5.2.2.2 Mobile robot

The raspberry pi based mobile robot implemented for act as the base station of the quadcopter. The quadcopter takeoff and land on the helipad which placed on the mobile robot. Also the quadcopter will follow the mobile robot using the FPV camera facing downward in the quadcopter. The camera video stream is sent to a raspberry pi. Then image processing algorithms running on the raspberry pi will determine the position of the mobile robot relative to the camera and the position coordinates will send to the quadcopter via a wireless RF link. Along with the position data, the quadcopter will follow the mobile robot which moves at constant velocity.

6.0 Relevant Theory and Analysis

6.1 Quadcopter Model

Figure 6 shows the main dynamic parameters of a quadcopter. The quadcopter is modeled with four motors in a cross formation. This cross structure is robust and reliable. The rotation axes of all motors are fixed and parallel. Furthermore, the propellers have fixed-pitch blades and their air flows points downwards (to get an upward lift). These considerations shows that the structure is rigid and the only things that can change are the propeller speeds.

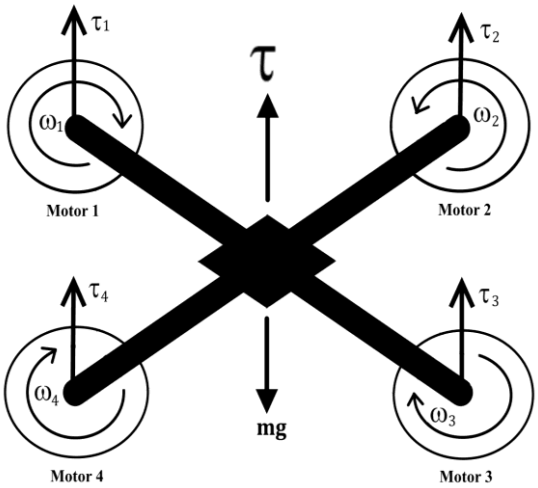


Figure 6. Quadcopter dynamics

The side of motor 1 & 2 are considered to be the front of the quadcopter. Motor 1 & 3 rotates in Clockwise direction and motor 2 & 4 rotates in Counterclockwise directions. This configuration0of opposite pair's directions removes the need for a0tail rotor (which are needed in a standard0helicopter structure). Even7though the0quadcopter has 6 DOF, it is equipped with only four8propellers, because of that it is not possible to reach a desired9position in 3D space for all the DOF, but four at maximum. However, thanks to its structure, it is quite easy to choose the four best4controllable variables and to decouple them5to make the controller easier. The four quadcopter6targets are thus related to the four basic movements2which allow the helicopter to3reach a certain height and attitude. It follows the1description of these basic movements of Roll, Pitch, Yaw and Altitude.

6.1.1 Throttle (N)

This command is given by changing all the propeller speeds by the similar amount. It prompts to a vertical force body-fixed frame which raises or brings down the quadcopter. In the event that the helicopter is in horizontal position, the vertical direction of the inertial frame and one

of the body fixed frame coincide. Generally the provided thrust generates both vertical and horizontal accelerations in the inertial frame. Figure 7 explains the throttle command on a quadcopter sketch.

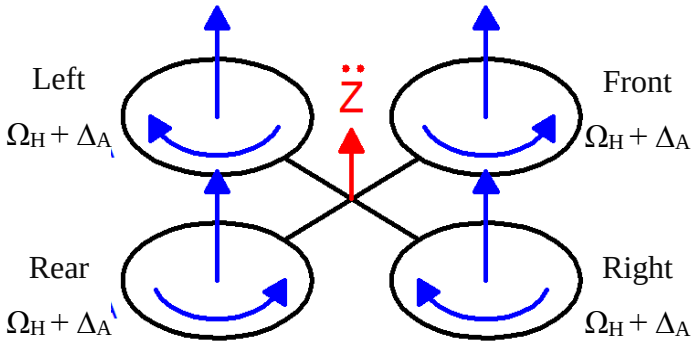


Figure 7: Throttle movement

In Figure 7 blue it is determined the speed of the propellers which, for this situation, is equivalent to $(\Omega_H + \Delta_{A)}$ for each one. Δ_A (rad s^{-1}) is a positive variable which represents an addition regard of the constant Ω_H. Δ_A can't be too huge because the model would in the long run be impacted by strong non linearity or saturations.

6.1.2 Roll (N m)

This command is given by increasing the left propeller speed and by decreasing the right propeller or vice versa [4]. It prompts to a torque with respect to the x_B axis which makes the quadcopter turn. The overall vertical thrust is the same as in hovering, subsequently this command drives just to a roll angle acceleration. Figure 8 shows the roll command on a quadcopter sketch.

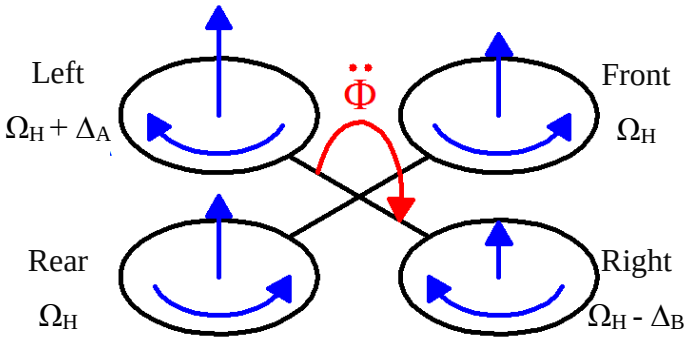


Figure 8: Roll movement

The positive factors Δ_A and Δ_B [rad s−1] are kept up the vertical thrust unchanged. It can be shown that for little values of Δ_A, $\Delta_B \approx \Delta_A$. As in the past case, they can't be too huge because the model would in the end be impacted by strong non linearity or saturations.

6.1.3 Pitch (N m)

This command is fundamentally the same as the roll and is provided by increasing the rear propeller speed and by decreasing the front one or vice versa. It prompts to a torque with regarding to the y_B axis which makes the quadcopter turn [5]. The general vertical thrust is the similar as in hovering, consequently this command leads only to a pitch angle acceleration. Figure 9 shows the pitch command on a quadcopter sketch.

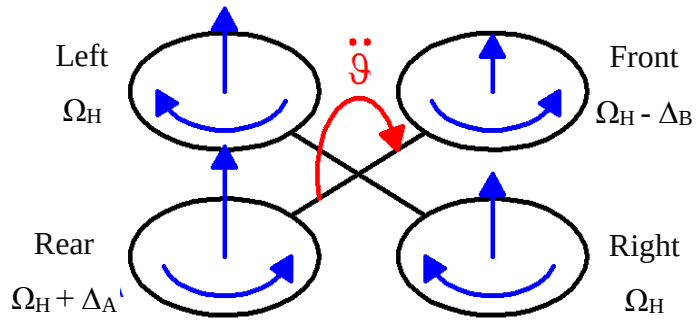


Figure 9: Pitch movement

As in the before case, the positive variables ΔA and ΔB are kept up the vertical thrust unchanged and they can't be too extensive. Moreover, for small values of ΔA, it occurs $\Delta B \approx \Delta A$.

6.1.4 Yaw (N m)

This command is given by increasing the front-rear propellers' speed and by decreasing that of the left-right couple or vice versa. It prompts to a torque regarding the z_B axis which makes the quadcopter turn. The yaw movement is produced to the fact that the left-right propellers rotate clockwise while the front-rear ones rotate anticlockwise [4][5]. Consequently, when the overall torque is uneven, the helicopter turns on itself around Z_B. Figure 10 shows the yaw command on a quadcopter sketch. As in the past two cases, the positive variables Δ_A and Δ_B are kept up the vertical thrust unchanged and they can't be too vast. Besides it keeps up the equivalence $\Delta_B \approx \Delta_A$ for small values of Δ_A.

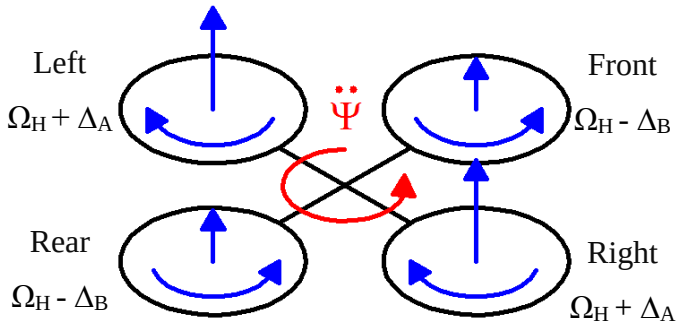


Figure 10: Yaw movement

6.2 PID controller

A proportional–integral–derivative controller [23] is a control loop feedback controller commonly used in robots, industrial control systems and many other systems. A PID0controller continuously0determine an error value e(t) which0is the difference between0the desired set-point and a measured0process variable value and applies a correction0based on proportional, integral, and derivative terms, (which are denoted as P, I, and D respectively) which provides the controller name. The advantage of using PID controller over other controllers is that plant model is not needed to get a stable output. Figure 11 clearly shows the basic block diagram of a PID controller.

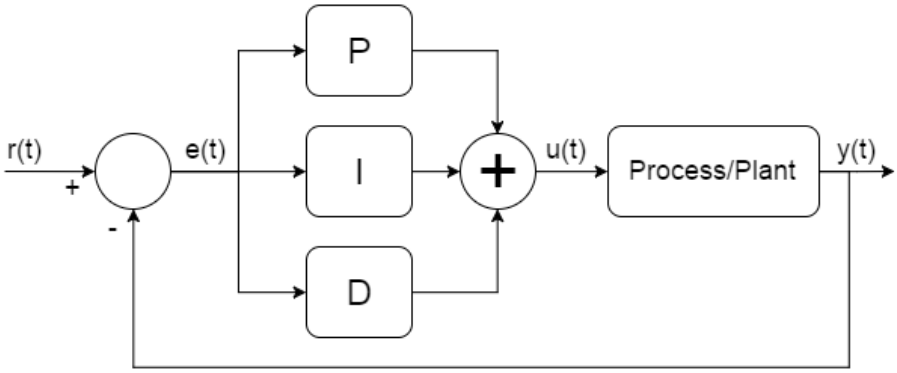


Figure 11. PID controller

Equation (1) expressed below the mathematical formula of a continuous domain PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d(e(t))}{dt} \qquad (1)$$

Where u is a generic controlled variable, e is the error between the desire state and the process output y, K_P is the proportional coefficient, K_I is the integral coefficient and K_D is the derivative coefficient.The primary contribute (P) is proportional to the error and define the proportional bandwidth. Inside this interval the output will be proportional to the error while outside the output will be least or greatest. The second contribute (I) fluctuates according to the integral of the error. Despite the fact that this component increases the overshoot and the settling time, it vanishes the steady state error. The third contribute (D) fluctuates according to the derivate of the error. This segment diminish the overshoot and the settling time.

Table 1. Effects of PID coefficients [25]

CL Response	Rise Time	Overshoot	Settling Time	Steady-State Error
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Small Change	Decrease	Decrease	No Change

Few main parameters are used to determine the performance of a control system. They are rise time, overshoot, settling time and steady-state error. Where u is a generic controlled variable, e is the error between the desire state and the process output y, KP is the proportional coefficient, KI is the integral coefficient and KD is the derivative coefficient.The primary contribute (P) is proportional to the error and define the proportional bandwidth. Inside this interval the output will be proportional to the error while outside the output will be least or greatest. The second contribute (I) fluctuates according to the integral of the error. Despite the fact that this component increases the overshoot and the settling time, it vanishes the steady state error. The third contribute (D) fluctuates according to the derivate of the error. This segment diminish the overshoot and the settling time.

Table 1, clearly shows how the PID controller coefficients effects the above mentioned system parameters.

Figure 12 shows the effects of the PID controller coefficients as a step response.

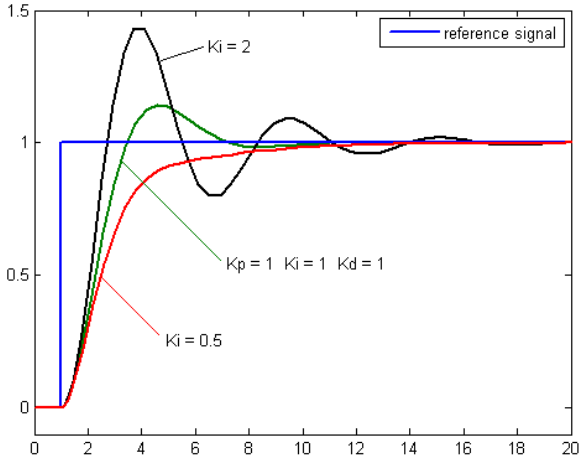


Figure 12. Effects of PID controller coefficients

6.3 Kalman Filter

The Kalman filter has the structure of a standard state observer, as illustrated in Figure 13. The contrast between the measured system output and the estimated system output is scaled by the Kalman filter gain 'K' and feedback to the observer. The Kalman fillter gain 'K' itself, is the solution of an optimization problem under the assumption that the process and measurement noise are uncorrelated white noise signals. I.e. the gain matrix, that minimizes the expectation of the estimation error, is chosen. With respect to the linear quadratic regulator problem, the ideal solution for the Kalman optimization problem is found by solving a discrete algebraic Riccati equation.

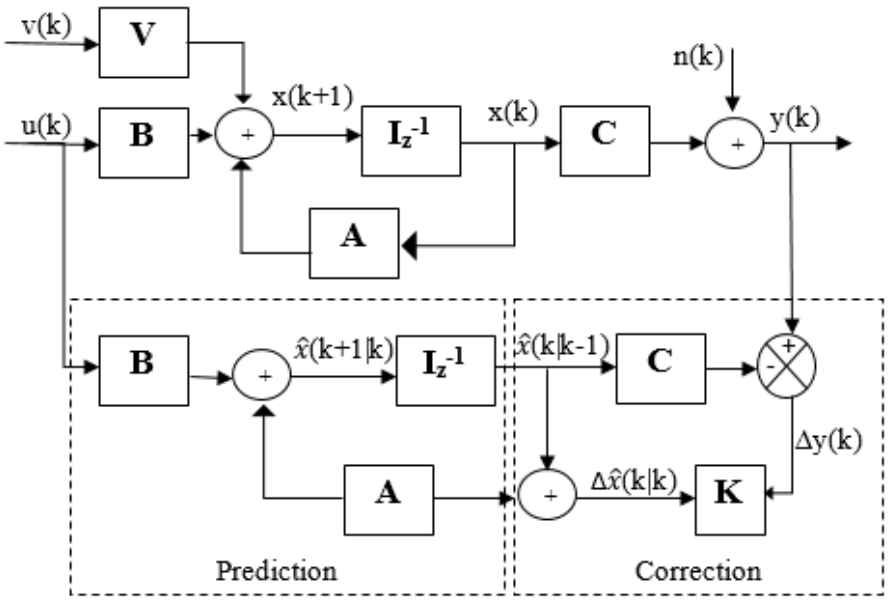


Figure 13: Kalman Filter schematic diagram

Consider the following system, where A, B, C, V are known matrixes and n(k) and v(k) are zero mean stochastic noises at the input and at the output with covariance matrix N.

$$\mathbf{x}(k+1) = \mathrm{A}\,\mathbf{x}(k) + \mathbf{B}\,\mathbf{u}(k) + \mathrm{V}\,\mathbf{v}(k) \qquad \text{---------- (2)}$$

$$\mathbf{y}(k) = \mathrm{C}\,\mathbf{x}(k) + \mathbf{n}(k) \qquad \text{---------- (3)}$$

Then Kalman Filter can be defined in the following form:

$$\underset{\text{new estimate}}{\hat{\mathbf{x}}(k+1|k+1)} = \underset{\text{old estimate}}{\mathrm{A}\,\hat{\mathbf{x}}(k|k)} + \mathbf{B}\,\mathbf{u}(k) + \underset{\text{correction matrix}}{\mathbf{K}(k+1)}\ [\underset{\text{new measure-ment}}{\mathbf{y}(k+1)} - \underset{\text{predicted measurement based on old estimate}}{\mathbf{C}(\mathrm{A}\,\hat{\mathbf{x}}(k|k) + \mathbf{B}\,\mathbf{u}(k))}] \qquad \text{----- (4)}$$

State estimation:

$$\textit{prediction:} \quad \hat{\mathbf{x}}(k+1|k) = \mathrm{A}\,\hat{\mathbf{x}}(k|k) + \mathbf{B}\,\mathbf{u}(k) \qquad \text{---------- (5)}$$

$$\textit{correction:} \quad \hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}[\mathbf{y}(k+1) - \mathrm{C}\,\hat{\mathbf{x}}(k+1|k)]$$

---------- (6)

Correction matrix:

$$\mathbf{K} = \mathbf{P}\,\mathbf{C}^{T}[\mathbf{C}\,\mathbf{P}\,\mathbf{C}^{T} + \mathbf{N}]^{-1} \quad \text{---------- (7)}$$

Where P is covariance matrix of the estimation error and N is the covariance matrix of the output noise.

$$\mathbf{P}\,(k+1) = E\left\{\tilde{\mathbf{x}}(k+1|k)\,\tilde{\mathbf{x}}^{T}(k+1|k)\right\} \quad \text{---------- (8)}$$

$$\tilde{\mathbf{x}}(k\,|\,j) = \mathbf{x}(k) - \hat{\mathbf{x}}(k\,|\,j) \quad \text{---------- (9)}$$

$$\mathbf{N} = E\left\{\mathbf{n}(k)\,\mathbf{n}^{T}(k)\right\} \quad \text{---------- (10)}$$

6.4 Image Processing Theories

6.4.1 Morphological Transformation

While non-linear filters are often used to enhance grayscale and color images, they are also used extensively to process binary images. Such images often occur after a thresholding operation,

$$\theta(f,t) = \begin{cases} 1 & \text{if } f \geq t, \\ 0 & \text{else,} \end{cases} \quad \text{---------- (11)}$$

e.g., converting a scanned grayscale document into a binary image for further processing such as optical character recognition.

The most common binary image operations are called morphological operations, since they change the shape of the underlying binary objects. To perform such an operation, first convolve the binary image with a binary structuring element and then select a binary output value depending on the threshold result of the convolution. The structuring element can be any shape, from a simple 3 X 3 box filter, to more complicated disc structures. It can even correspond to a particular shape that is being sought for in the image. [10] [11]. Figure 14 shows the binary Image morphology of a letter.

Figure 14: Binary image morphology: (a) original image; (b) dilation; (c) erosion

Equation (12) shows a close-up of the convolution of a binary image f with a 3 X 3 structuring element s and the resulting images for the operations described below. Let

$$c = f \otimes s \qquad \text{---------- (12)}$$

be the integer-valued count of the number of 1s inside each structuring element as it is scanned over the image and S be the size of the structuring element (number of pixels). The standard operations used in binary morphology include:

- dilation: dilate (f, s) = Θ (c; 1);
- erosion: erode (f, s) = Θ (c; S);
- majority: maj(f,s) = Θ (c, S/2);
- opening: open (f, s) = dilate (erode (f, s), s);

6.4.1.2 Erosion and Dilation

The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e.:

- Removing noise
- Isolation of individual elements and joining disparate elements in an image.
- Finding of intensity bumps or holes in an image

Dilation operations consist of convoluting an image A with some kernel (B), which can have any shape or size, most of the time it is a square or circle. The kernel B has a defined stay point, normally being the center of the kernel. As the kernel B is scanned over the image, we determine the maximal pixel esteem overlapped by B and replace the image pixel in the stay point position with that maximal value. As you can find, this amplifying operation causes bright regions within an image to “develop” (therefore the name dilation). Take for instant the image below. By using dilation, we can get the background enlarge around the dark regions of an object.

