

# **SURVEILLANCE QUADCOPTER - वेग**

**MET-2017 Major Project  
Semester Report**

Submitted in partial fulfillment of the requirements for the  
degree of

**BACHELOR OF TECHNOLOGY**  
in  
**MECHATRONICS  
ENGINEERING**

By

*Abhishek Tyagi 42370211217*

*Sahil Kumar 41870211217*

*Siddhant Jha 02970211217*

*Yash Arora 03670211217*



Under the guidance of:

**Mr. Arun Gupta (HoD – Mechatronics)**

**&**

**Dr. R.K. Dhammi**

**DEPARTMENT OF MECHATRONICS ENGINEERING  
DELHI INSTITUTE OF TOOL ENGINEERING, OKHLA PHASE-II, NEW DELHI  
(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)**



---

*DEPARTMENT OF MECHATRONICS  
ENGINEERING, DELHI INSTITUTE OF  
TOOL ENGINEERING, OKHLA  
INDUSTRIAL AREA, PHASE – 2, NEW  
DELHI 110020*

---

**CERTIFICATE**

This is to certify that major project titled "**Surveillance Quadcopter - वेग**" is the bona-fide work of student of group .... (Batch 2017-2021), who carried out the research under our supervision. certified further, that to the best of our knowledge the work reported here in does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**Guide**

Dr. R. K. Dhammi

**Co-Guide**

Mr. Arun Gupta

HoD - Mechatronics

The B.Tech Major Project Viva-Voce Examination of Group-\_\_\_\_\_ has been held on \_\_\_\_\_.

---

## ***DECLARATION***

---

It is hereby certified that the work which is being presented in the B. Tech Major Project Report titled " **Surveillance Quadcopter - वेग** " in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the department of Tool Engineering of "DELHI INSTITUTE OF TOOL ENGINEERING" ,Delhi (Affiliated to GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI) is an Authentic record of our own work carried out during a period from march 2021 to June 2021 under the guidance of Mr. Arun Gupta (HoD), Dr. R. K. Dhammi.

The matter presented in the B. Tech Major Project Report has not been submitted by us for the award of any other degree of this or any other Institute.

<i><b>Name</b></i>	<i><b>Enrollment Number</b></i>	<i><b>Signature</b></i>
<i><b>Abhishek Tyagi</b></i>	<i><b>42370211217</b></i>	
<i><b>Sahil Kumar</b></i>	<i><b>41870211217</b></i>	
<i><b>Siddhant Jha</b></i>	<i><b>02970211217</b></i>	
<i><b>Yash Arora</b></i>	<i><b>03670211217</b></i>	

**Place: Delhi Institute of Tool Engineering (DITE), Okhla**

**Date:\_\_\_/07/2021**

---

## ***ACKNOWLEDGEMENT***

---

We would like to stand our sincere thanks to **Professor S. Maji**, worthy **Director-cum Principal** for providing us the facilities to carry out our project work. We express our sincere gratitude **Dr. R. K. Dhammi & Mr. Arun Gupta**, for his valuable Guidance and suggestion throughout our project work. He provided an academic environment of logical enrichment. Advice and stimulation to 3-D printer and provided us her valuable suggestion and precious time in accomplishing our project report. We are thankful to **Dr. Charu Gaur (Co-guide), Assistant Professor** for their time-to-time suggestions to complete our project work. For their valuable Guidance and for providing us an opportunity to undergo this project and learn and design some-thing innovative and productive.

<i>Name</i>	<i>Enrollment Number</i>	<i>Signature</i>
<i>Abhishek Tyagi</i>	<i>42370211217</i>	
<i>Sahil Kumar</i>	<i>41870211217</i>	
<i>Siddhant Jha</i>	<i>02970211217</i>	
<i>Yash Arora</i>	<i>03670211217</i>	

# Contents

## Contents

<b>SURVEILLANCE QUADCOPTER - वेग.....</b>	<b>ii</b>
<b>List of figures.....</b>	<b>1</b>
<b>List of tables.....</b>	<b>2</b>
<b>Nomenclature .....</b>	<b>3</b>
<b>Chapter 1 - Introduction .....</b>	<b>4</b>
<b>Chapter 2 - Literature Survey.....</b>	<b>6</b>
<b>Chapter 3 – Methodology .....</b>	<b>11</b>
3.1 Mathematical model of quadcopter .....	11
3.2 Mathematical model of quadcopter with one failed rotor .....	16
3.3 Attitude controller.....	18
3.4 Trajectory Controller .....	26
<b>Chapter 4 - Simulation.....</b>	<b>28</b>
4.1. Simulation for attitude controller comparison .....	29
4.2. Simulation for trajectory following – feasible landing point .....	30
4.3. Simulation for trajectory following – using a trajectory planner .....	31
<b>Chapter 5 - Results and Discussions.....</b>	<b>34</b>
5.1 Attitude controller comparison .....	34
5.2 Trajectory planner.....	38
5.3 Trajectory tracking .....	40
<b>Chapter 6 - Object detection with Raspberry Pi 4 on the Drone .....</b>	<b>45</b>
<b>Introduction: - .....</b>	<b>45</b>
<b>Background: - .....</b>	<b>45</b>
<b>Problem Statement: - .....</b>	<b>45</b>
<b>Applications: -.....</b>	<b>46</b>
<b>Challenges: -.....</b>	<b>46</b>
<b>Related Works: -.....</b>	<b>46</b>
<b>Bounding Box: -.....</b>	<b>47</b>

<b>Approach:</b> .....	<b>47</b>
<b>Classification and Regression:</b> - .....	<b>47</b>
Two-Stage Method: - .....	47
Unified Method: - .....	48
<b>Architecture Design:</b> .....	<b>49</b>
<b>Experimental Setup:</b> .....	<b>52</b>
<b>Dataset:</b> .....	<b>52</b>
<b>Implementation Details:</b> .....	<b>52</b>
Pre-processing .....	52
Environment .....	52
Network.....	53
<b>Experimental Result Analysis:</b> .....	<b>54</b>
<b>Qualitative Analysis:</b> .....	<b>54</b>
<b>Quantitative Analysis:</b> .....	<b>57</b>
<b>Chapter 7 - A Real-time Face Recognition System using PCA and various Distance Classifiers on Raspberry pi 4</b> .....	<b>58</b>
<b>1 Literature Review</b> .....	<b>58</b>
<b>2.1 Face Detection Approaches</b> .....	<b>58</b>
<b>2.2 Face Recognition Approaches</b> .....	<b>58</b>
<b>2 Proposed System</b> .....	<b>58</b>
<b>3.1 Steps Involved</b> .....	<b>59</b>
<b>3.2 Various Distance metrics</b> .....	<b>61</b>
<b>3.3 The decision on the test</b> .....	<b>62</b>
<b>3 Database</b> .....	<b>62</b>
<b>4 Codes</b> .....	<b>62</b>
<b>5 The Yale Face Database B</b> .....	<b>62</b>
<b>5.1 The AT &amp; T Database of Faces (ORL)</b> .....	<b>63</b>
<b>6 Results</b> .....	<b>64</b>
<b>6.1 Overall Results</b> .....	<b>64</b>
<b>6.2 The Yale Face Database B</b> .....	<b>64</b>

6.3 AT& T Face Database .....	65
6.1 A Realtime Application: Processtime.....	66
Conclusions of Various Computer Vision Surveillance Techniques on Drone System.....	67
Further work .....	68
SLAM Based Autonomous Drone: .....	68
References .....	69

## List of figures

Figure 1: Control architecture (taken from [3]) .....	13
Figure 2: Comparison between controllers on the basis of total error (taken from [4]) .....	14
Figure 3: Inertial and body frames of quadcopter (taken from [1]) .....	17
Figure 4: Quad plant Simulink Block .....	21
Figure 5: Block diagram of Attitude Controller.....	24
Figure 6: Desired path with switching .....	40
Figure 7: A VR model of quadcopter during simulation (viewpoint 1).....	42
Figure 8: A VR model of quadcopter during simulation (viewpoint 2).....	42
Figure 9: Step responses of controllers – $\Phi$ .....	43
Figure 10: Step responses of controllers – $\theta$ .....	44
Figure 11: Step responses of controllers – $\Psi$ .....	45
Figure 12: Step responses of controllers – $Z$ .....	46
Figure 13: Image containing different shapes in different colors .....	47
Figure 14: Black circle given as output .....	47
Figure 15: Maze input to the trajectory planner.....	48
Figure 16: Path generated by the trajectory planner .....	48
Figure 17: Actual path followed .....	49
Figure 18: Coordinate wise comparison of desired and actual paths.....	50
Figure 19: Plots of attitude variables vs Time .....	50
Figure 20: Desired path generated by trajectory planner .....	51
Figure 21: Actual path followed by quadcopter.....	52
Figure 22: Coordinate wise comparison of desired path and actual path .....	52
Figure 23: Plots of Attitude variables vs Time .....	53



## List of tables

Table 1: Parameter values for quad plant .....	37
Table 2: Initial conditions for trajectory control simulation 1 .....	37
Table 3: Initial conditions for trajectory control simulation 2 .....	38
Table 4: Gain values for attitude controller 1 .....	39
Table 5: Gain values for attitude controller 2 .....	39
Table 6: Gain values for attitude controller 3 .....	39
Table 7: Gain values for trajectory controller .....	40
Table 8: Gain values for trajectory controller .....	41
Table 9: Characteristic parameters to a step input for $\Phi$ .....	44
Table 10: Characteristic parameters to a step input for $\theta$ .....	44
Table 11: Characteristic parameters to a step input for $\Psi$ .....	45
Table 12: Characteristic parameters to a step input for $Z$ .....	46
Table 13: Computational time of controllers .....	46