Figure 15: (a) original image (b) dilated image

This operation is the sister of dilation. What this does is to determine a least over the region of the kernel. As the kernel is scanned over the image, we determine the least pixel value overlapped by and replace the image pixel under the stay point with that minimal value. Similarly, for the instant for dilation, we can apply the erosion operator to the original image (shown below). You can see in the outcome below that the bright areas of the image background get thinner, through the dark zones get bigger [11]. Figure 16 shows the Original image vs Eroded Image.

Figure 16: (a) Original Image (b) Erode Image

6.4.2 Non-Linear Image Filtering

6.4.2.1 Gaussian blur

The so-called blur can be understood as taking a pixel as the average value of its surrounding pixels.

1	1	1
1	2	1
1	1	1

Figure 17: pixel as the average value of its surrounding pixels

On the above Figure 17, 2 is the center point, the surrounding points are 1.

1	1	1
1	1	1
1	1	1

Figure 18: loosing center point

The center point will take the average value of its surrounding points, it will be 1. From value perspective, it's a smoothing. On graphic, it's a blur effect. As shown in the Figure 18, the center point will lose its detail.

Weight of normal distribution:

Figure 19 shows the Normal distribution is an acceptable weight distribution model.

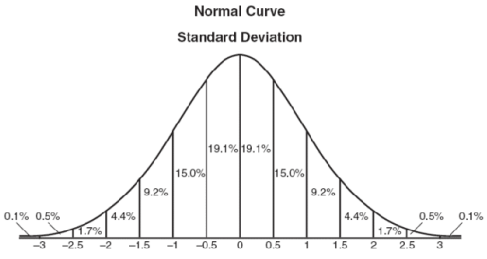


Figure 19: weight normal distribution

On graphic, normal distribution is a Bell-shaped curve, the closer to the center, the bigger the value.

Gaussian function:

The normal distribution above is one dimensional, the graph shown in **Error! Reference source not found.** is two dimensional. We need two-dimensional normal distribution.

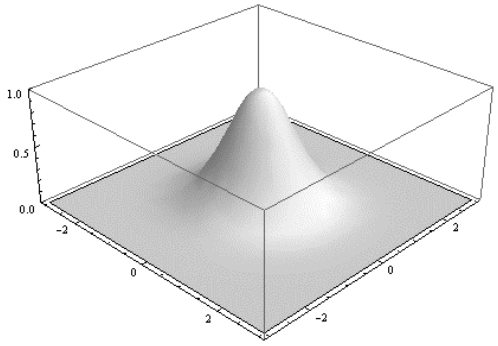


Figure 20: density function

The density function of normal distribution is called Gaussian function. The one dimension format is,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \qquad \text{---------- (13)}$$

Here μ is the average of x, because center point is the origin point when calculating average value, so μ equals to 0. Equation (14) is a derivation of Gaussian function when μ = 0.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2} \qquad \text{---------- (14)}$$

Based on the one dimension function, we can derive the two-dimensional Gaussian function.

Equation (15) expressed below is the two-dimensional Gaussian function

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad \text{---------- (15)}$$

With this function, we can calculate the weight of each point.

Weight matrix:

Assume the coordinate of the center point is (0,0), then the coordinates of 8 points which are nearest to it are shown in Figure 21.

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

Figure 21: the coordinates of 8 points

To calculate the weight matrix as shown in the Figure 22, we need to set the value of σ, σ=1.5, then the weight matrix of blur radius 1 is,

0.0453542	0.0566406	0.0453542
0.0566406	0.0707355	0.0566406
0.0453542	0.0566406	0.0453542

Figure 22: weight matrix calculation

Figure 23 shows the sum of the weights of these 9 points is 0.4787147. If only calculate the Weighted average of these 9 points, then the sum should be 1, hence the above 9 values should divide 0.4787147.

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

Figure 23: Sum of weights

Calculate Gaussian Blur:

With weight matrix shown in Figure 24, we can calculate the value of Gaussian Blur. Assume we have 0 pixels now, the gray value (0-255):

14	15	16
24	25	26
34	35	36

Figure 24: weight matrix

Figure 25 shows each point multiplies its weight value:

14x0.0947416	15x0.118318	16x0.0947416
24x0.118318	25x0.147761	26x0.118318
34x0.0947416	35x0.118318	36x0.0947416

Figure 25: multiplication with weight value

Now we have resultant weight matrix as shown in Figure 26:

1.32638	1.77477	1.51587
2.83963	3.69403	3.07627
3.22121	4.14113	3.4107

Figure 26: resultant weight matrix

Add these 9 values up, then you will get the Gaussian Blur value of the center point. Repeat this process for all other points, then you will get graph after Gaussian blur. The example graphs shown in Figure 27 show the original, 3 pixels' blur radius and 10 pixels' blur radius. The bigger the blur radius, the more blur the picture is. [12] [13]

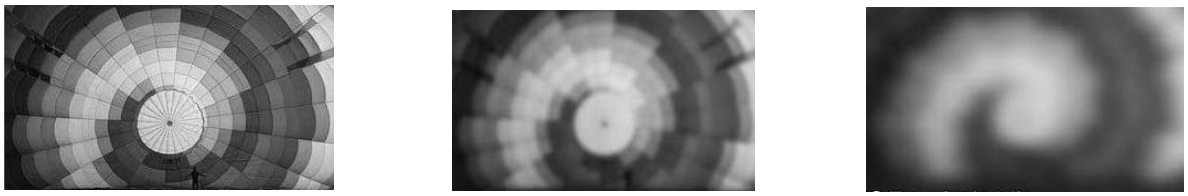


Figure 27: the original, 3 pixels' blur radius and 10 pixels' blur radius

6.4.3 Canny Edge Detector

The Canny Edge detector was developed by John F. Canny in 1986. Also, known to many as the optimal detector, Canny algorithm aims to satisfy three main criteria: [14]

- **Low error rate:** Meaning a good detection of only existent edges.
- **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimized.
- **Minimal response:** Only one detector response per edge.

Steps,

1. Filter out any noise. The Gaussian filter is used for this purpose.
   Equation (16) expressed below an example of a Gaussian kernel of size = 5.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \quad \text{---------- (16)}$$

Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

a) Apply a pair of convolution masks (in x and y directions):

Equation (17) expressed below the mathematical model of convolution masks for x and y direction to find the intensity gradient of the image.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \qquad \text{---------- (17)}$$

b) Find the gradient strength and direction with:
Equation (16) expressed below the Gradient and the direction

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan(\frac{G_y}{G_x}) \qquad \text{---------- (18)}$$

The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)

2. *Non-maximum* suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines (candidate edges) will remain.
3. *Hysteresis*: The final step. Canny does use two thresholds (upper and lower):

a) If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge
b) If a pixel gradient value is below the *lower* threshold, then it is rejected.
c) If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

Canny recommended an upper: lower ratio between 2:1 and 3:1.

After compiling the code above, we can run it giving as argument the path to an image. For example, as shown in Figure 28 shows an input the image and canny edge detection output.

Figure 28: input image, output after canny edge detection

7.0 Design and Implementation

7.1 Quadcopter Simulation

7.1.1 Mathematical Model

As the quadcopter is moving in 3 dimensions and can rotate around every one of the three axes. The quadcopter structure is introduced in Figure 29 including the corresponding angular velocities (ω), torques (Tm) and forces (f) created by the four rotors. As observing the quadcopter from the Earth then it is most sensible to reference everything to the Earth frame of reference. The inertial frame is characterized by the ground, with gravity indicating in the negative z direction. The body frame is characterized by the orientation of the quadcopter, with the rotor axes indicating in the positive z direction and the arms indicating in the x and y directions.

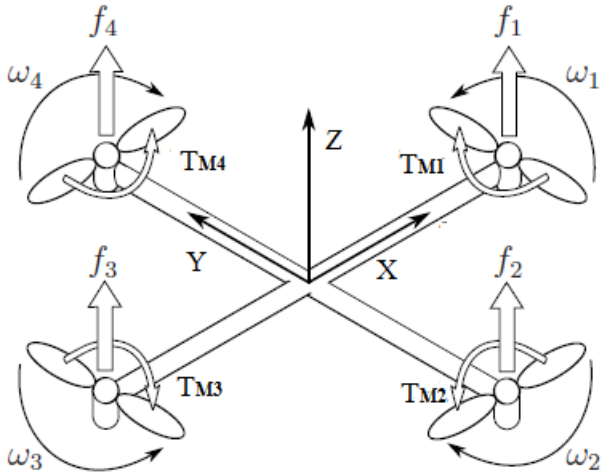


Figure 29: The inertial and body frames of a quadcopter

The Figure 30 block diagram explain the main functions wrote for mathematical simulation.

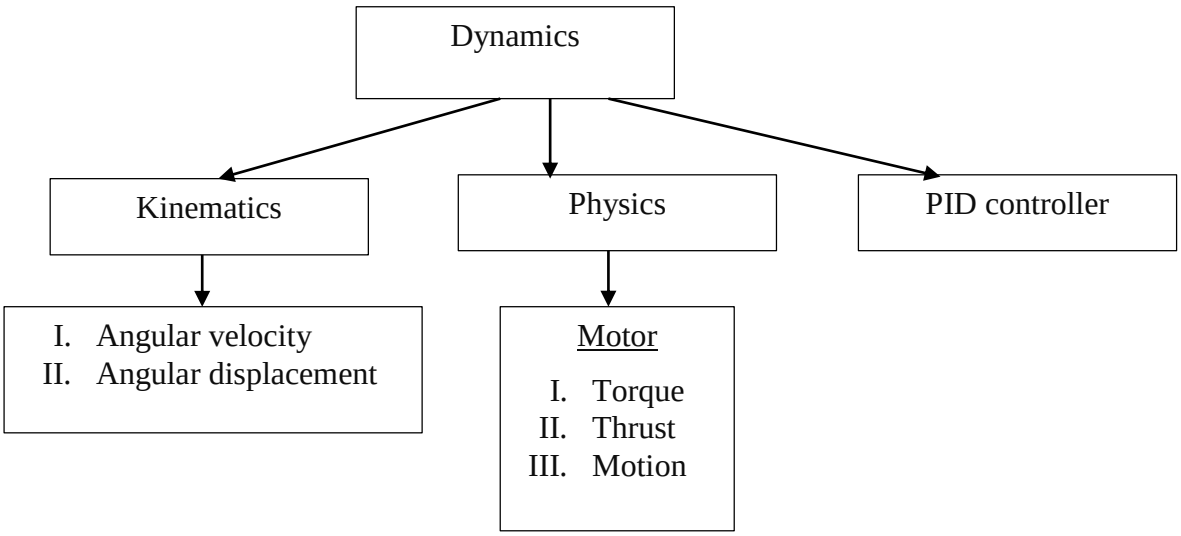


Figure 30: Flow char of Mathematical model functions

7.1.2 Simulink Model

7.1.2.1 F330 Quadcopter model

Figure 31, shows the 3D model of the F330 quadcopter drawn in Simulink application using the part exported from the SolidWorks application. Figure 32 shows the quadcopter on the mobile robot.

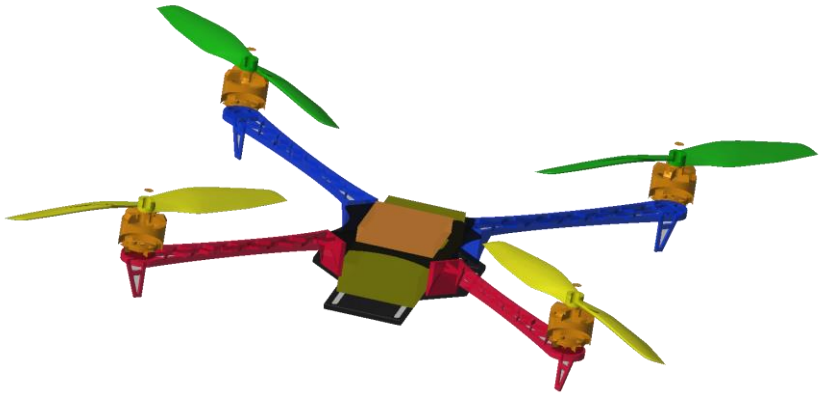


Figure 31: F330 quadcopter model in Simulink

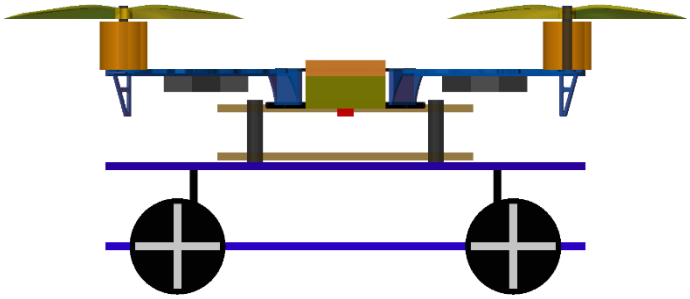


Figure 32. Quadcopter on the mobile robot

Simulation of the quadcopter describes how the system perform in an ideal environment. To obtain accurate results, the simulation model of the system should be an exact replica of the actual system. To achieve this, all the quadcopter parts were separately modeled using SolidWorks and imported to Simulink. The advantage of SolidWorks is that it provides all the internal parameters of a 3D part such as inertias around each X, Y & Z axes.

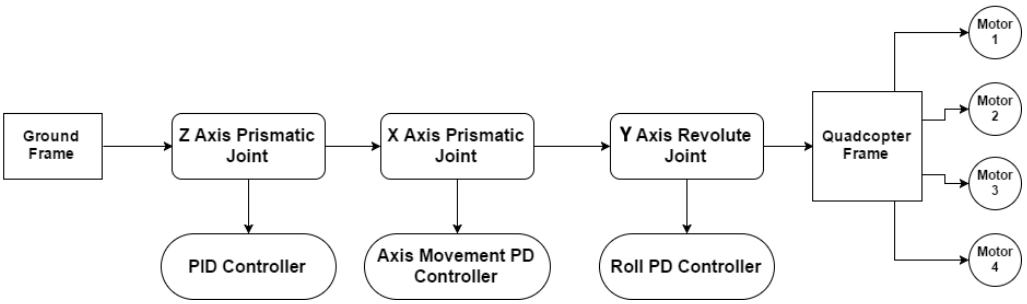


Figure 33. Full System simulation block diagram

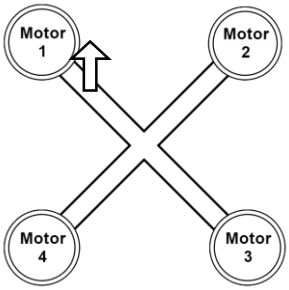


Figure 34. Quadcopter Motor layout

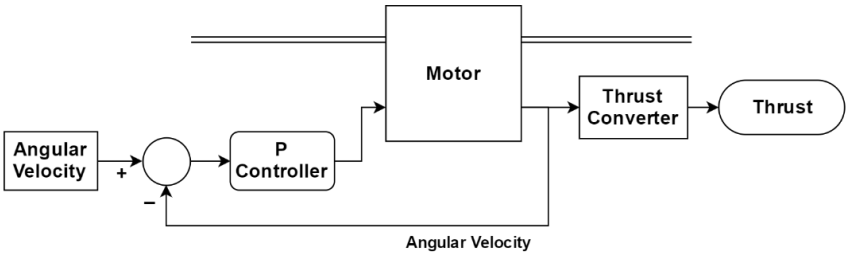


Figure 35. P controller for each motor

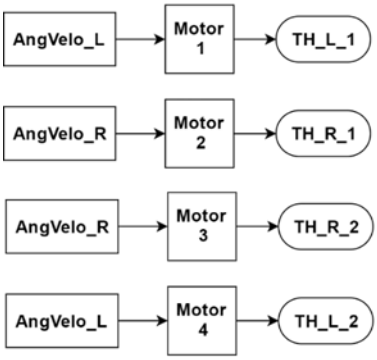


Figure 36. Each motor input and output parameters

Figure 33 shows a simple block diagram describing how the full system simulation was done in Matlab Simulink. There are three main joint in the full system. They are a prismatic joint which only moves the quadcopter through Z axis, another prismatic joint which moves the quadcopter through X axis and the revolute joint which turns the quadcopter around Y axis.

Three controller are implemented to control the above mentioned joints accordingly. Figure 34 shows the quadcopter motor layout used in this project.

Each propeller is connected to a revolute joint so it can spin around Z axis. When torque is applied, there will be an angular acceleration on the revolute joint and the angular velocity of the propeller will continue to increase. This doesn't happen in real world motors. The reason for this is that the joints in Simulink are ideal. There's no friction on the joints. To get rid of this problem and keep the propeller spinning at constant speed, a P controller is for each revolute joint. The inputs for the P controller is angular velocity. Figure 35 describes the use of the P controller to turn the motors at a constant angular velocity and how a single motor produces the thrust. Figure 36 shows the inputs for each motor and the output thrust produced by them.

7.1.2.2 Altitude hold PID controller

Figure 37 shows the simplified block diagram of the Simulink simulation of the altitude hold PID controller.

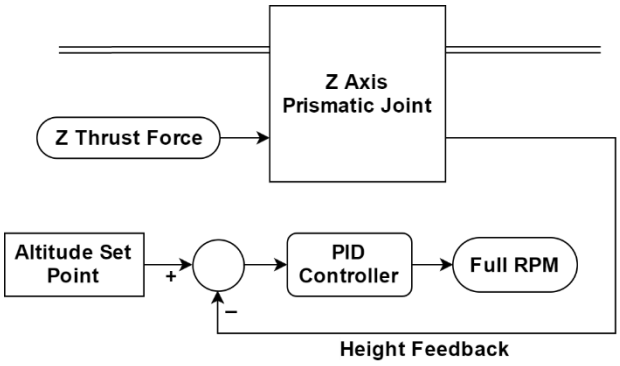


Figure 37: Altitude hold PID controller diagram

Using the (19) formula, the RPM of each motor is converted to thrust value, which is used to calculate the full thrust from all four motors.

[24] $$F = 4.392399 * 10^{-8} * RPM * \frac{d^{3.5}}{\sqrt{pitch}} * (4.23333 * 10^{-4} * RPM * pitch) \quad (19)$$

The whole quadcopter model is connected to the ground plain using a prismatic joint. This will allows the quadcopter to move up and down through Z axis. The prismatic joint will produce position of the joint as a feedback and it will take force through Z axis as input.

To implement the altitude hold PID controller, angular velocity for each motor should be calculated, so that the combined thrust of all motors moves the quadcopter up until it reaches the desired altitude set point. The combined thrust of all motors is used as force input for the prismatic joint. The feedback for the PID controller is the position of the prismatic joint which is also the position of the quadcopter through Z axis. The calculated output of the PID controller is the total RPM required.

7.1.2.3 Roll PD controller

Quadcopters move in 3D space by tilting around a certain axis for a certain amount of angle. When the quadcopter is hovering, it should always be kept horizontal (0 degrees) around both X and Y axes. To maneuver the quadcopter this manner, a PID controller is need which has the ability to keep the quadcopter at a given set point angle around a certain axis. Since an off-the shelf flight controller is used in the prototype system, it will have the roll pitch PID controller. But for the simulation, this PID controller should be implemented separately. This will allow the quadcopter to be moved through X axis later.

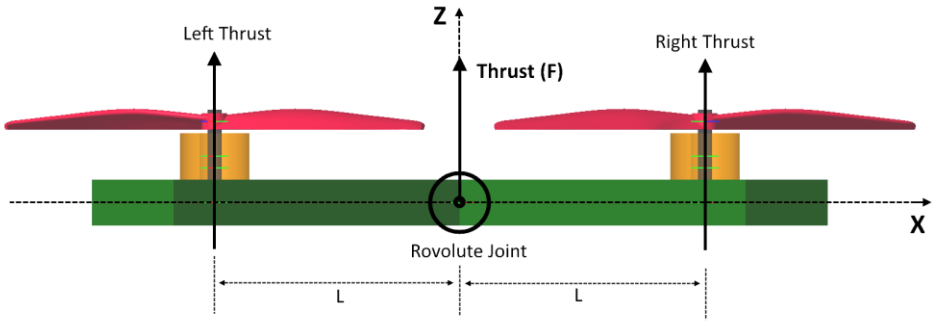


Figure 38. Forces acting on the quadcopter

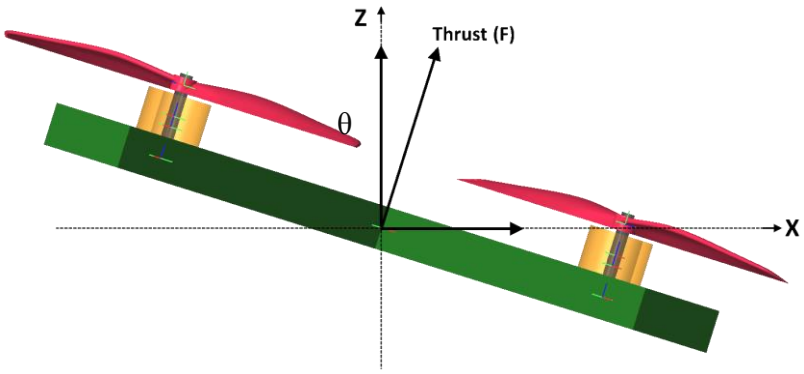


Figure 39. Quadcopter orientation when moving in X direction

When the quadcopter needs to move in X direction, the quadcopter should tilt around Y axis. In Figure 39, resultant force in Z direction ($F * \cos(\theta)$) will keep the quadcopter hovering. The resultant force in X direction ($F * \sin(\theta)$) will help the quadcopter move in X direction.

The PD controller will take desired angle as the input & angle of the revolute joint as feedback. The output of the PD controller is the RPM devotion which is needed to keep the quadcopter at the desired angle.

In the simulation, a revolute joint is added at the center of the quadcopter as shown in Figure 38 so it can turn around the Y axis. To move the revolute joint, a torque input should be inserted. The resultant torque which acts on the revolute joint is calculated using the left thrust by right thrust values as in Figure 41.

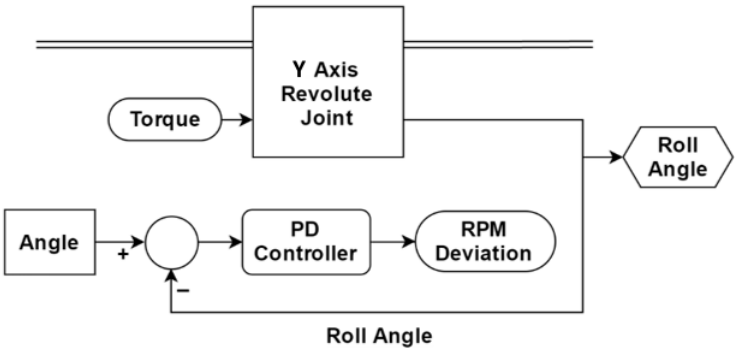


Figure 40. Simulink diagram for Roll PID controller

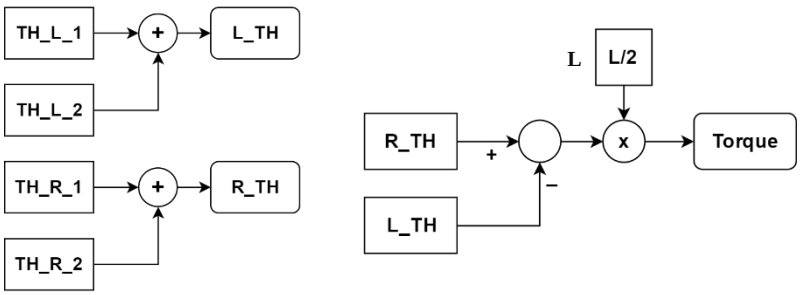


Figure 41. Simulink diagram for torque calculation

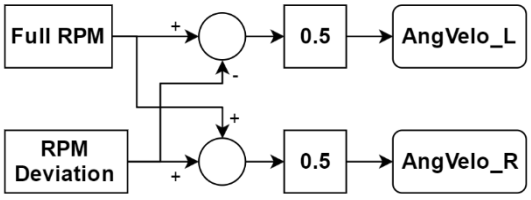


Figure 42. New angular velocity calculation for motors

Figure 40 shows the simulation block diagram of the Roll PD controller implemented in Simulink. Figure 41 shows how the torque calculation is done which is inserted to the to the X axis revolute joint. Figure 42 describes how the angular velocity of each side motors are calculated using the full RPM and the RPM deviation from the Roll PD controller.

7.1.2.4 Axis movement controller

The axis movement controller allows the quadcopter to move through X axis to any given set point. A prismatic joint and a PD controller is used for this purpose. The input for the PD controller is the X direction desired position for the quadcopter. Position of the prismatic joint is used as the feedback for the PD controller. The output of the PD controller will be the angle (in degrees) which the quadcopter should tilt to move in X direction. To move the quadcopter in X direction, the $F * \sin(\theta)$ resultant force was inserted to the prismatic joint as shown in Figure 39.

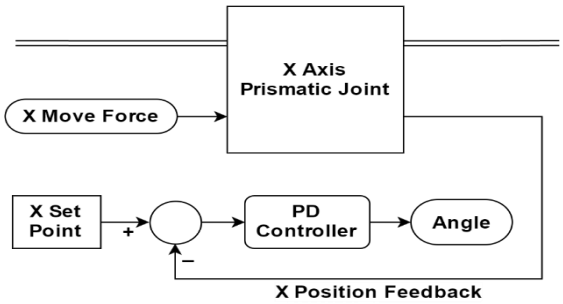


Figure 43. X axis movement controller

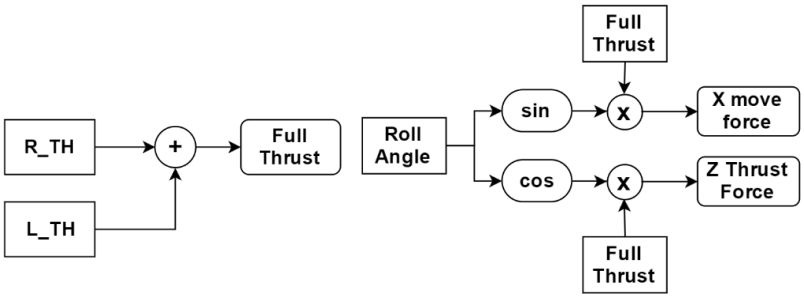


Figure 44. X direction force calculation

Figure 43 describes the simplified block diagram of the X axis movement controller implemented in Simulink. Figure 44 shows how the X & Z direction resultant forces were calculated.

7.2 Prototype testing system implementation

7.2.1 Design of the Test system

7.2.1.1 Quadcopter system with the secondary controller

Figure 45 shows the quadcopter platform implemented for testing and further study. The quadcopter testing platform consists with various sensors, communication peripherals and controllers which are need to implement and test different algorithms and controllers. The testing platform was built by adding a custom made stand to an off the shelf DJI F330 frame. The frame itself has enough space to hold a battery, flight controller and a radio receiver. The custom made stand has two main purposes. One is to hold all the sensors, controllers and other peripherals needed for testing. The second purpose is to add a cushion layer to the bottom of the quadcopter, so that at an emergency landing situation, the impact won't damage or harm the test platform.

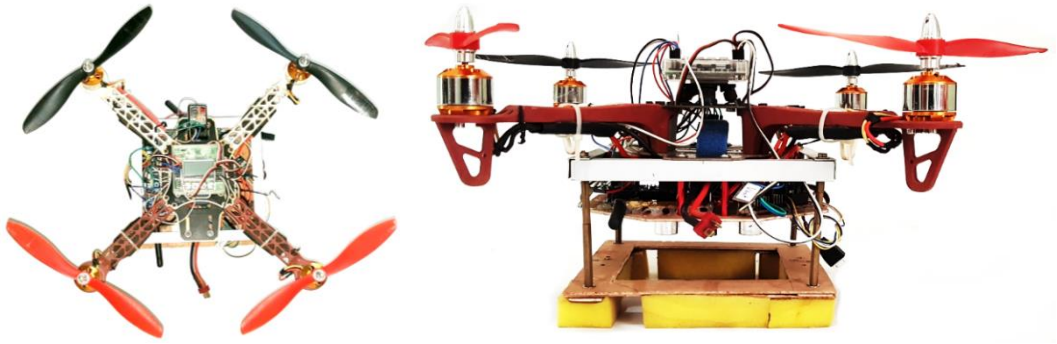


Figure 45. Quadcopter testing prototype

7.2.1.2 Altitude hold PID controller

In simulation, the Z axis prismatic joint directly provide the height of the quadcopter as a feedback. But when implementing the prototype testing system, some type of sensor should be used to measure the height of the quadcopter from the ground level. For our system a sonar sensor mounted under the quadcopter is used.

After achieving reliable readings from the sonar sensor, the altitude hold PID controller was implemented in the secondary controller.

```
error = alt_set_point - alti;
integral = integral + error;
derivative = error - pre_error;
output = (alt_kp*error) + (alt_kd*derivative) + (alt_ki*integral);
pre_error = error;
```

Since the altitude hold controller was tested, only the throttle input from the RF receiver was inserted to the secondary controller. All other roll, pitch and yaw control signal were directly inserted to the primary flight controller from the RF receiver. This allows the user to take back control of the quadcopter, if it didn't perform according to the user's desires.

The second safety measurement is the implementation of the ground control station. This allow real-time PID coefficients tuning, enable different modes and most importantly the kill switch which allows the quadcopter to be shut down immediately if anything goes wrong.

After the implementation of the altitude hold PID controller and tuning Kp, Kd and Ki values to achieve a stable output, an observation was made that, it always had a steady state error of 20 cm. When the altitude set point was set to 50 cm, the system only achieved 30 cm altitude and was keeping it steady. Even after tuning the Ki value, the steady state error couldn't be reduced. An assumption was made that this was due to the increased weight of the quadcopter test system. But after analyzing the secondary controller program intensely, it was discovered that a single program cycle of took 129 ms to execute, which means it was running at 7.75 Hz. There were two main reason for slowness of the secondary controller. First one was the buit-in library function that was used to read the remote controller values from the RF receiver. To read one remote signal, it took 14 ms. At that time, 5 remote signals were read using that function which means overall 70 ms was delay by reading 5 remote signals. As the solution, use of built-in libraries was stopped and a new program was written to read remote signals using Pin Change Interrupt (PCI) function in the microcontroller. After the new program was implemented, a 70 ms delay was removed from the program. The new program took 55 ms per cycle. The reason for this delay was the use of sonar library to read the distance from the sonar sensor. As the solution the use of library was removed and the same PCI method was used to read and calculate the sonar distance manually. After all the changes, the new program only took 4 ms per cycle (250 Hz). The overall performance of the program was improved from 7.75 Hz to 250 Hz which is a 3125% increase in speed.

After the speed improvement, the PID controller started working much faster. The previously used PID coefficients were completely changed for the new faster system. After tuning the PID controller with new parameters, an observation was made of a new problem. The PID controller had to be tuned according to the altitude set point. After intense analysis, the cause of the problem was discovered. At every testing session, the altitude hold PID controller was enabled when the quadcopter was on the ground, which meant the error was maximum at the time.

Because of that PID controller output was very large. This enabled large amount of startup oscillation of the quadcopter. Since the weight of the system was increased, the PID controller has hard time keeping the quadcopter steady.

The solution was to implement a proper take-off function for the quadcopter and enabling the altitude PID controller after that.

7.2.1.3 Takeoff procedure

According to the weight of the quadcopter and the battery voltage, the throttle value need for the quadcopter to hover changed. Because of the increase in weight, the battery drained faster. Which meant the takeoff function should be dynamic and it would work for whatever value is given for altitude set point.

As the solution, a dynamic and reliable takeoff function was implemented. It started with a baseline throttle value. After every second the throttle value increased by 10. This process will loop until the altitude of the quadcopter reached 80% of the set point value. After that the takeoff function exits and the altitude hold PID controller was enabled which is described in Figure 46.

After the implementation of the takeoff function, the whole system became much smoother. Less startup oscillations, which meant the PID controller worked much better at keeping the quadcopter steady.

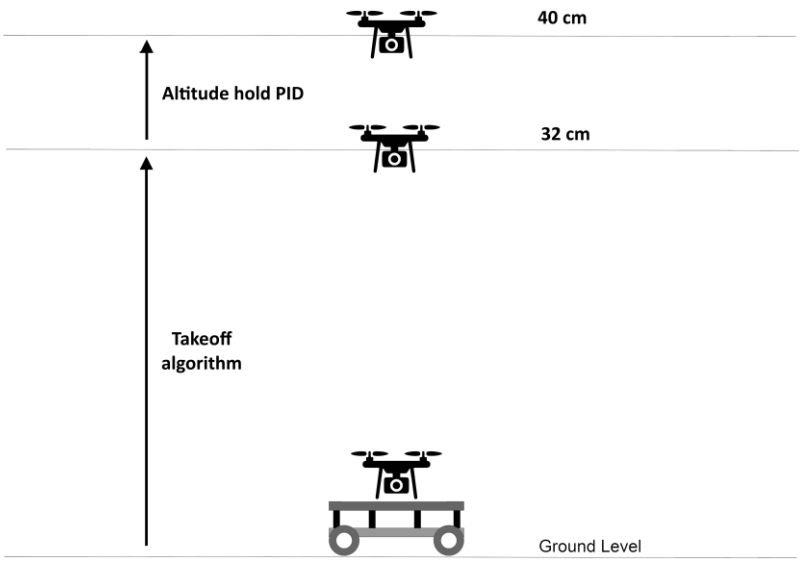


Figure 46. Takeoff procedure

7.2.1.4 Landing procedure

As shown in Figure 47, the Landing procedure worked in to stages. The first stage is descending of the quadcopter. At this stage the altitude set point is reduced by 5 cm at every 2 seconds until it reaches 10 cm. This allows the quadcopter to reduce its altitude smoothly.

After quadcopter reaches 10 cm altitude, the second stage is enabled. At this stage, the throttle value of is reduced by 20 every 200 ms until it reaches 1500. At the end of this stage, quadcopter will be landed on the platform smoothly.

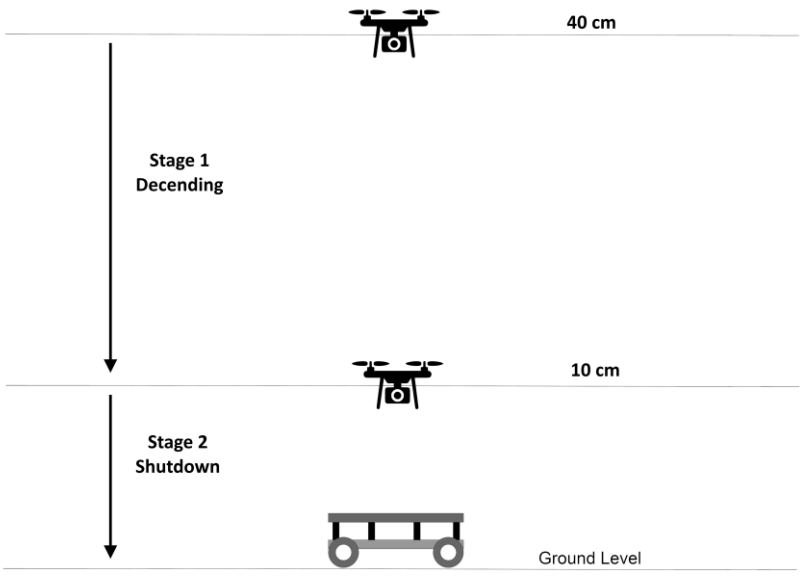


Figure 47. Landing procedure

7.2.1.5 Axis movement controller

Even though in simulation, the prismatic joint directly provide the its position as a feedback, the testing system needs some type of sensing method to determine the quadcopters position relative to the mobile robot.

They method we used is real-time image processing. A wireless camera is fitted under the quadcopter which faces downwards. The mobile platform has a red square on top of it. The wireless camera feed is taken in to a separate computer for faster real-time image processing. Using OpenCV and Python programming languages, the camera feed is processed to find the X, Y coordinates of the red square as shown in Figure 48. These coordinates are then send to the quadcopter through wireless RF link which will be used to control the movement of the quadcopter relative to the red square.

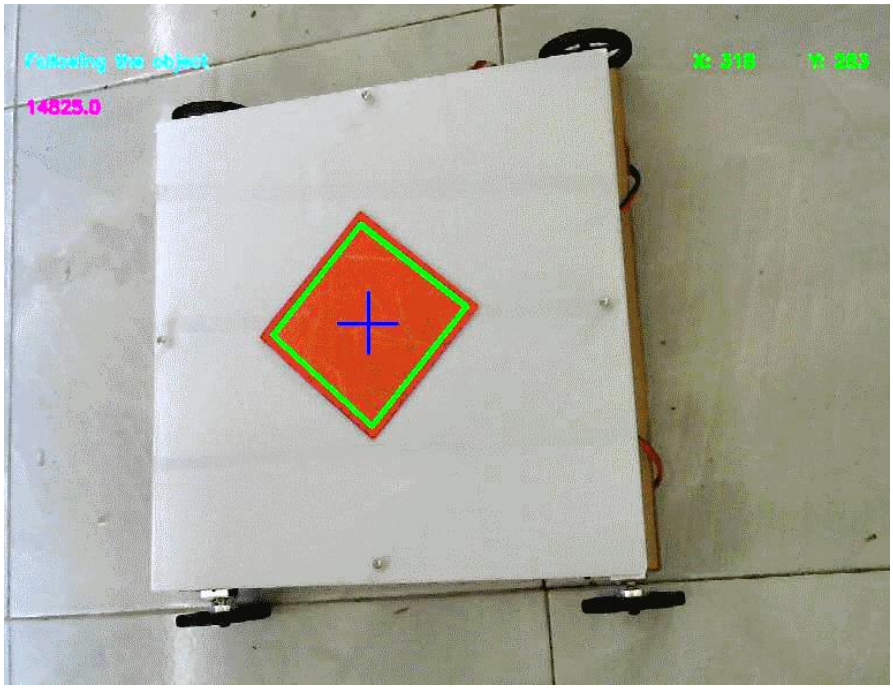


Figure 48. X, Y coordinates of the red square after image processing

The resolution of the camera is 704 x 576. Using this data and the real-time X, Y coordinates, a 2D map is designed to make it easier to implement the axis movement controller.