## Nomenclature

$\{O\}(O,X,Y,Z)$	Inertial frame
$\{B\}(O_B,X_B,Y_B,Z_B)$	Body frame
$\varepsilon$	Quadcopter position w.r.t $\{O\}$ , m
$\eta$	Quadcopter Euler angles w.r.t $\{O\}$ , rad
$\Phi$	Roll angle, rad
$\theta$	Pitch angle, rad
$\Psi$	Yaw angle, rad
$P,Q,R$	Angular velocities about $X_B,Y_B,Z_B$ respectively, rad/s
$V_X,V_Y,V_Z$	Linear velocities about $X_B,Y_B,Z_B$ respectively, m/s
$C_D$	Thrust coefficient of the motor
$\omega_i$	Rotational speed of $i^{\text{th}}$ rotor, rad/s
$A$	Cross-sectional area of the propeller's rotation, $\text{m}^2$
$r$	Radius of rotor, m
$T_i$	Thrust given by $i^{\text{th}}$ rotor, N
$A_x, A_y, A_z$	Linear drag coefficients in the X,Y,Z respectively, N.s/m
$M_\Phi$	Rolling moment, N.m
$M_\theta$	Pitching moment, N.m
$M_\psi$	Yawing moment, N.m
$L$	Distance between the center of propeller and the center of quadcopter, m
$B$	Torque coefficient of motor
$I_R$	inertia moment of rotor, $\text{kg.m}^2$
$A_r$	Rotational drag coefficient, N.m.s
$I$	Inertia matrix, $\text{kg.m}^2$
$m$	Mass of quadcopter, kg
$g$	Acceleration due to gravity, $\text{m/s}^2$

## Chapter 1 - Introduction

A Quadcopter is a rotor-based, unmanned aerial vehicle. Quad copters are becoming increasingly popular because of their small size and high maneuverability and find applications in diverse fields. The dynamics of a quadcopter is highly non-linear. Furthermore, it is an under-actuated system with six degrees of freedom and four control inputs. The thrust as well as the torques required for tilting the quadcopter are the control inputs which determine the motion of the vehicle. The thrust as well as torques are generated by adjusting the rotor speeds. The thrust generated by the rotor blades is always in the direction of the central axis of the quadcopter. Therefore, to achieve propulsion in a particular direction, the axis of quadcopter should be tilted with respect to the vertical. The translational motion of a quadcopter is hence coupled with its angular orientation, making quadcopter dynamics and control very complex.

Quad copters have applications in many fields, some of which are listed below.

- Reconnaissance – used to gather military intelligence by scouting enemy territory. Because of their small size and minimal noise generated, they can move undetected.
- Aerial surveillance – road patrol, home security, law and order. The ease of motion between points through air and large visibility of the surroundings, especially with a strong camera makes quad copters a prime candidate for aerial surveillance needs.
- Used in motion picture film making and photography for aerial shots and views.
- Assistance for search and rescue operations in disaster struck areas or in case of fire.
- Used in automation systems in industries for material handling purposes.
- Delivery of goods and items.
- Used for 3D modelling of terrains or large structures as well as thermal imaging.

Failure of quadcopter may occur due to many reasons such as

- Electronic Speed Control (ESC) burn out – ESC may burn out if the current exceeds the maximum permissible current. This causes the propeller associated with the ESC to stop spinning and can lead to failure.
- Damage to motor, whether due to physical damage or exceeding the maximum current value can cause the motor to stop functioning mid-flight or may cause loss in efficiency.
- Any physical damage to the propeller blades such as dents, nicks, cuts etc. can cause vibrations and may cause the propeller blade to come off mid-flight.
- Bending of propeller blades can lead to a decrease in lift and increased noise during operation leading to a decrease in efficiency.

The loss of quadcopter propeller blades can cause the quadcopter to crash. Apart from the monetary losses associated with the damage to quadcopter parts, it can have many negative consequences. Loss of a quadcopter used for reconnaissance work can lead to loss of valuable military intelligence and causes the risk of it being discovered by the enemy. In motion picture film making, thermal imaging photography etc., the equipment mounted on the quadcopter are very costly and propeller failure can lead to the damage of valuable equipment. In search and rescue operations in disaster affected regions, the failure of the quadcopter can lead to possible delays, increasing the risk on the life of affected people. In material handling systems, failure may lead to damage of costly parts. Added to all this, there is also the risk of the quadcopter crashing on to people and causing injuries especially in public spaces.