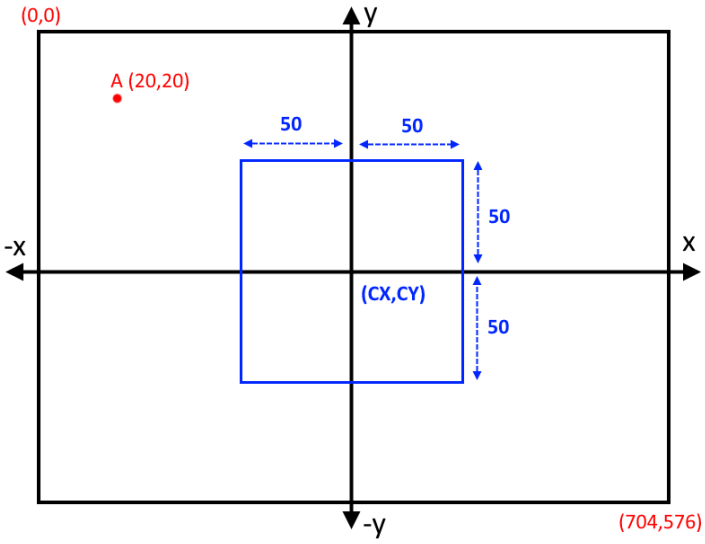


Figure 49. Axis controller coordinate system

The red colored coordinates are related to the camera. Point A is a sample data point sent from the image processing algorithms to the secondary controller. CX, CY are the center points of the virtual coordinate system used inside the secondary controller. Point A coordinate should be converted to new coordinate system before using with the axis movement controller.

$$CX = 704/2 = 352$$

$$CY = 576/2 = 288$$

$$DX = Ax - CX = -332$$

$$DY = CY - Ay = 268$$

Dx and Dy are the new coordinates which will be used for the axis movement controller.

The blue box in the middle of Figure 49, is the idle are for the quadcopter. As long as the quadcopter is inside the box, it will always be above the mobile platform and it is sufficient enough for the quadcopter to land safely on the mobile platform. The axis movement controller doesn't try to controller the quadcopter when it is in the idle zone. When the quadcopter leaves the idle zone, the controller will try move it back into the idle zone.

7.2.2 Image processing Algorithm implementation

The main objective is to takeoff off the quad copter from the mobile robot, follow the landing platform for few meters and landing onto the same mobile platform. After the few seconds of takeoff, quad copter needs to track down the pre-defined marker which is on the landing platform (trailer of the mobile robot) by itself autonomously. In order to achieve this goal we were assigned to implement an image processing algorithm.

Algorithm specifications as follows:

1. Track Specific Object / marker
2. Calculate its Angle
3. Determine the Tracked Object Orientation

The Figure 50 shown below depicts the approach and the Methodology of the processing procedure

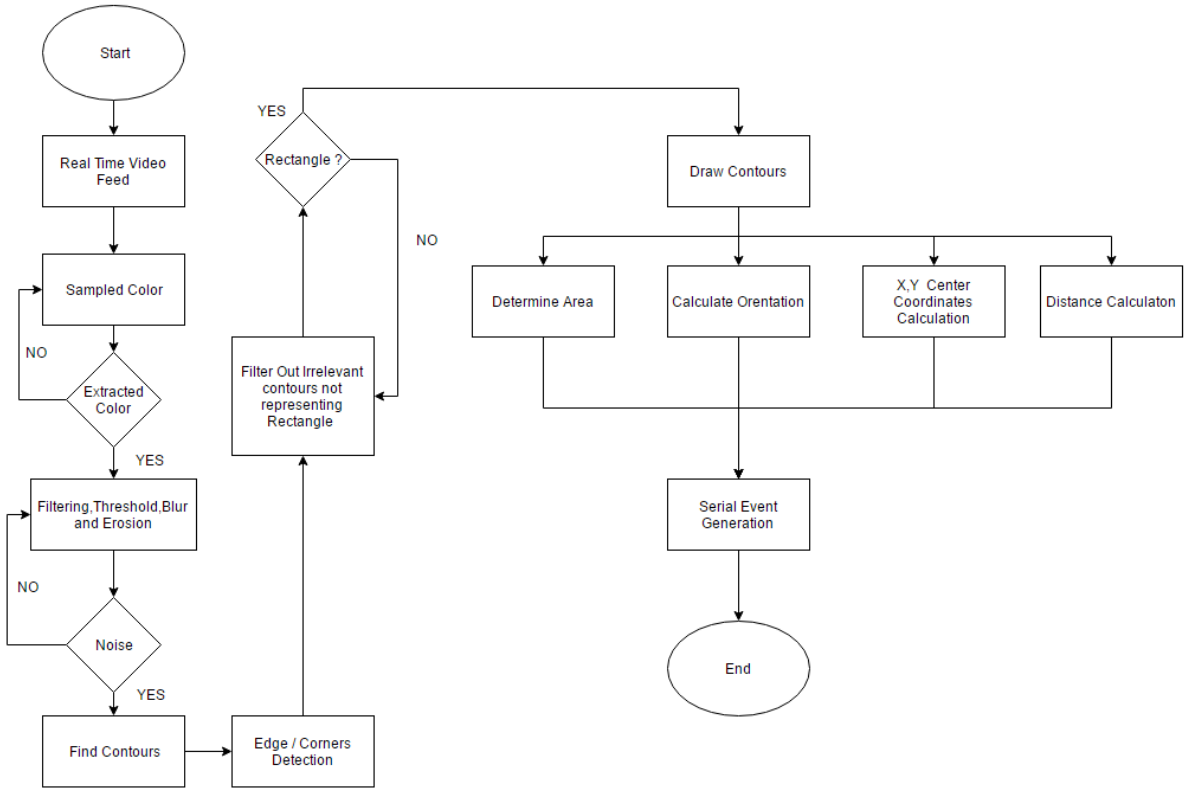


Figure 50: Methodology of Image Processing Algorithm

7.2.2.1 Pre-defined marker tracking and following algorithm

A Red color square shape sticker is glued on the landing platform of the mobile robot to identify as the pre-define marker. Then the FPV camera module mounted under the quad copter body**Error! Reference source not found.**. Since this is a wide-angle camera the distorted images can be expected. Therefore, first we calibrated the camera and calibrated results as follows

fx = 511.16718125 Cx = 108.8047538]

fy = 536.68575864 Cy = 64.70238472]

where fx, fy are camera focal lengths and Cx, Cy are optical centers.

distortion coefficients:

[-5.03769208e-01 1.03407209e+00 4.70793382e-04 -4.45855637e-03 7.63882593e-01]

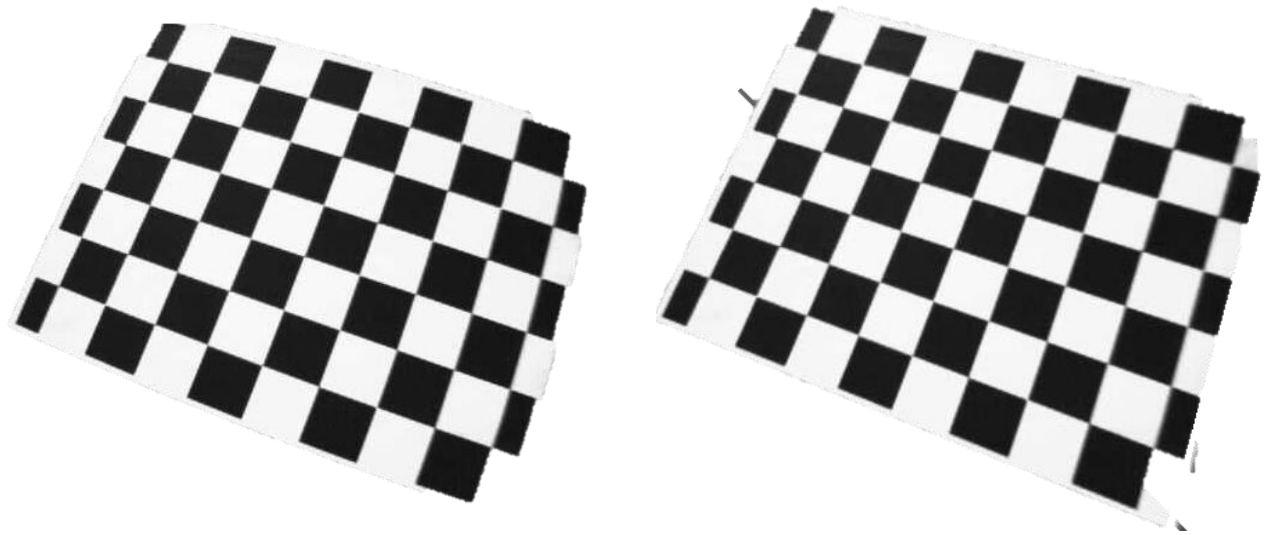


Figure 51: (a) Before Calibration and (b) After calibration

As you can see in the Figure 51(a) bevel shaped (fish-eye) distortion can be seen. This a common problem in wide angle cameras and 51(b) the un-distorted image. A clear change can be seen after calibration. All Curved edges has been removed.

The fine-tuned the HSV range color wanted. In this case the color was red. Then the extracted color filtered using a Gaussian filter to eliminate the noise. Then these frames are thresholded. Thresholded frame then put into morpho graphical transformation for eliminate further noise. Here incoming frames filter out 5 times to observe the exact color we wanted. After eliminating the noise, we found the contours of the extracted object. Then performed a canny edge detection operation to obtain the edges and corner of the object. Now we found the edges. As a geometrical property of square, square shape must have four edges with equal length in it. If the object has 4 edges and equal length of contours surrounded, then it confirmed that the square is detected. After confirming the detected object is a square then algorithm draw contours on real time feed. Then the properties such as area, aspect ratio, solidity and perimeter are calculated. Center coordinates (X, Y) and area then transmit via Serial Link to the quad copter to keep its position steady.

7.2.2.2 Determining YAW Angle

Suppose the mobile platform is travelling through a curved path. When the curve of the path increases the maker of the object also start to relatively move in a curved path. In that case, since our quad copter not going to make any curved moves, the probability of missing the object is high. To overcome this issue, we attached an arrow shaped marker on to the landing

platform to determine the turning angle of the mobile robot. This will make easy to keep track of the mobile robot. In the same algorithm, while keeping track of the red colored square it will also track down the orientation of the landing platform autonomously by controlling its YAW according to the angle. An arrow shape definitely contains 7 edges in it. This property is used to find the contours of the Red colored arrow. The Triangle head of the array kept vertically to observe the reference point of the angle. Which means zero degree. Then by using fit ellipse () function in OpenCV the orientation was determined.

7.2.3 Kalman filter implementation for IMU sensors

The kalman filter is implemented as two sector as shown in Figure 52. The 2D kalman filter is implemented for finding roll, pitch and yaw using MPU6050 and HMC5883L IMU sensors for quadcopter. The 1D kalman filter is used to measure the acceleration of the mobile robot using ADXL335.

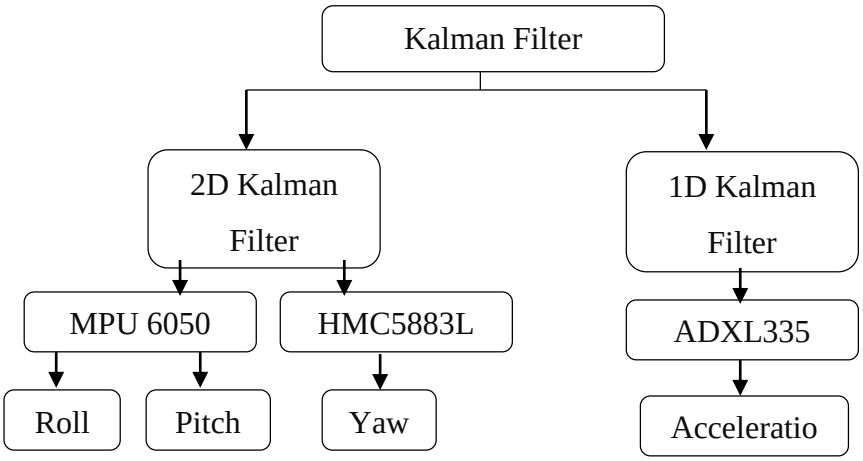


Figure 52: The flow chart of 1D and 2D kalman filter used in IMU

8.0 Results

8.1 Simulation testing

8.1.1 Mathematical Modelling Simulation

The determined formulas of motion for a quadcopter, beginning with the voltage-torque relation for the brushless motors and working through the quadcopter kinematics and dynamics. The simulation formulas ignored aerodynamically impacts such as non-zero free stream velocity and blade-flapping, yet included air friction as a linear drag force in all directions. The utilized of the equations of motion to make a simulator in which to test and visualize quadcopter control mechanisms. The quadcopter mathematical modelling is tested with the PID controller.

The values for all of physical constants, a function to process the rotation matrix R, and functions to change over from an angular velocity vector w to the derivatives of roll, pitch, and yaw and vice-versa.

The Figure 53 shows that the basic input parameters of the quadcopter. The input parameters values is given in the Table 2, these parameters is used to simulate the quadcopter.

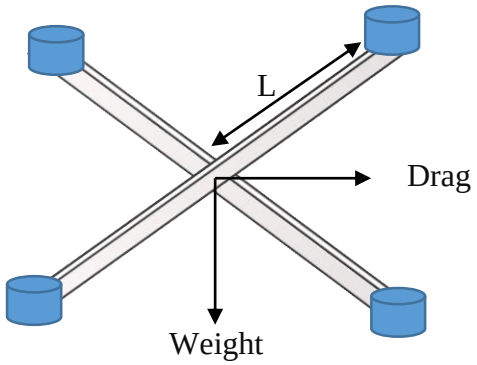


Figure 53 : Input parameters of Quadcopter model

Table 2: Input parameters of the Quadcopter simulation

Parameter	Values
g = Gravity (ms^{-2})	9.81
Mass (Kg)	0.950
L = Length of mid of the frame	0.165
k = dimensioned constant	$3x10^{-6}$
b = drag coefficient	$1x10^{-7}$

The quadcopter in a 3D visualization which is updated as the simulation is running as shown in Figure 54, when PID values shown in Table 3.

Table 3: PID parameters

Controller parameter	Values
P	8.2
I	10
D	8.0

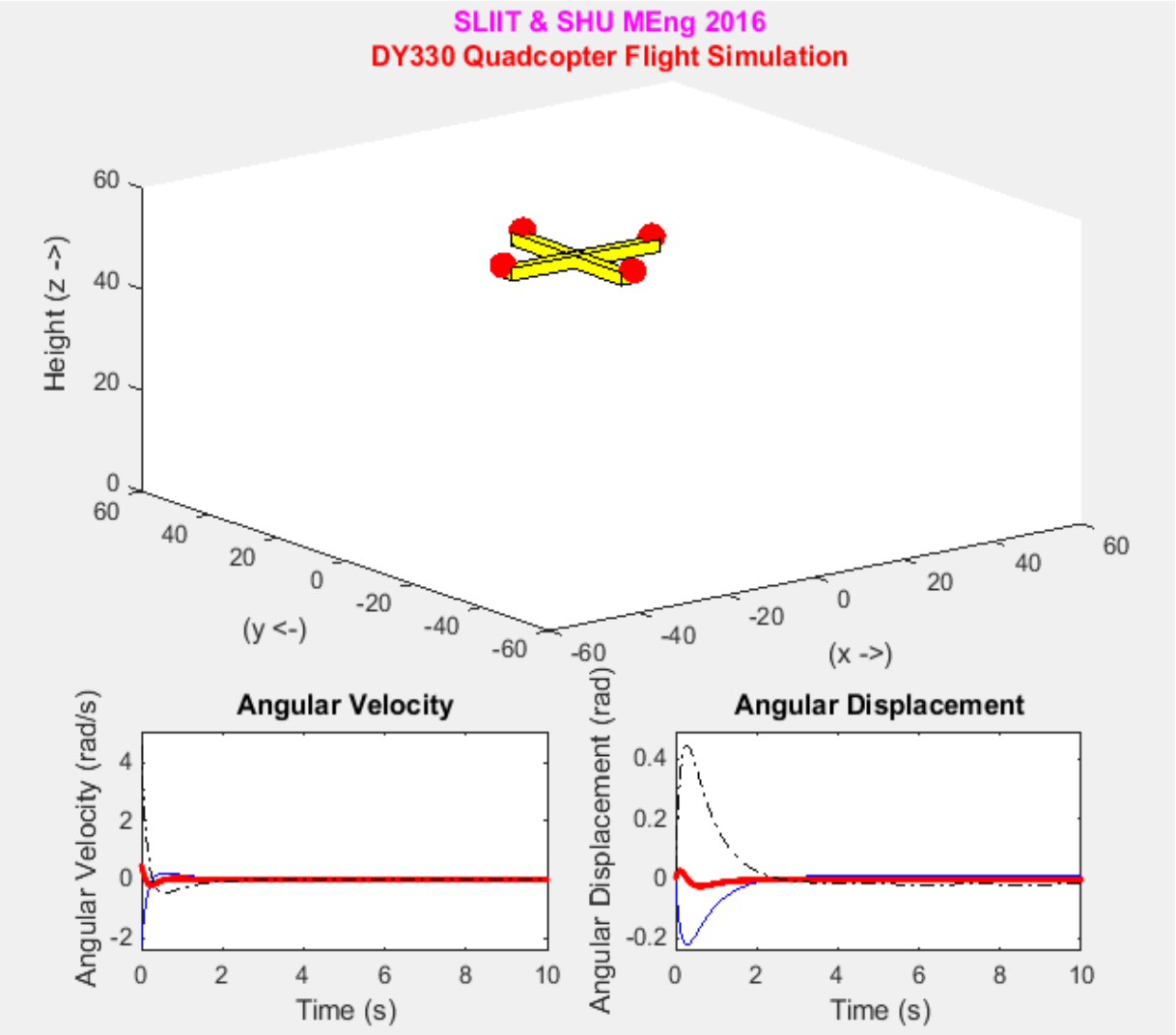


Figure 54: Mathematical Quadcopter simulation

8.1.2 Simulink Model Simulation

8.1.2.1 Altitude Hold PID controller

The first graph of the Figure 55, shows the RPM of a single motor, the second graph shows the altitude response of the PID controller and the third graph shows the total thrust used when the altitude hold PID controller is running. The settling time of the controller is 15 seconds. Table 4 shows the PID coefficients used to obtain these results.

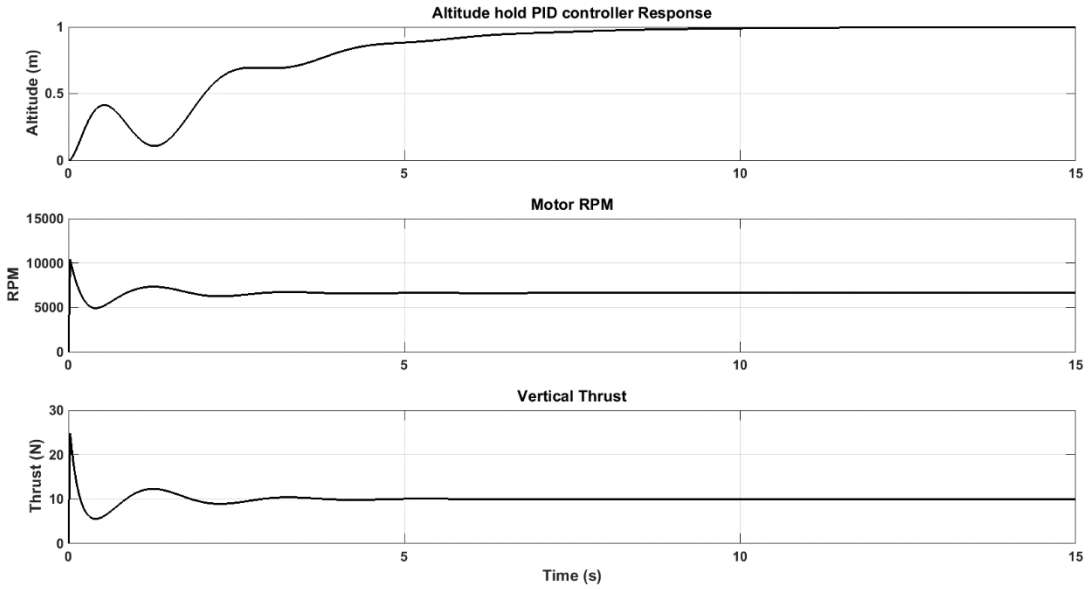


Figure 55. Simulation altitude hold PID controller in action

Table 4. Simulation altitude hold PID controller coefficients

K_p	9.00
K_d	2.00
K_i	4.00

8.1.2.2 Roll PD controller

The roll PD controller has a very stable response as seen from Figure 56. In here, within the first 3 seconds the set point is 0 degrees. From 3 seconds to 6 seconds the set point is change to -15 degrees. From 6 seconds to 9 seconds the set point is again changed to 0 degrees. The first graph of Figure 56, shows the angle response of the PD controller. The second and third graphs show the RPM of left and right motors and finally in the fourth graph the output of the PD controller is shown. Table 5 shows the PD coefficients used to obtain these results.

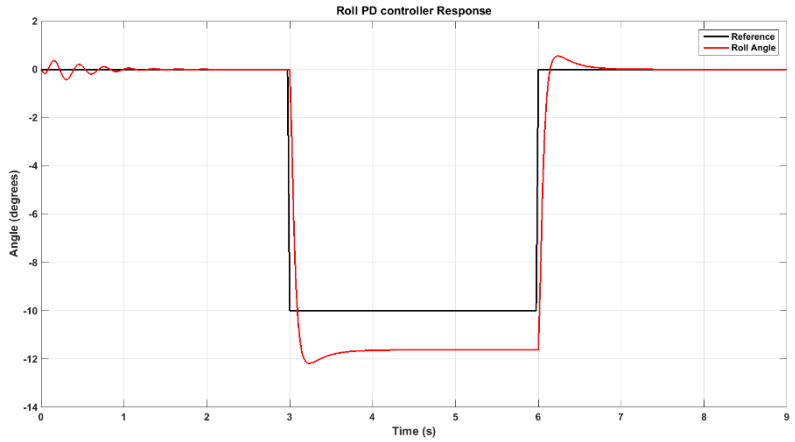


Figure 56. Roll PD controller response

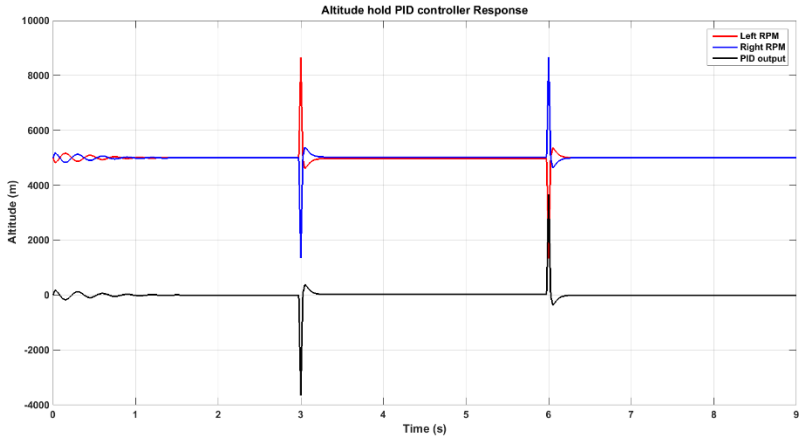


Table 5. Roll PD controller coefficients

Kp	40000
Kd	10000

8.1.2.3 Axis movement controller

As seen from Figure 57, the axis movement controller takes 7 seconds to settle down. The input to the controller is limited to ± 15 degrees to prevent the quadcopter from going out of control. The first graph of Figure 57 shows the X position of the quadcopter and the second graph shows the roll angle of the quadcopter during the movement. The response of the PD controller is kept slow to prevent the quadcopter from going out of control. Table 6 shows the PD coefficients used to obtain these results.

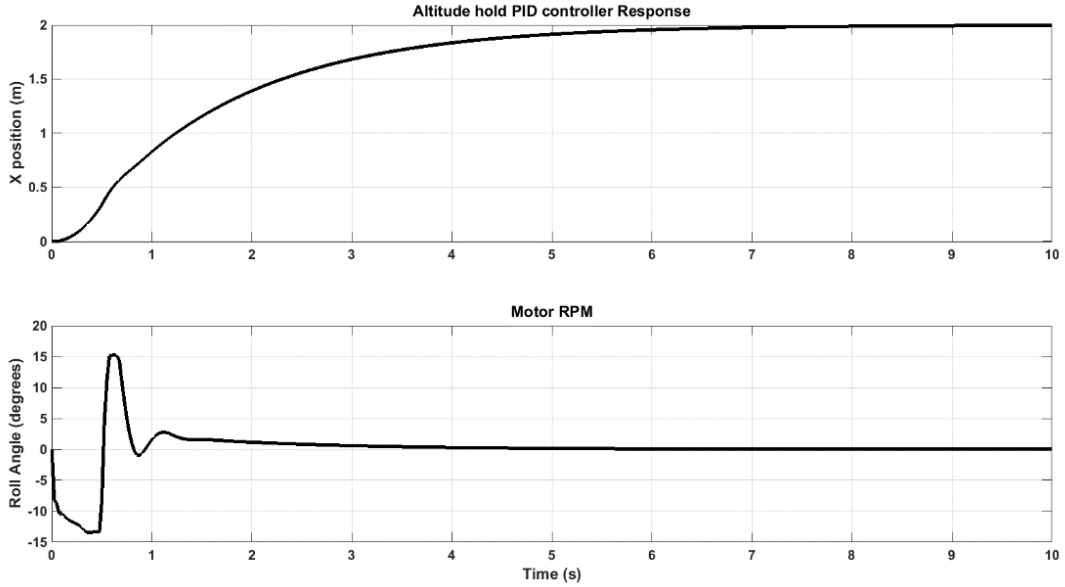


Figure 57. Axis movement PID controller in action

Table 6. Axis movement PID controller coefficients

Kp	40
Kd	60

8.1.2.4 Full system simulation

The Figure 58 shows the full system simulation results. The full system simulation runs as follows. At the start, the quadcopter takes off. When the quadcopter reaches the given altitude of 1 meters, then the mobile robot starts to move at a constant speed of 10 cms^{-1}. Then the quadcopter follow the mobile robot. After 10 seconds the mobile robot does stop. After waiting for 5 seconds, the mobile robot starts moving again at a constant speed of 20 cms^{-1}. After 5 seconds of moving, the mobile robot will stop. Then the quadcopter start the landing procedure. First it descends down to 20 cm height and then the quadcopter goes down as a ramp and shut down. Through all the simulation, the quadcopter follows the mobile robot.

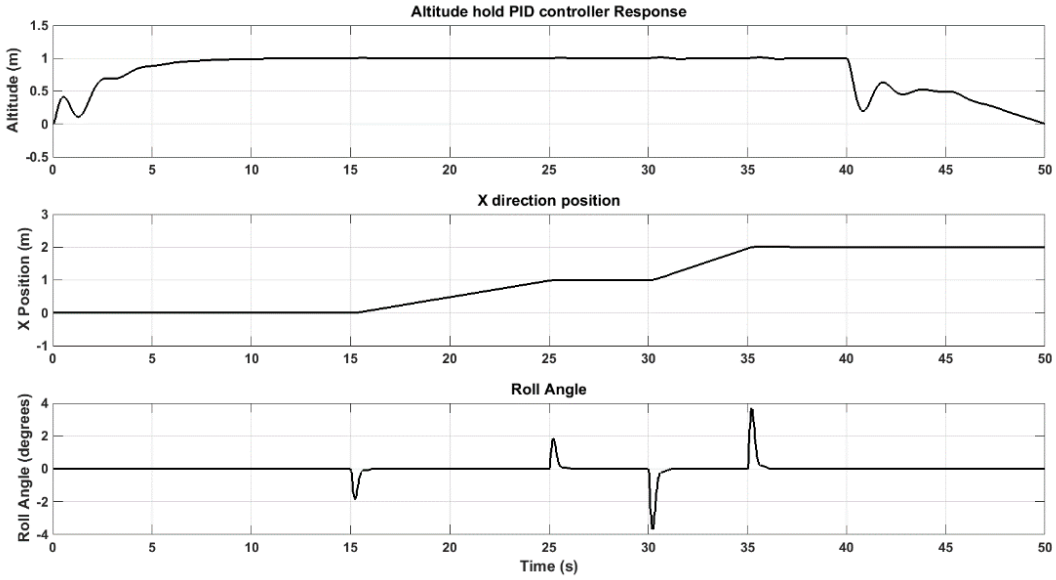


Figure 58. Full system simulation results

The first plot in Figure 58 shows the altitude of the quadcopter. The second plot shows the x direction movement and the third plot shows the roll angle of the quadcopter.

8.2 Prototype Testing System

8.2.1 Altitude Hold PID controller

As seen from Figure 59, the test altitude hold PID controller works well. But there is a lot of oscillation, which means the PID controller can be fine-tuned to achieve much better performance results.

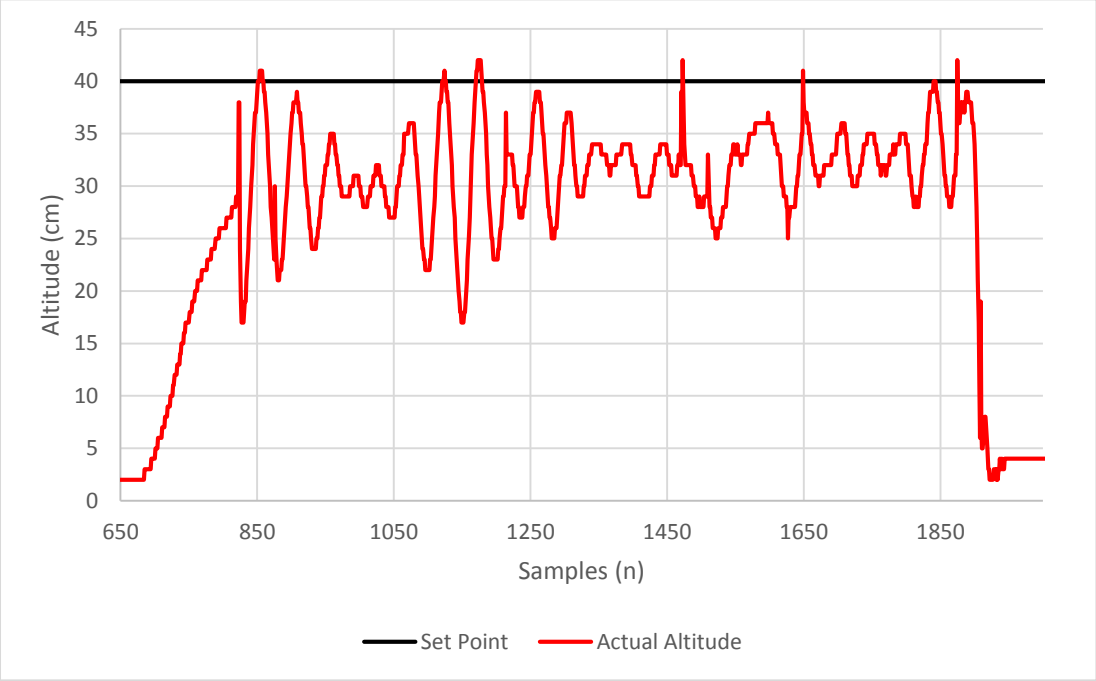


Figure 59. Test altitude Hold PID controller in action

Table 7.Test altitude hold PID controller coefficients

Kp	1.1
Kd	20
Ki	0.000003

The biggest reason for the oscillation is the increase of weight in the test system. Because of that, the altitude hold PID controller has a hard time keeping the quadcopter steady. The takeoff procedure works smoothly, producing a stable response. Table 7 shows the PID coefficients used to obtain these results.

8.2.2 Pre-Defined Marker tacking results

Figure 60 shows the results of pre-defined marker tracking algorithm.

(a) Logitch c270 HD webcam (b) FPV camera

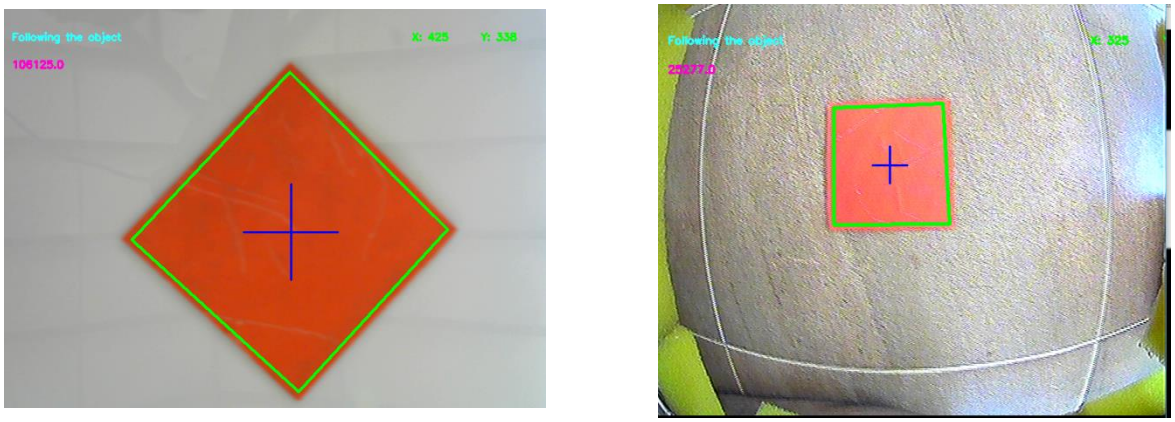


Figure 60: Image Processing results from normal camera and a wide-angle camera

8.2.3 YAW Angle Determination Results

Figure 61 Shows the Angle Determining Results from the Algorithm.

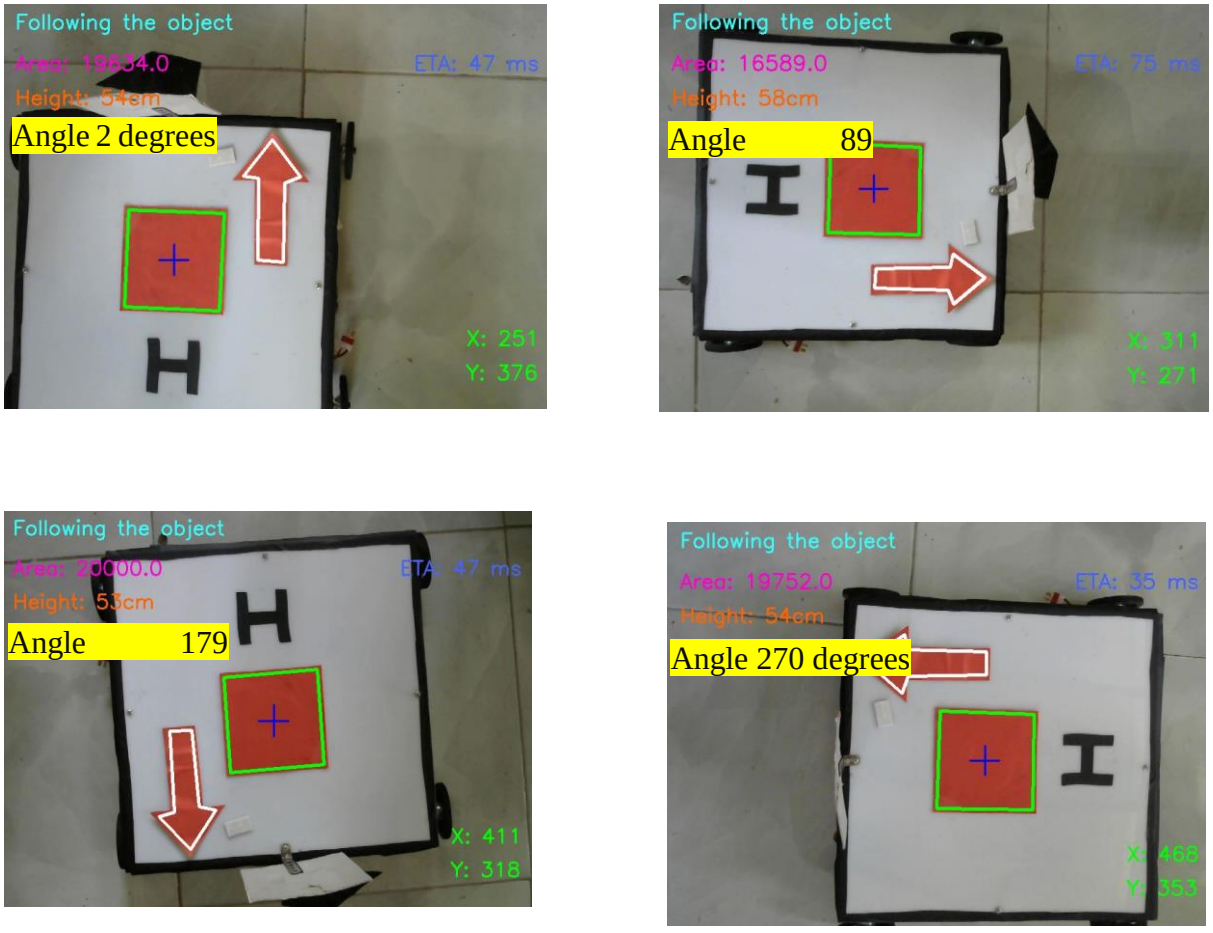


Figure 61: Angle Determining Results from the Algorithm

8.2.4 Distance determination Results Using Triangle Simillarity

Figure 63 Shows the Distance Calculation Results using Triangle Similarity.