The potential risk of loss of a propeller is high in many of the situations in terms of cost as well as other factors. Hence, a mechanism for the control of a quadcopter against the possibility of failure is a necessity. Quadcopter control, even with four propellers functioning, is a complex problem because it is an under-actuated and highly non-linear system. Various control algorithms like PID and feedback linearization are used for the purpose of control. The control problem becomes more complex when there is complete loss of one or more propellers. The first stage in devising an algorithm for such a case is fault detection. If left unchecked, the fault leads to the failure of the system. In this case, the loss of a propeller is the fault. After detecting the nature of the fault and the component in which the fault has occurred, the next stage is replacing the model of the system with a new model which takes into account the effect of the fault. The original model does not accurately represent the behavior of the system in the event of a fault. Finally, a controller should be devised to control the system represented by the new model. The controller used for this purpose should ensure that the quadcopter stays in flight regardless of the failure of a rotor and that its motion could be controlled sufficiently enough to land it safely on a desired location in the vicinity.

The quadcopter has to maneuver through obstacles at times. Image processing is used for this purpose in this project. One variant of the model involves generating a path through a maze. The image (resolution: 50x50) of the maze is fed into MATLAB as a 50x50x3 array. Each pixel containing an obstacle is assigned a high cost. The cost assigned to the pixel reduces as the quadcopter move away from the obstacles. The aim is to move from the source to the destination while minimizing the cost. The path thus obtained would not only be devoid of obstacles, but also be at a safe distance from them.

Another variant involves scanning the image for safe landing points in case failure occurs. Given the landing points, the one nearest to the quadcopter at the time of failure could be found out using a simple distance formula. But this may not always be the most suitable landing point. The velocity of the quadcopter when the failure occurs should also be taken into account. Instead of going to the nearest landing point, a better alternative is to move to the one which can be reached in the shortest time.

## Chapter 2 - Literature Survey

The employment of quadcopters in challenging applications like rescue, surveillance comes from its ability to perform aggressive maneuvers and follow complex trajectory in 3D space. For these applications, precise angle handling of quadcopters is important. This calls for a clear understanding of the system dynamics before designing a controller to achieve the purpose.

Mathematical modeling is the first and most critical step towards understanding the system dynamics and monitoring the response. The differential equations governing the quadcopter dynamics derived using the two most popular approaches (Newton-Euler equations and Euler-Lagrange equations) is presented in [1]. While the derivations are listed for a simplified model, the paper also presents a more realistic model for the quadcopter with the inclusion of the drag force caused by air resistance. Other complex dynamic interactions like aerodynamic effects and blade flapping have been neglected due to challenges in modeling. A matrix approach for the derivation of governing equations is elaborated in [2]. The concept of mixed frame of reference for describing the state variables along with its convenience in developing the mathematical model is described. A more detailed study of the dominant aerodynamic effects on the quadcopter is addressed in [3]. The derivation for exogenous forces on the body of quadcopter, considering even the complex aerodynamic phenomena neglected in the simplified models used in [1] and [2] is also given. The steady state thrust and reaction torque (due to rotor drag) for a hovering rotor in free air is modelled using momentum theory. The lumped parameter approximation used for thrust and reaction torque expressions is shown here, with the constant value obtained from static thrust tests. The dynamic model considering flapping dynamics and rotor stiffness for induced drag is presented here, though these terms are minor considerations from robotics perspective. It is mentioned that high gain control can dominate all the secondary aerodynamic effects, and high-performance control can be achieved using the simple static thrust model.

Different control methods for attitude stabilization have been researched, including PID controllers ([1], [3], [5] and [8]), back-stepping controller [4], sliding mode controller ([4] and [12]), linear quadratic regulator ([8] and [10]) and feedback linearization control ([9], [10], [11] and [12]). A hierarchical control approach is implemented in [3], with nested feedback loops as shown in the figure. The control problem is decoupled into position controller and attitude controller, with the position controller providing the set-points to the attitude controller.

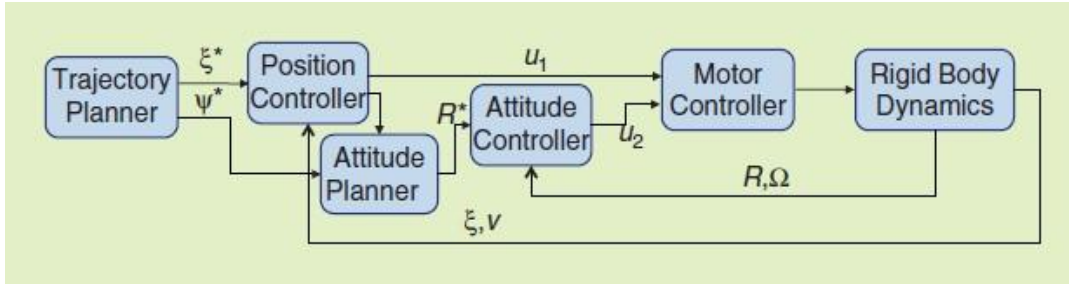


Figure 1: Control architecture (taken from [3])

An exponentially converging attitude controller is presented considering the measure of error in rotations. A skew-symmetric matrix is generated to go from actual attitude vector to the desired attitude vector. For small deviations from the hover position, the error matrix is linearized and a PD controller gives satisfactory performance. The control law for large deviations is also given, where the error matrix is not linearized. A much simpler PD controller is implemented in [1], where the expression for thrust and torque components are obtained from the error in attitude values, and the individual rotor speeds are calculated from the thrust and torque commands. Here, the motor dynamics is neglected while deriving the force and torque expressions from the motor-propeller system. A PID controller with an additional term for angular acceleration feedback is used for attitude control in [5]. This additional term allows the gains to significantly increase, thereby yielding higher bandwidth. A first order time delay in thrust is also included in the model for controlling each angle.

Among the nonlinear controllers, the most popular approach is feedback linearization control which in turn has two approaches. Both the approaches are discussed in [10] - Exact linearization and non-interacting control via dynamic feedback and Dynamic inversion with zero-dynamics stabilization. The former approach involves the use of dynamic feedback control law, and the nonlinear system cannot be solved using static feedback control. With the position variables chosen as output function, the thrust input is delayed till its second derivative and the system is extended to include the thrust input and its first derivative as the system states. The extended system fulfills the condition for feedback linearization and can be transformed via dynamic feedback into a system which is fully Linear and controllable. The latter approach uses attitude variables as the output variables, and dynamic inversion is carried out with small angle approximation. Here, the attitude variables are differentiated till the input terms appear and the system is not extended. The paper also shows the implementation of Linear Quadratic Regulator (LQR) on the quadcopter, where model linearization is performed using small angle approximation, and the LQR function in MATLAB is used to solve the algebraic equation using Riccati's method and obtain the control gains.

Feedback linearization with position variables and yaw angle as output terms of interest is also shown in [12], where the equations are differentiated twice till the input terms appear. Repeated differentiation of dynamic equations causes it to be very sensitive to noise, along with high cost of

computation. Here, the computation is reduced by assuming small angle approximation which also lowers the extent of nonlinearity in the system. Feedback linearization by dynamic inversion is discussed in [10] and [11], where the attitude variables are considered as outputs of interest. A two layer architecture is adopted for structured tracking, with a dynamic inversion inner loop and an internal dynamics stabilization outer loop. Dynamic inversion is carried out with small angle approximations for the Euler angles, which gives a simplified expression for the matrix to be inverted. A back stepping approach is used in [10] to design the linear controller for the linearized dynamics and the residual dynamics. The paper also presents a detailed stability analysis for the two controllers. However, in [11] the traditional PD controller is used to provide the linear control inputs in the inner loop and a PID is designed to perform trajectory following along with internal dynamics stabilization.

A comparative study between these different controllers for attitude stabilization and control is addressed in [4]. The controllers are applied to the system and analyzed separately to find the optimum controller for the quadcopter. As seen in the figure where total error is used to evaluate the performance of different controllers, the sliding mode technique proves to be the superior to other techniques. But the PD controller shows reasonable performance compared to the sliding mode technique, and as its implementation is much easier most of the literature contains PD controller in the model. Inverse control based on feedback linearization is also easy to implement, with better settling time compared to PID and sliding mode controllers though the response is slower.

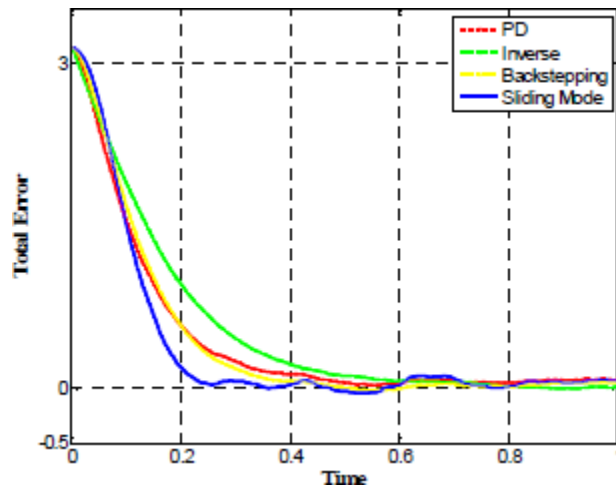


Figure 2: Comparison between controllers on the basis of total error (taken from [4])

A Comparison between PID control and LQR control is given in [8]. Here, multiple PID controllers are designed for a near hover condition neglecting the gyroscopic effects. In order to implement the LQR, the system is linearized around each state. Linearization around the equilibrium point ignoring the gyroscopic effects causes a huge drift from reality, and is avoided.

Feedback linearization controller and sliding mode controllers have superior performance compared to PID as seen in [4], and a detailed analysis between these two controllers is performed in [12]. While feedback linearization controller is simpler to implement, uncertainty in the dynamic model can severely affect performance, and even cause instability. In addition, the dependence on higher derivative terms of states makes it highly sensitive to external disturbances. The sliding mode controller is a more robust approach which compensates for model uncertainties and external disturbances. However, handling these uncertainties causes very high input gains and is a serious problem in power-limited systems like mini quadcopters. Feedback linearization controllers also use more efficient inputs without chattering, compared to sliding mode controller.