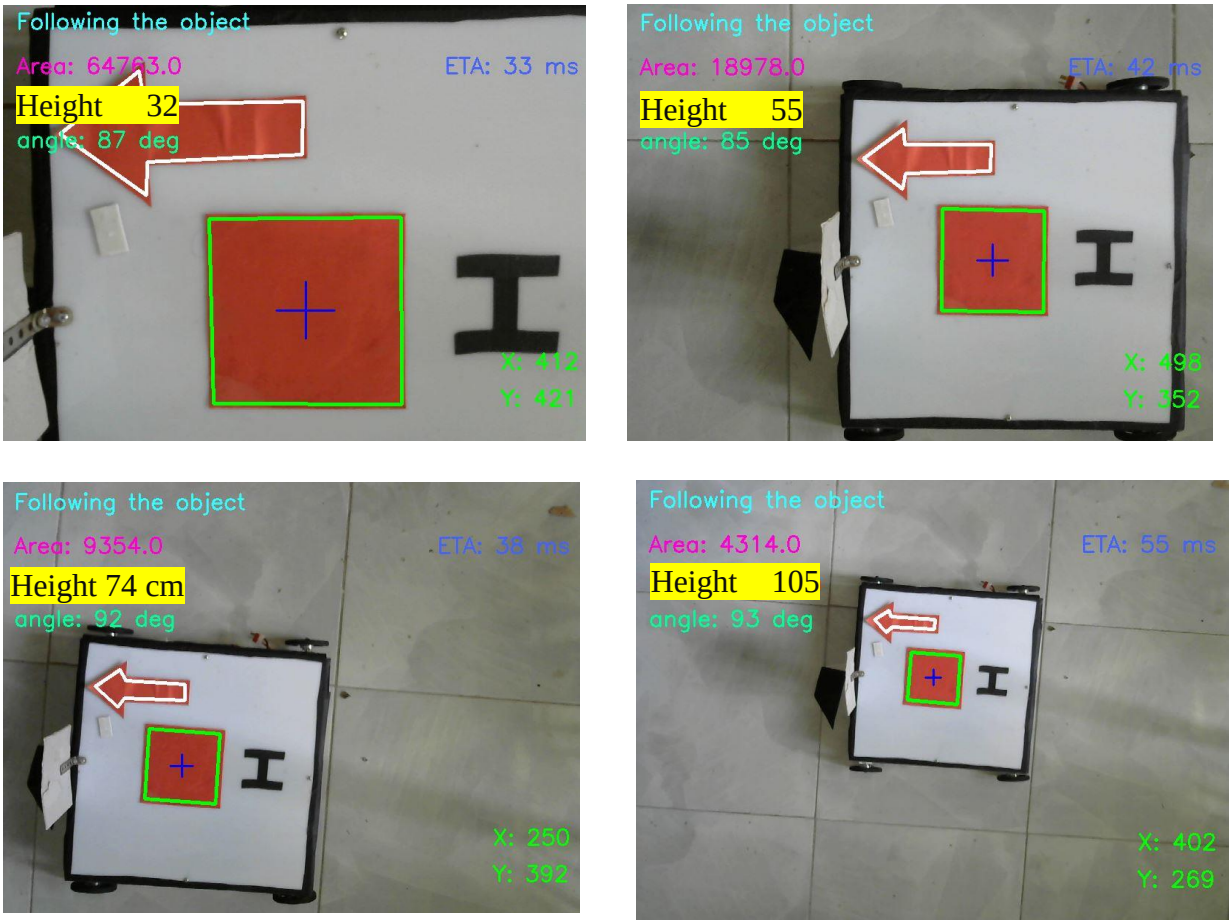


Figure 63: Shows the Distance Calculation Results

Error! Reference source not found. Analytical Distance to the Object.

Table 8: Analytical distance

Perceived Focal Length	Perceived Width in pixels	Actual Width in centimeters	Area in Square pixels	Distance to the object in cm
675.498	225.1666	10	50700	30.00
675.498	134.1641	10	18000	50.34
675.498	88.31761	10	7800	76.49
675.498	64.03124	10	4100	105.495

Figure 64shows the comparison between distance vs perceived width of the object.

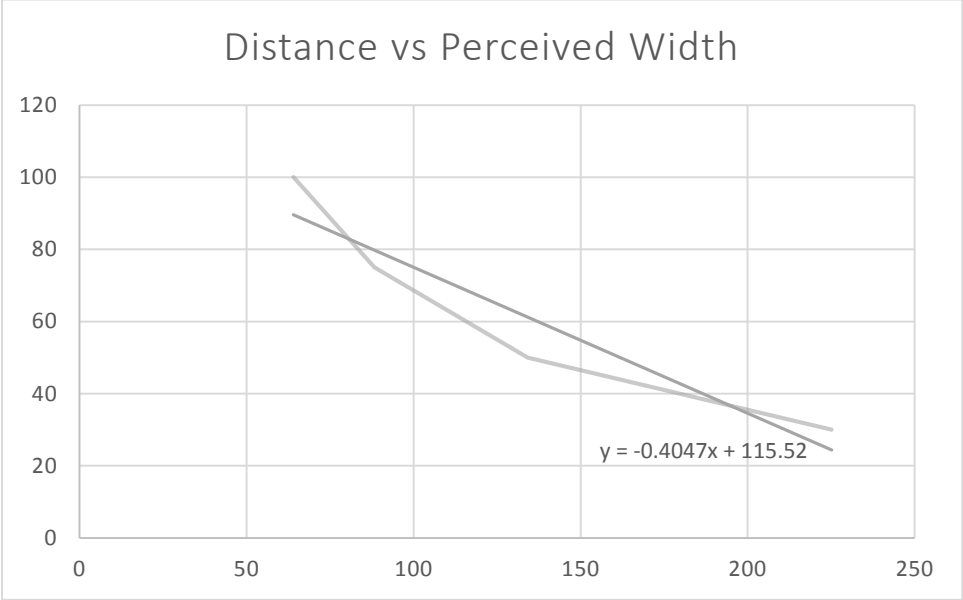


Figure 64: Distance vs Perceived width

Figure 65 shows the comparison between height vs area vs perceived width.

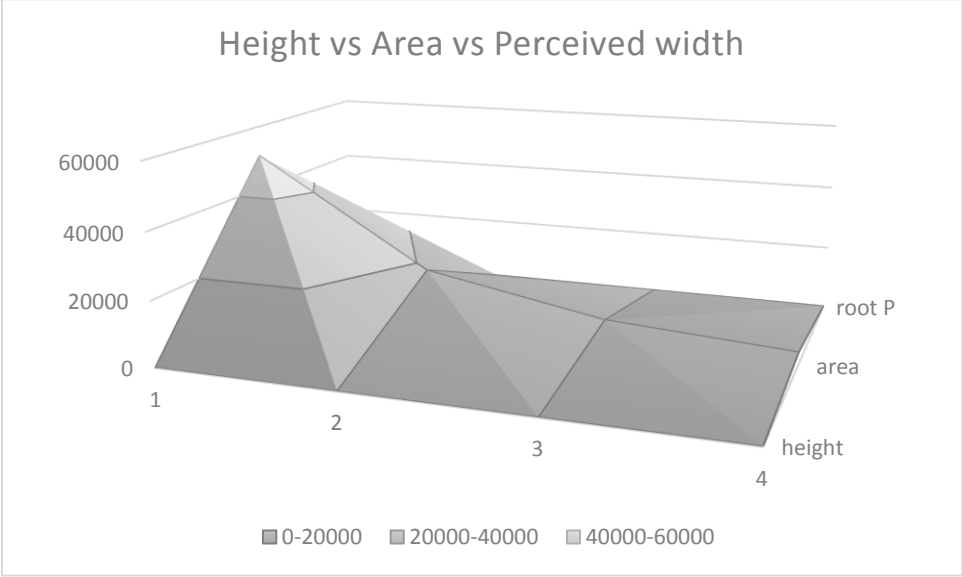


Figure 65: Height vs Area vs width

8.2.5 Kalman filtered IMU outputs

The kalman filter is implemented for removing vibration noise from the IMU sensors. The matlab serial monitor is implemented for checking the kalman filter output before applying the quadcopter in the real world. The cube size box made which contains Arduino mega, Bluetooth dongle, HMC588L and MPU6050 for model to check the output as shown in Figure 66.

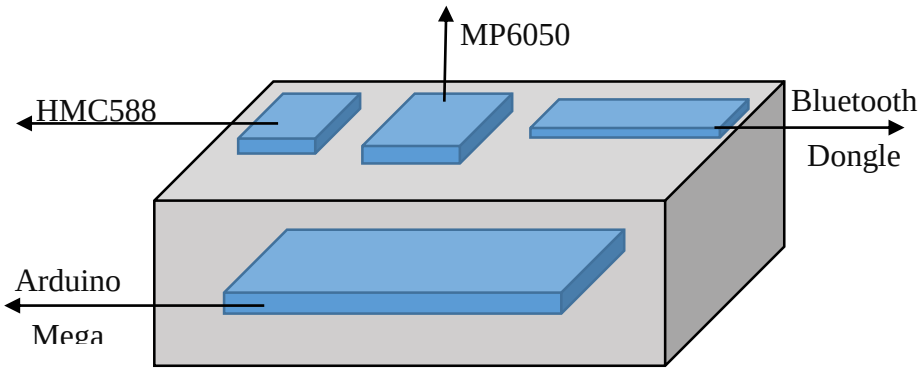


Figure 66: The cube model

The Figure 67 shows the output with and without kalman filter when the cube model move. This picture clearly shows that when sudden vibration came it will remove the noise and smooth that vibration output.

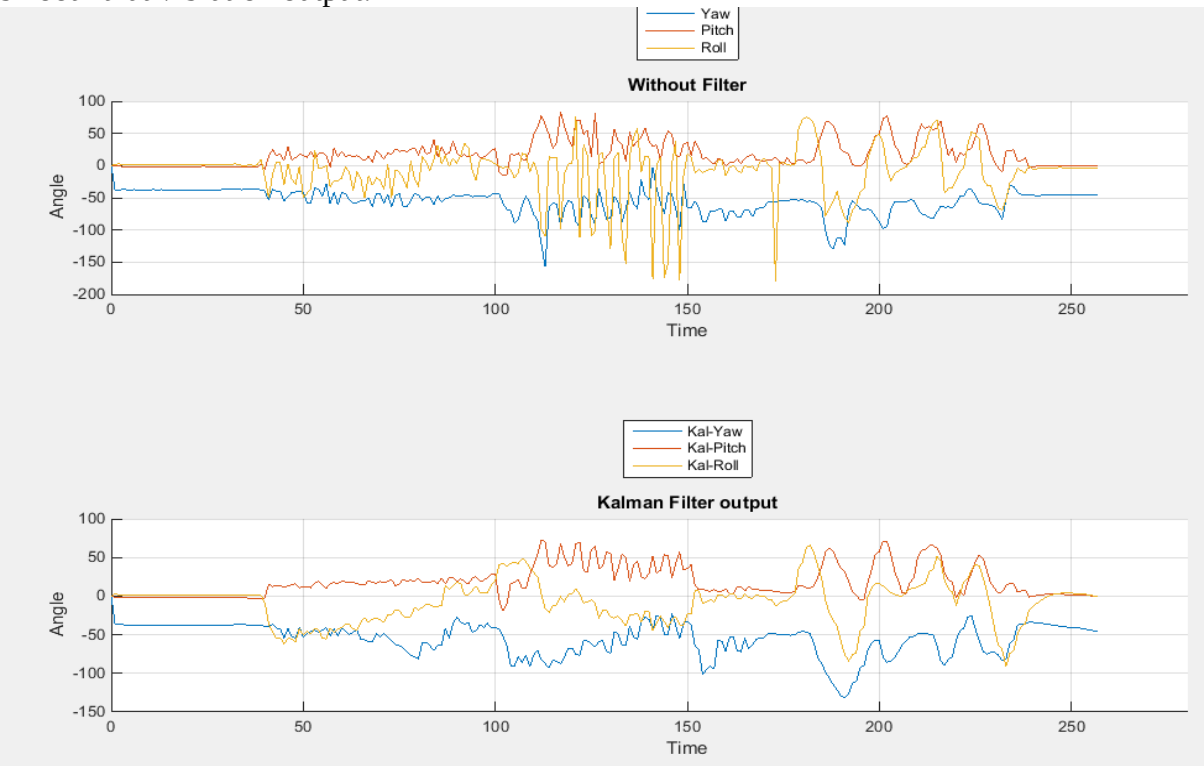


Figure 67: The yaw, pitch and roll with and without kalman filter for cube model

After checking the kalman filter using matlab serial monitor, it applied in the quadcopter. The Figure 68 shows the pitch output of the MPU 6050 with and without kalman filter.

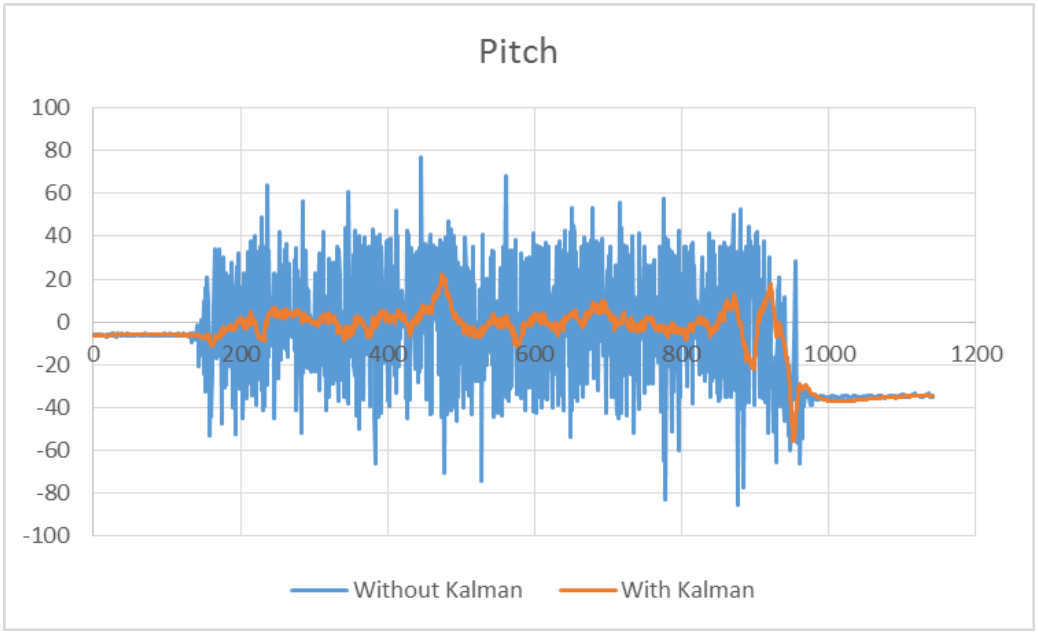


Figure 68: The Pitch output of while quadcopter move to X axis

The Figure 69 shows that the roll output with and without kalman filter when quadcopter rotating at Y axis.

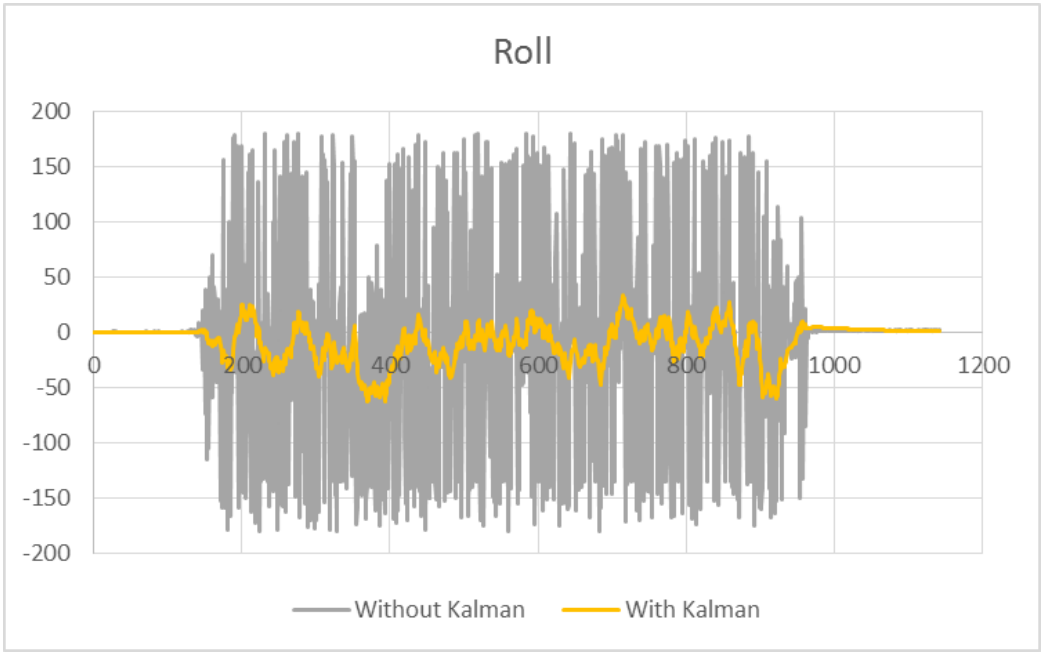


Figure 69: The roll output of while quadcopter move to Y axis

The Figure 70 shows that the yaw output with and without kalman filter when quadcopter rotating at Z axis.

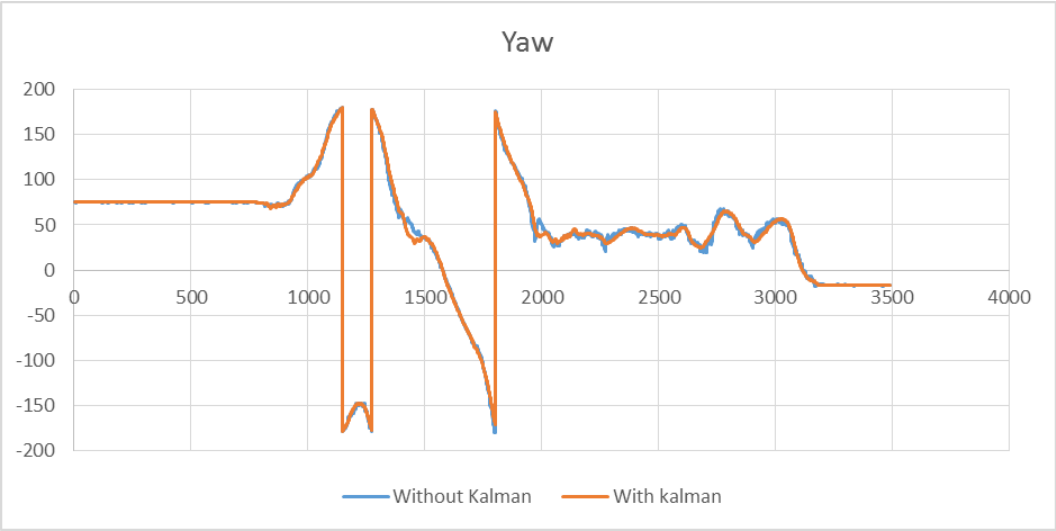


Figure 70: The Yaw output of while quadcopter move to Z axis

The accelerometer returns the value when the acceleration is occur but it have much the vibrations so finally it is returning values of the angle is wrong. The pitch and roll measured using of accelerometer that is the reason both values have more noise in that data without kalman filter. So with the help of kalman filter the vibration noise can easily remove. The magnetometer is used to measure the yaw of quadcopter but the output of magnetometer does not have that much noise as accelerometer.

The 1D kalman filter is used to measure the velocity of the mobile robot. The Figure 71 shows that kalman filter acceleration and velocity of the mobile robot.

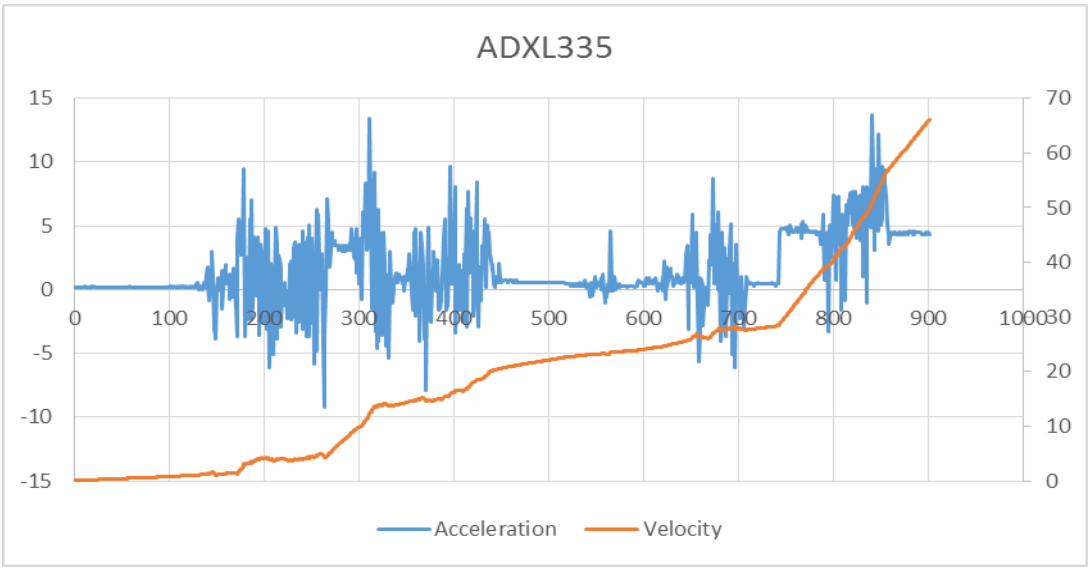


Figure 71: The acceleration and velocity of mobile robot

8.3 Discussion of results

8.3.1 Simulation vs. real world

As seen from Table 4 and Table 7 the PID controller coefficients between the simulation and the experimental system are very different. There are few reasons for this difference.

The first one is, since the simulation environment is an ideal one and it will not replicate a real world system. All the prismatic and revolute joints used in the simulation doesn't have any friction. But when we consider fly a quadcopter in real world, the air in the environment will cause air resistance on the quadcopter which will affect the movement of the quadcopter in 3D space.

The second reason is the battery voltage. In the simulation there is no reduce power source applied with the quadcopter. But in the experiment, the battery voltage has a huge impact on the thrust produced. When the battery voltage goes down, for the same throttle value inserted to the flight controller, the actual thrust produced from the propellers go down. Because of that the PID controller have to compensate for the battery voltage.

The third reason is the deviation of the sonar senor readings. Since the X, Y axis movement controllers are not implemented yet, when the altitude hold controller is testing, the quadcopter tends to move through X and Y directions. These movements are manually controlled by the remote controller. When the quadcopter moves in any X or Y directions its angle change accordingly. When the quadcopter is angled, the sonar values increase by small amount. Same time, since the quadcopter is angled, the total thrust in Z direction is reduced and the quadcopter starts to go down. But because of the flawed reading from the sonar sensor, the altitude hold PID controller is slow to response and a lot of oscillation happens because of that.

The fourth reason is the difference of PID controllers used in simulation and experiment. In the simulation the PID controller works in continuous time domain, but in the experiment the PID controller works in discrete time domain.

The fifth reason is the difference in simulation and experimental control signals. In the simulated system, final value inserted to each motor is the angular velocity for that motor and a P controller keeps the motor at the given angular velocity constantly. But in the experimental system, the speed of the brushless motors are controlled using a ESC which only takes speed inputs as 50 Hz signals with 1000 us to 2000 us duty cycle values. Which means that the output

of the altitude hold PID controller in the experimental system should always produce a value between 1000 and 2000 which then can be transferred to the ESCs.

To achieve similar results between simulated and experimented systems, the simulation should also have the same control properties similar to the experimental system.

8.3.2 Image Processing Algorithm

A fully completed computer vision based algorithm was developed for autonomous quad copter which gives the abilities of following and landing on to a mobile platform autonomously. Algorithm was tested in both intel 2.9 GHz 64-bit core i5 CPU (PC) and a 1.2 GHz 64-bit quadcore ARMv8 CPU (Raspberry pi 3). In algorithm, both serial and the processing part took 40 milliseconds per frame on PC and 180 milliseconds on raspberry pi 3. With compare to PC, the raspberry pi 3 takes a lot of time for handling the processing part. From the testing results we observe that, for automation it is necessary to reduce time per execution cycle of the code. The minimum we could achieve was 40 milliseconds. That made a great change in handling quad copter autonomously.

We noticed that sometimes the crosshairs and bounding box regions of the detected target tend to "flicker". This is because many computer vision and image processing functions are very sensitive to noise, especially noise introduced due to the motion and the vibration of the quad copter a blurred square can easily start to look like an arbitrary polygon, and thus our target detection could fail. Another thing is that the HSV value ranges are varying due to different weather conditions. For same color, we had to change HSV values frequently. Use of track bars made our work ease. Since we used a FPV wide angle camera, sometimes it fails to detect the square shape properties of the marker due to its wide-angle view. the marker looks like a rhombus or rectangle when quadcopter moveout from the target. Even though the color is visible to the computer vision, target acquiring might fails. To overcome this problem, we tried out blob detection method. However, blob detection python script has been able to successfully detects the target.

In the distance measuring part, when we captured the photos, our measuring tool had a bit of slack in it and thus the results are offset by roughly 1 mm to few centimeters. we also captured photos rashly and not 100 % on top of the marker which added another error to our calculations. This could cause varying results between actual and the measured heights. A comparison between Measured and the actual results are tabulated in **Error! Reference source not found.**.

Table 9: Comparison Actual vs Measured heights

Actual Measurement (cm)	Experimental Results (cm)
30	30.00
50	50.34
75	76.49
100	105.495

8.3.3 The kalman filter outputs

The 2D kalman used in MPU6050 takes 3.7ms to process the value so it works around 270Hz frequency. He 1D kalman filter takes 1.2ms so this at 833 KHz frequency. The ADXL335 is not that much accurate sensor that is the reason the Figure 35 output shows more noise and using 2D kalman filter in MPU6050 it merge the accelerometer and gyroscope value so it will reduce the vibration noise more than only use the accelerometer output in ADXL335. For 1D kalman filter process noise covariance (q) and measurement noise covariance (r) has to be find more accurate value to get the proper kalman output. That is somehow difficult to choose those value and the parameter (q and r) defined to 1D kalman filter randomly checking numbers which gives more precision output.

9.0 Conclusion and Future work

9.1 Conclusion

The takeoff, altitude hold hovering and the landing of the quadcopter is fully complete. But the altitude hold PID controller can be fine-tuned to achieve much better results. But we were unable to completely finish the axis movement controller. Implementing smooth takeoff and landing function for the quadcopter and the implementation of the altitude hold PID controller are key results that have been achieved.

The mathematical model of quadcopter executes to predict how real hardware implement quadcopter will function. Mathematical model is, the initial phase in comprehending the mathematical standards and physical laws which are connected to the quadcopter system. The goal is to characterize the mathematical model which will describe the quadcopter behavior with satisfactory accuracy and which can be, with specific changes, appropriate for the comparative setups of real hardware quadcopter. Toward the start of mathematical model derivation, coordinate systems are defined and clarified. By utilizing those coordinate systems, relations between variables defined in the earth coordinate system and in the body coordinate system are defined. Facilitate, the quadcopter kinematic is portrayed which enables setting those relations. Additionally, quadcopter dynamics is utilized to introduce forces and torques to the model through usage of Newton-Euler method.

Simulation demonstrate show the roll, pitch and yaw angles as a component of time. Also, the system try to control only angular velocities, so positions and linear velocities do not approximate to zero. Notwithstanding, the z position will remain consistent, because the system constrained the total vertical thrust to be to such an extent that it keeps the quadcopter perfectly aloft, without climbing or sliding. Be that as it may, this is truly just a curiosity. With the restricted detecting that there is nothing can do to control the linear position and velocity of the quadcopter. While in theory could determine the linear velocities and positions from the angular velocities, practically speaking the values will be so noisy as to be totally pointless. In this manner, the system restrict to just stabilizing the quadcopter angle and angular velocity.

Since the start of the implementation of the testing system, we faced a lot of problems. Some of them were solved. But some can’t be solved without doing major changes to the system. One such issue is the weight of the quadcopter. Since the addition of the custom made stand and the sensor the whole weigh has increased. Which means the system using more power which leads to less flight time. The altitude hold controller has a hard time keeping the

quadcopter steady when the battery is draining. Losing weight is not possible since all the sensors and controller are necessary. The only solution is to move to a bigger quadcopter platform such as DJI F450. With the motor and propellers recommended for this frame, the additional weight of the sensor won't be a problem for the system.

When implementation of the Image processing algorithm, also had face a lot of problems. Some of them were figure out and remove those problems. The raspberry pi 3 on the mobile robot did not give us the expected outcome of the image processing algorithm. We found out that it's processing power is not powerful enough to handle both serial event and the processing part at the same time. Therefore, we carried out testing on a PC. As we used a wireless FPV camera, the compatibility with the raspberry was raised another major problem. At the moment, we're trying to reduce the weight of the quad copter, so mounting a compatible wireless device would give additional weight which exceeds the weight lifting limit of the quad copter. The main challenge within the visual tracking process is that the environmental conditions because the system perform its tasks in dynamic environments. Changes in environmental conditions altered the HSV values significantly. However, with the Implementation of online HSV tuner the problem seems to be solved. When we were capturing the live video feed from the FPV camera, we identified that some frames were lost during hovering period. Since the FPV cameras are wide angle and wireless capturing devices we assumed that the reason for the loss might be signal drop or the resolution. but the actual issue was the fish eye effect of the camera module. Due to the fish eye effect the edges and the length of the square shape object got distorted. When the camera move to a corner of the defined resolution the probability of distortion seems to be very high. Therefore, camera calibration was done to eliminate and reconstruct the images that we want to have for the processing.

Using sonar sensor as an altitude measuring sensor is only accurate up to 2 meters. Using a sensor like LiDAR will be much more accurate, reliable.

Implement a gimbal platform where all the sensors can be attached will be a very important step. The gimbal system will be attached underneath the quadcopter and all the sensors used such as camera, LiDAR and sonar will be attached to the gimbal system. It will keep all the sensor horizontal which means all the sensor readings will be more accurate even when the quadcopter is moving.

Even though KK 2.1.5 flight controller is easy to use, it's not a reliable flight controller. These off the shelf flight controllers are designed to be controlled using Radio controller. Because of

that, there is a dead band area in all roll, pitch and yaw controllers of the flight controller. Which means they will act as non-linear system when we implement the axis movement controller. To solve this problem, we have to design our own flight controller which will take much more time and effort.

9.2 Future work

Implementing the axis movement controller using image processing and controller such as PID will improve the accuracy of the system significantly. As the next step to implement the autonomous surveillance system, both mobile robot platform and the quadcopter should be equipped with GPS and an algorithm should be implemented for the quadcopter to follow the mobile robot using GPS at high altitudes. The FPGA-SOC based image processing algorithm is develop for future. This increase the speed of the overall system process. Also 2D kalman filter library also try to write by own to increase the speed of the process and update the 1D kalman filter for more accurate way.

10.0 References

[1]. Quad-Rotor UAV 2016 [ONLINE] Available at: http://depts.washington.edu/soslab/mw/images/2/2b/Group3_finalreport.pdf. [Accessed 17 April 2016].

[2]. Wireless control quadcopter with stereo camera and self-balancing system 2016. [ONLINE] Available at: http://eprints.uthm.edu.my/2926/1/mongkhun_qetkeaw_1.pdf. [Accessed 17 April 2016].

[3]. Autonomous Quadcopter Docking System 2016 [ONLINE] Available at: https://people .ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2012to2013/ssm92/ssm92_report_201305171020.pdf. [Accessed 17 April 2016].

[4]. Design and Implementation of a Quadcopter with Visual Control 2016 [ONLINE] Available at: http://www.es.ele.tue.nl/education/5HC99/wiki/images/0/0f/ Report_ group3 .pdf. [Accessed 17 April 2016].

[5]. Quadrotor prototype 2016. [ONLINE] Available at: https://fenix.tecnico.ulisboa.pt/ download File/395139421058/Tese_de_Mestrado.pdf. [Accessed 17 April 2016].

[6]. Modelling and Control of an Unmanned Aerial Vehicle2016. [ONLINE] Available at: https: //espace.cdu.edu.au/eserv/cdu:41657/Thesis_CDU_41657_Hanna_W.pdf. [Accessed 17 April 2016].

[7]. Study on the 3-DOF attitude control of free-flying vehicle 2001. [ONLINE] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=931661&url=http %3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F7417%2F20158%2F00931661. [Accessed 17 April 2016].

[8]. Dynamics Identification & Validation, and Position Control for a Quadrotor 2016. . [ONLINE] Available at: http://students.asl.ethz.ch/upl_pdf/229-report.pdf. [Accessed 17 April 2016].

[9]. Quadcopter Fuzzy Flight Controller 2016. . [ONLINE] Available at: http://oru.diva-portal.org/smash/get/diva2:567121/FULLTEXT01.pdf. [Accessed 17 April 2016].

[10]. Back stepping based PID Control Strategy for an under actuated Aerial Robot [ONLINE]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.475.9263&rep =rep1&type=pdf. [Accessed 17 April 2016].

[11]. J. Jeffrey et al, “UAV Quadcopter Landing on a Displaced Target”, 2014.

[12]. K. E. Wenzel, P. Rosset, A. Zell.”Low-Cost Visual Tracking of a Landing Place and Hovering Flight Control with a Microcontroller”, Journal of Intelligent & Robotic Systems, 2009, Vol. 57(1-4), pp. 297- 311.

[13]. K. E. Wenzel, A. Masselli, A. Zell, ”Automatic Take O , Tracking and Landing of a Miniature UAV on a Moving Carrier Vehicle”, Journal of Intelligent & Robotic Systems, Vol. 61, 2011, pp. 221-238.

[14]. G. Xu, Y. Zhang, S. Ji, Y. Cheng, Y. Tian, ”Research on computer vision-based for UAV autonomous landing on a ship”, Pattern Recognition Letters, Vol. 30(6), 2009, pp. 600-605.

[15]. Maria Isabel Ribeiro. 1986. Kalman and Extended Kalman Filters: Concept, Derivation and Properties. [ONLINE] Available at: http://users.isr.ist.utl.pt/~mir/ pub/kalman.pdf. [Accessed 19 July 2016].

[16]. Ramsey Faragher. 1992. Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation. [ONLINE] Available at: https://www.cl.cam.ac.uk /~rmf25/papers/Understanding%20the%20Basis%20of% 20the%20Kalman%20Filter.pdf. [Accessed 23 July 2016].

[17]. Lindsay Kleeman. 2000. Understanding and Applying Kalman Filtering. [ONLINE] Available at: http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_ kalman_ basics.pdf. [Accessed 15 July 2016].

[18]. Greg Welch. 1991. An Introduction to the Kalman Filter. [ONLINE] Available at: http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf. [Accessed 19 July 2016].

[19]. Majid Dadafshar. 2014. ACCELEROMETER AND GYROSCOPES SENSORS: OPERATION, SENSING, AND APPLICATIONS. [ONLINE] Available at: http://pdfserv. maximintegrated.com/en/an/AN5830.pdf. [Accessed 13 August 2016].

[20]. A guide to using IMU. [ONLINE] Available at: http://www.starlino.com /imu_guide.html. [Accessed 12 August 2016].

[21]. InvenSense Inc. 2013. MPU-6000 and MPU-6050 Product Specification. [ONLINE] Available at: https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V 3%20 4.pdf. [Accessed 15 August 2016].

Appendix A

Mathematical modelling

Matlab code

%%%%%% Simulation Code %%%%%%%%

```
function result = simulate(controller, tstart, tend, dt)
    % Physical constants.
    g = 9.81; %g
    m = 0.95; %Kg
    L = 0.165; % L mid of the frame (m)
    k = 3e-6; %k  constant
    b = 1e-7; %drag coefficient

     if nargin < 4
         tstart = 0;
         tend = 4;
         dt = 0.005;
     end
     ts = tstart:dt:tend;
     N = numel(ts);
     I = diag([5e-3, 5e-3, 5e-3]);
     kd = 0.25;
     % Output values
     xout = zeros(3, N);
     xdotout = zeros(3, N);
     thetaout = zeros(3, N);
     thetadotout = zeros(3, N);
     inputout = zeros(4, N);
     % initial PID
     controller_params = struct('dt', dt, 'I', I, 'k', k, 'L', L, 'b', b,
 'm', m, 'g', g);

     % Initial system state.
     x = [0; 0; 50];
     xdot = zeros(3, 1);
     theta = zeros(3, 1);


     if nargin == 0
         thetadot = zeros(3, 1);
     else
         deviation = 300;
         thetadot = deg2rad(2 * deviation * rand(3, 1) - deviation);
     end

     ind = 0;
     for t = ts
         ind = ind + 1;



             %input or controller input.
             if nargin == 0
                 i = input(t);
             else
                 [i, controller_params] = controller(controller_params,
      thetadot);
             end
```

```
            omega = thetadot2omega(thetadot, theta);
            a = acceleration(i, theta, xdot, m, g, k, kd);
            omegadot = angular_acceleration(i, omega, I, L, b, k);


            omega = omega + dt * omegadot;
            thetadot = omega2thetadot(omega, theta);
            theta = theta + dt * thetadot;
            xdot = xdot + dt * a;
            x = x + dt * xdot;

            xout(:, ind) = x;
            xdotout(:, ind) = xdot;
            thetaout(:, ind) = theta;
            thetadotout(:, ind) = thetadot;
            inputout(:, ind) = i;
        end

        result = struct('x', xout, 'theta', thetaout, 'vel', xdotout, ...
                'angvel', thetadotout, 't', ts, 'dt', dt, 'input', inputout);
    end


    % thrust
    function T = thrust(inputs, k)
        T = [0; 0; k * sum(inputs)];
    end

    %torques.
    function tau = torques(inputs, L, b, k)
        tau = [
            L * k * (inputs(1) - inputs(3))
            L * k * (inputs(2) - inputs(4))
            b * (inputs(1) - inputs(2) + inputs(3) - inputs(4))
        ];
    end

    % acceleration
    function a = acceleration(inputs, angles, vels, m, g, k, kd)
        gravity = [0; 0; -g];
        R = rotation(angles);
        T = R * thrust(inputs, k);
        Fd = -kd * vels;
        a = gravity + 1 / m * T + Fd;
    end

    %angular acceleration
    function omegad = angular_acceleration(inputs, omega, I, L, b, k)
        tau = torques(inputs, L, b, k);
        omegad = inv(I) * (tau - cross(omega, I * omega));
    end


% (roll, pitch, yaw) to omega
function omega = thetadot2omega(thetadot, angles)
    phi = angles(1);
    theta = angles(2);
    psi = angles(3);
    W = [
        1, 0, -sin(theta)
```