All the above papers that implemented feedback linearization (both approaches) have used small angle approximation. This may hold well in near hover conditions, but for a quadcopter involved in trajectory following or sometimes complex motions will have the Euler angles reaching high values. So, a more general approach for feedback linearization is required, which will be addressed through this project.

In addition to the controller, the motor dynamics and their interactions with the drag forces on the propellers is also modelled in [3], with a first order linear approximation. As rotor speed drives the dynamic model, high-quality control of the motor speed is critical for the overall control of the vehicle. Direct voltage control is sufficient in most cases, as the steady state motor speed is directly proportional to the voltage supplied. The performance of the controller is limited by the amount of current that can be supplied by the batteries. Assuming that the battery is able to supply the required current for all rotor speeds and current levels do not exceed the limiting value at any stage, the motor controller can be removed from the system.

Trajectory tracking is a widely studied problem for quadcopters. A lot of literature has focused on nonlinear methods like input-output linearization using differential flatness theory and backstepping control which enables acrobatic maneuvers. For normal trajectory tracking, such approaches are not necessary. Trajectory control in [1] is done using a heuristic approach to generate a symmetric function for jounce to control the acceleration and in turn determine the control inputs to achieve the desired trajectory. A PD controller is integrated into the heuristic method for better response to the disturbances and to stabilize the quadcopter during its trajectory tracking. In [3], the dynamics is linearized about the desired trajectory. An acceleration vector command is computed to minimize the error in the trajectory. The commands for roll and pitch are then calculated based on this acceleration vector and desired yaw angle and fed to the attitude controller. A position controller without linearization is also addressed, where the position error is projected along the yaw axis. A trajectory planning algorithm is discussed in [3], where the problem is simplified by assuming differential flat theory for quadcopters that respect the dynamics of the under actuated system. The trajectory is generated by minimizing a cost function derived from jounce and yaw accelerations. Hence, real time planning is carried out to generate the optimal path. Another trajectory planning algorithm is illustrated in [5]. Here, a sequence of desired



waypoints is inputted and a dynamically feasible trajectory is generated that traverses the waypoints in minimum time while satisfying acceleration and velocity constraints. The controller consists of a piecewise PI control in the along direction, and PID in the cross-track direction which provides the respective control inputs to follow the desired path. The controllers mentioned above require derivatives of the desired path to be given as input, which limits aggressive maneuvers of the quadcopter. This calls for a trajectory controller that does not depend on the higher order derivatives and can enable complex motions of the quadcopter.

Based on the literature survey, the following objectives were established for this project:

1. Implement a linear and nonlinear controller for attitude control of a quadcopter, and perform a comparative study for the same.
2. Integrate a trajectory controller into the attitude controller, to follow the path commands given by the trajectory planner to the nearest landing point. Determine the coordinates of the landing points in the inputted map using image processing and identify the landing site nearest to the point of failure.
3. Develop a trajectory planning algorithm that helps traverse a maze without hitting the walls and test the trajectory controller for this complex trajectory.
4. Develop a failure detection module for the quadcopter to enable switching of controllers in order to ensure stability of the system.

## Chapter 3 – Methodology

### 3.1 Mathematical model of quadcopter

The kinematics and dynamics of a quadcopter can be clearly understood by considering two frames of reference: earth inertial frame  $\{O\}$  and body fixed frame  $\{B\}$ . The earth inertial frame is defined with gravity pointing in the negative  $z$  direction, and the coordinate axes of the body frame are along the arms of the quadcopter. 4 DC motors are placed at the extremities of all the arms, with a propeller mounted on them to provide the required thrust. In the structure shown in figure 3, Motors 1 and 3 rotate in the counter-clockwise direction with angular velocities  $\omega_1$  and  $\omega_3$ , whereas motors 2 and 4 rotate in the clockwise direction  $\omega_2$  and  $\omega_4$ .

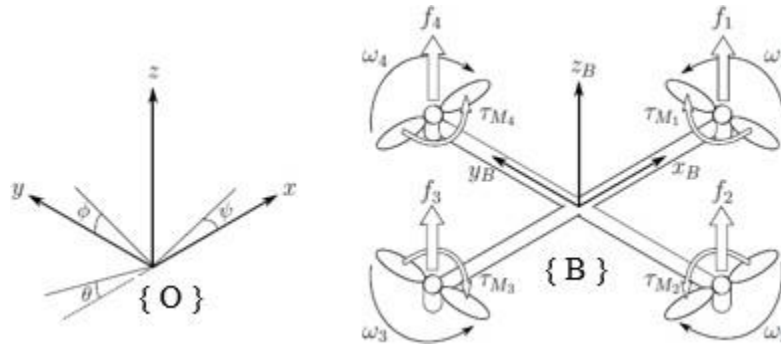


Figure 3: Inertial and body frames of quadcopter (taken from [1])

The absolute position of the center of mass of the quadcopter is expressed in the inertial frame as  $\varepsilon = [X \ Y \ Z]^T$ . The attitude or the angular position is defined in the inertial frame with the ‘roll-pitch-yaw’ Euler angles  $\eta = [\Phi \ \theta \ \Psi]^T$ . The linear velocities  $V_B = [V_X \ V_Y \ V_Z]^T$  and angular velocities  $v = [P \ Q \ R]^T$  are defined in the body frame.