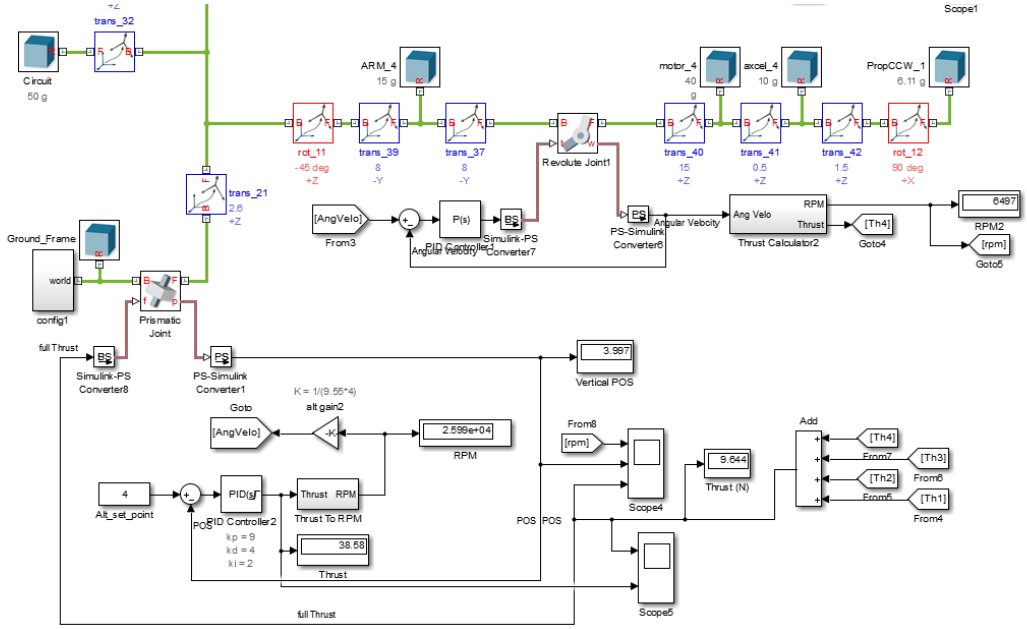
```
            0, cos(phi), cos(theta)*sin(phi)
            0, -sin(phi), cos(theta)*cos(phi)
        ];
        omega = W * thetadot;
    end

    % omega to (roll, pitch, yaw)
    function thetadot = omega2thetadot(omega, angles)
        phi = angles(1);
        theta = angles(2);
        psi = angles(3);
        W = [
            1, 0, -sin(theta)
            0, cos(phi), cos(theta)*sin(phi)
            0, -sin(phi), cos(theta)*cos(phi)
        ];
        thetadot = inv(W) * omega;
    end
```

%%%% Rotation Matrix %%%%%%

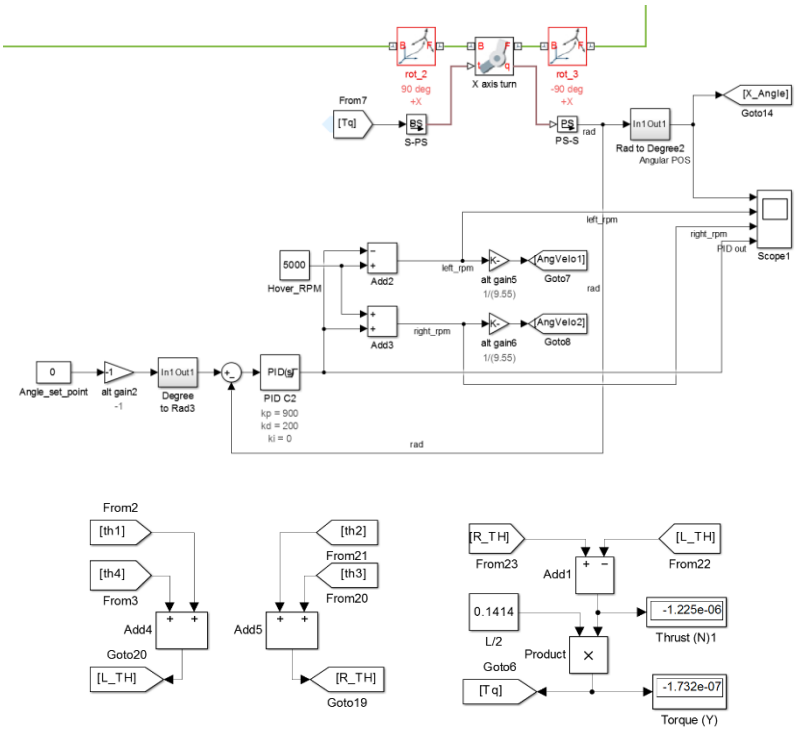
```
function R = rotation(angles)
    phi = angles(3);
    theta = angles(2);
    psi = angles(1);
    I = [0 1 -0.02; -1 0 0; 0 0.002 1];
R = zeros(3);
    R(:, 1) = [
        cos(phi) * cos(psi) - cos(theta) * sin(phi) * sin(psi)
        cos(theta) * cos(psi) * sin(phi) + cos(phi) * sin(psi)
        sin(phi) * sin(theta)
    ];
    R(:, 2) = [
       - cos(psi) * sin(theta) - cos(phi) * cos(theta) * sin(psi)
        cos(phi) * cos(theta) * cos(psi) - sin(phi) * sin(psi)
        cos(phi) * sin(theta)
    ];
    R(:, 3) = [
        sin(theta) * sin(psi)
        - cos(psi) * sin(theta)
        cos(theta)
    ];
end
```

Simulink Model

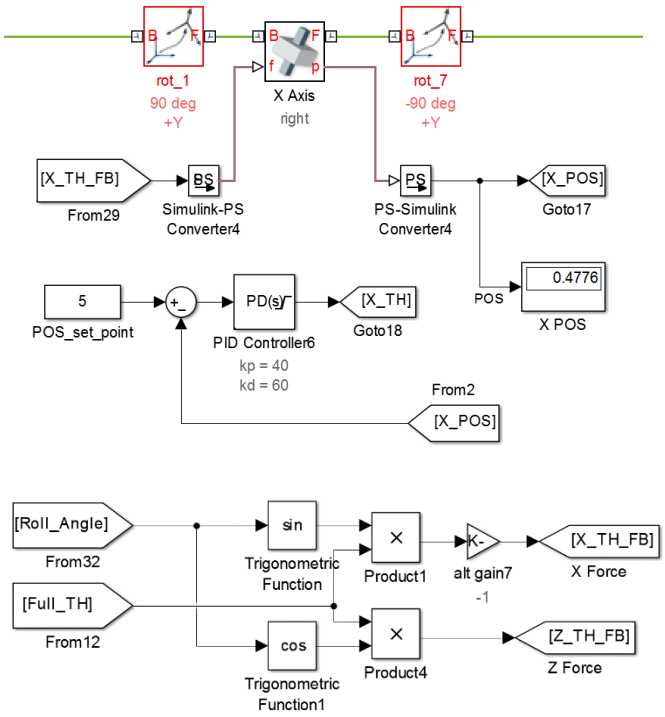
Altitude hold PID controller



Roll PD controller

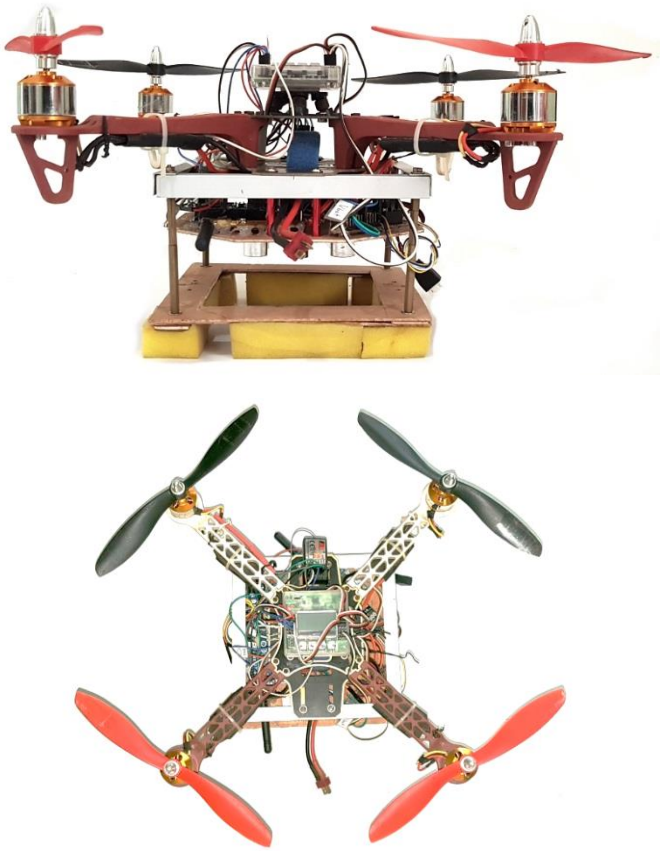


Axis movement controller

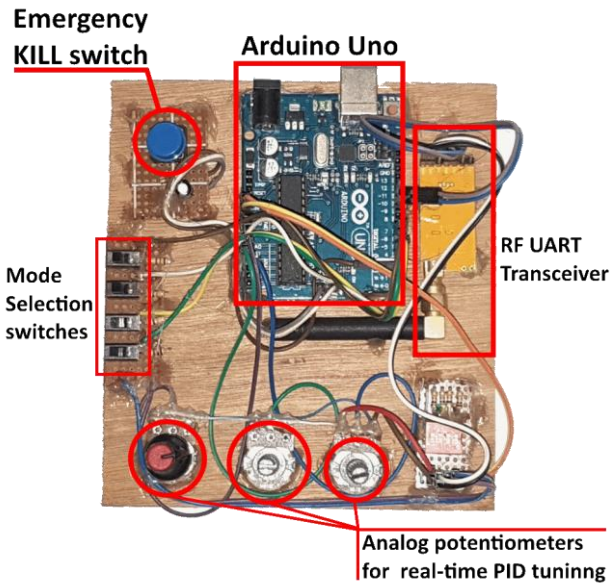


Appendix B

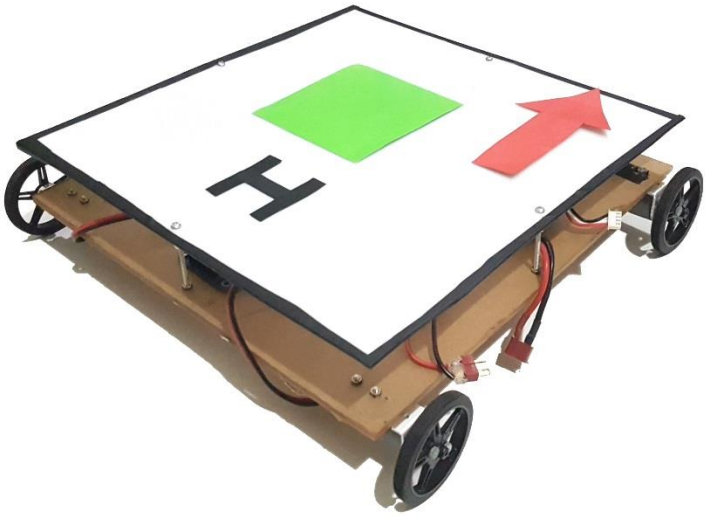
Testing quadcopter system



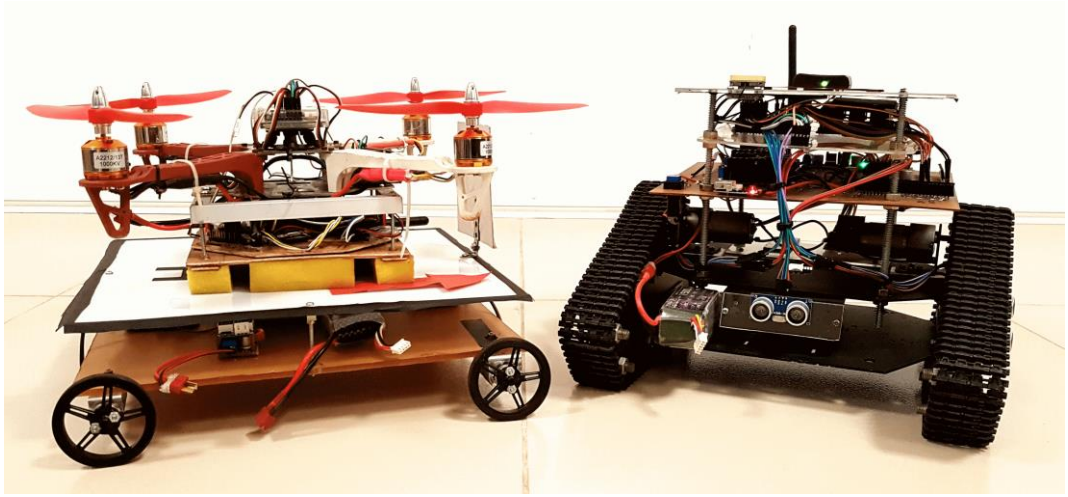
Ground control station



Mobile robot platform



Complete system



Appendix C

Step 1: Define a resolution for the capture. The resolution was set to 704 x 576.

```
width = 704
height = 576

camera.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, width)
camera.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, height)
```

Step 2: Create trackbars for online tuning.

function cv2.createTrackbar is used to create a slide bar feature for changing the HSV values.

```
hh='Hue High'
hl='Hue Low'
sh='Saturation High'
sl='Saturation Low'
vh='Value High'
vl='Value Low'

cv2.createTrackbar(hl, 'image',0,179,nothing)
cv2.createTrackbar(hh, 'image',0,179,nothing)
cv2.createTrackbar(sl, 'image',0,255,nothing)
cv2.createTrackbar(sh, 'image',0,255,nothing)
cv2.createTrackbar(vl, 'image',0,255,nothing)
cv2.createTrackbar(vh, 'image',0,255,nothing)
```

Step 3: define arrays for minimum and maximum range of HSV values.

```
rangeMin = np.array([hul, sal, val], np.uint8)
rangeMax = np.array([huh, sah, vah], np.uint8)
```

Step 4: Processing the image

image transformation to HSV -

```
imgHSV = cv2.cvtColor(frame, cv2.cv.CV_BGR2HSV)
```

Threshold the transformed image using minimum and maximum HSV values –

```
imgThresh = cv2.inRange(imgHSV, rangeMin, rangeMax)
```

Morphological Transformation: Erosion for better results.

```
imgErode = cv2.erode(imgThresh, None, iterations = 5)
```

Gaussian blur for eliminate the noise of the eroded image.

```
blurred = cv2.GaussianBlur(imgErode, (7, 7), 0)
```

Step 5: Canny Edge detection and Find contours of the blurred image.

Canny Edge detection algorithm is used to determine the number of corners in the image

```
edged = cv2.Canny(blurred, 50, 150)
```

finding contours of the detetcted edges.

```
(cnts,_)=cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
```

Step 6: check for contour length which is bound a square shape which has four edges in it.

```
if len(approx) >= 4 and len(approx) <= 6:
```

if condition is true, then determine the properties of the bounded object by contours. Following peace of code will determine the aspect ratio, area, hullArea, and solidity of the object.

```
 (x, y, w, h) = cv2.boundingRect(approx)
 aspectRatio = w / float(h)
 area = cv2.contourArea(c)
 hullArea = cv2.contourArea(cv2.convexHull(c))
 solidity = area / float(hullArea)
```

Step 7: Set the rules for detect only square shape term. Here square shape properties will be set.

```
 keepDims = w > 25 and h > 25
 keepSolidity = solidity > 0.9
 keepAspectRatio = aspectRatio >= 0.8 and aspectRatio <= 1.2
```

Step 8: Draw Contours

At this step the algorithm clarify the detected object is Red colored Square shape and graphically draw an outline of the window frame.

```
if keepDims and keepSolidity and keepAspectRatio and minArea > 50:


   cv2.drawContours(frame, [approx], -1, (0, 255, 0), 4)
```

Step 9: Find the Center of the Object

Method moments was used to determine the center of the tracked object. cX and cY gives the center coordinates of the tracked object.

```
M = cv2.moments(approx)
```

```
(cX, cY) = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

Step 10: Calculating the Area of the detected square shape.

```
area = round(M['m00'])
```

Step 11: Serial Event Generation

Calculated area and the center coordinates (cX, cY) of the detected square shape then sent as arguments to a function named quadMovement () through another function called Storage (). The serial event is generated inside the quadMovement () function. This function calls every 1s second in a separate Thread. Which is declared inside the Storage () function. For every second these data will be sent via RF link to quadcopter.

Define Serial COM port and the baud rate:

```
ser = serial.Serial('COM5', 115200)
```

Serial event exception handler:

```
try:
    ser.close()
    ser.open()

except Exception, e:

    print "error open serial port: " + str(e)
    exit()
```

Serial event generation function:

```
def quadMovement(directionX,directionY):

    value_1 = str(directionX)
    value_2 = str(directionY)

    ser.write("A")
    ser.write(value_1)
    ser.write(",")
    ser.write(value_2)
    ser.write(",")
    ser.write("1212")
    ser.write(",?#")
    print ser.readline()
```

Separate Thread Driven Function For fast event handling:

```
def Storage(quad_X,quad_Y):

    t1 = threading.Thread(target = quadMovement, args = (quad_X, quad_Y))
    t1.start()
    t1.join()
```

Step 12: Displaying the data on the screen

Draw a cross symbol on the center of the detected square shape using cv2.line function:

```
(cX, cY) = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
(startX, endX) = (int(cX - (w * 0.15)), int(cX + (w * 0.15)))
(startY, endY) = (int(cY - (h * 0.15)), int(cY + (h * 0.15)))

cv2.line(frame, (startX, cY), (endX, cY), (255, 0, 0), 2)
cv2.line(frame, (cX, startY), (cX, endY), (255, 0, 0), 2)

Displays a text when there is no target marker is detected:

status = "Searching the Object"
cv2.putText(frame,status, (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0, 0, 255), 2)

Displays when tracking the object:

status_2 = "Following the object"
cv2.putText(frame, status_2, (20, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255, 255,
51), 2)

Displays Center coordinates on screen:

cv2.putText(frame,"X: " + str(cX), (600, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0,
255, 0), 2)
cv2.putText(frame,"Y: "+ str(cY), (700, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0,
255, 0), 2)


Displays the area of the detected object:

cv2.putText(frame,str(area), (20, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(210, 0,
255), 2)
```

Step 13: Distance Calculation

A separate method is calling when distance in needed. This will ultimately returns the Actual length to where it called.

```
def Distance(incom):

    F = 675.498
    W = 10
    A = int(incom)
    P = math.sqrt(A)

    D’ = int((W*F)/P)

    return D’
```

Step 14: check the bounded contours within the range of less than 8 and greater than 5.

```
if len(approx) >= 5 and len(approx) < 8:
```