The relationship between these two frames is expressed using the rotation matrix  $R_1$

$$R_1 = \begin{bmatrix} C_\Psi C_\theta & C_\Psi S_\theta S_\Phi - S_\Psi C_\Phi & C_\Psi S_\theta C_\Phi + S_\Psi S_\Phi \\ S_\Psi C_\theta & S_\Psi S_\theta S_\Phi + C_\Psi C_\Phi & S_\Psi S_\theta C_\Phi - C_\Psi S_\Phi \\ -S_\theta & C_\theta S_\Phi & C_\theta C_\Phi \end{bmatrix} \quad (3.1)$$

Where  $C_\theta = \cos(\theta)$  and  $S_\theta = \sin(\theta)$ .  $R_1$  is orthogonal, which implies  $R^{-1} = R^T$  which is the

rotation matrix from { B } to { O }.

Since all the motors are identical, the derivation is explained for a single one. The thrust acting on the quadcopter by a single motor-propeller system is given by momentum theory:

$$T_i = C_D \rho A r^2 \omega_i^2 \quad (3.2)$$

Where  $C_D$  is thrust coefficient of the motor,  $\rho$  is the density of air,  $A$  is the cross-sectional area of the propeller's rotation,  $r$  is the radius of rotor and  $\omega_i$  is the angular speed of the rotor. For simple flight motion, a lumped parameter approach is considered to simplify the above equation to

$$T_i = K \omega_i^2 \quad (3.3)$$

Combining the thrust from all the 4 motor-propeller system, the net thrust in the body frame Z direction is given by:

$$T = K \sum \omega^2 \quad (3.4)$$

Therefore, the net thrust acting on the quadcopter in the body frame is:

$$F^B = [0 \quad 0 \quad T]^T \quad (3.5)$$

In addition to thrust, a drag force also acts on the quadcopter which is a resisting force. It has components along the coordinate axes in the inertial frame directly proportional to the corresponding velocities. The drag force is given in the component form as

$$F^D = \begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \quad (3.6)$$

where  $A_x$ ,  $A_y$  and  $A_z$  are the drag coefficients in the  $x$ ,  $y$  and  $z$  directions.

If all the rotor velocities are equal, the quadcopter will experience a force in  $z$  direction will move up, hover or fall down depending on the magnitude of the force relative to gravity. The moments acting on the quadcopter cause pitch, roll and yaw motion. Pitching moment  $M_\phi$  occurs due to difference in thrust produced by motors 2 and 4. Rolling moment  $M_\theta$  occurs due to difference in thrust produced by motors 1 and 3.

$$M_\phi = L(T_4 - T_2) \quad (3.7)$$

$$M_\theta = L(T_3 - T_1) \quad (3.8)$$

in which  $L$  is the distance between the center of propeller and the center of quadcopter.

Yawing moment  $M_\psi$  is caused by the drag force acting on all the propellers and opposing their rotation. Again from the lumped parameter approach,

$$\tau_{Mi} = B\omega^2 + I_R\dot{\omega}_i \quad (3.9)$$

where  $\tau_{M1}$  is the torque produced by motor 1,  $B$  is the torque constant,  $I_R$  is the inertia moment of rotor. The effect of  $\dot{\omega}_1$  is very small and can be neglected.

$$M_\psi = B(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \quad (3.10)$$

The rotational moment acting on the quadcopter in the body frame is:

$$M^B = [M_\phi \ M_\theta \ M_\psi]^T \quad (3.11)$$

There is also a rotational drag which is a resistive torque that acts on the body frame which is proportional to the body from angular velocities. The rotational drag is given by:

$$M^R = [A_r P \ A_r Q \ A_r R]^T \quad (3.12)$$

where  $A_r$  is the rotational drag coefficient.

The model presented here has been simplified by ignoring several complex effects like, blade flapping (deformation of blades at high velocities and flexible materials), surrounding wind velocities etc.

Newton-Euler formulation is used to derive the dynamic equations of motion for the quadcopter. The quadcopter is assumed to have a symmetrical structure, so the Inertia matrix is diagonal and time-invariant, with  $I_{XX} = I_{YY}$ .

$$I = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \quad (3.13)$$

In the body frame, the force producing the acceleration of mass  $m \dot{V}_B$  and the centrifugal force  $v \times (m V_B)$  are equal to the gravity  $R^T G$  and the total external thrust  $F^B$  and the aero dynamical drag force  $R^T F^D$ .

$$m \dot{V}_B + v \times (m V_B) = R^T G + F^B - R^T F^D \quad (3.14)$$

In the case of a quadcopter, it is convenient to express the dynamics with respect to a mixed frame  $\{ M \}$  with the translational dynamics with respect to the inertial frame and  $\{ O \}$  and the rotational dynamics with respect to the body frame  $\{ B \}$ .

In the inertial frame, centrifugal effects are negligible. The only forces coming into play are the gravitational force, thrust, drag and acceleration of the mass of quadcopter.

Rewriting this,

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + R \frac{F^B}{m} - \frac{F^D}{m} \quad (3.16)$$

Making the following substitution and taking the component form gives the dynamic equation for translational motion:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} U \\ V \\ W \end{bmatrix} \quad (3.17)$$

$$\dot{U} = (\sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi) \frac{T}{m} - \frac{A_x}{m} U \quad (3.18 \text{ a})$$

$$\dot{V} = (-\sin \phi \cos \psi + \cos \phi \sin \theta \sin \psi) \frac{T}{m} - \frac{A_y}{m} V \quad (3.18 \text{ b})$$

$$\dot{W} = -g + (\cos \phi \cos \theta) \frac{T}{m} - \frac{A_z}{m} W \quad (3.18 \text{ c})$$

Again, considering the rotational dynamics in the body frame, the angular acceleration of the inertia  $I \dot{v}$ , the centripetal forces  $v \times (I v)$  and the gyroscopic forces  $\mathcal{T}$  are equal to the external torque  $M^B$  and the torque generated due to aero dynamic drag.

$$I \dot{v} + v \times (I v) + \mathcal{T} = M^B - M^D \quad (3.19)$$

Rewriting this equation,

$$\dot{v} = I^{-1} \left( - \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \begin{bmatrix} I_{xx} P \\ I_{yy} Q \\ I_{zz} R \end{bmatrix} - I_R \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Omega + M^B - M^D \right) \quad (3.20)$$

$$\dot{P} = \left( \frac{I_{XX} - I_{YY}}{I_{ZZ}} \right) QR - \frac{I_R}{I_{XX}} Q \Omega + \frac{M_\Phi}{I_{XX}} - \frac{A_r}{I_{xx}} P \quad (3.21 \text{ a})$$

$$\dot{Q} = \left( \frac{I_{ZZ} - I_{XX}}{I_{YY}} \right) PR - \frac{I_R}{I_{YY}} P \Omega + \frac{M_\theta}{I_{YY}} - \frac{A_r}{I_{yy}} Q \quad (3.21 \text{ b})$$

$$\dot{R} = \left( \frac{I_{XX} - I_{YY}}{I_{ZZ}} \right) PQ + \frac{M_\Psi}{I_{ZZ}} - \frac{A_r}{I_{zz}} R \quad (3.21 \text{ c})$$

The transformation of angular velocities from body frame to inertial frame is given by:

$$\begin{bmatrix} \dot{\Phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \quad (3.22)$$

Using the complete equations of motion describing the dynamics of the system, a quadcopter plant is created in Simulink using the Level-2 S-Function block. For the plant, the 4 rotor speeds  $\{\omega_1, \omega_2, \omega_3, \omega_4\}$  are considered as the inputs and the 12 states in the mixed frame  $\{U, V, W, P, Q, R, \Phi, \theta, \Psi, X, Y, Z\}$  as the outputs.

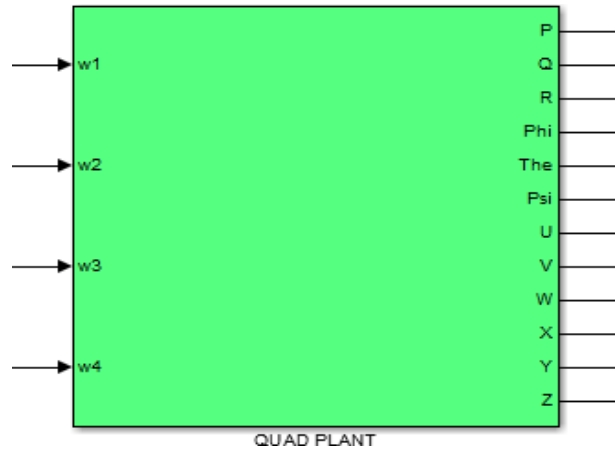


Figure 4: Quad plant Simulink Block

Using the equation of thrust and torque mentioned in dynamics of quadcopter and the above four equations of thrust and torques, the values of the angular velocities at each of the four rotors are obtained and shown in Equation 3.25.

$$\begin{pmatrix} T \\ m_\phi \\ m_\theta \\ m_\psi \end{pmatrix} = \begin{pmatrix} K & K & K & K \\ 0 & KL & 0 & -KL \\ -KL & 0 & KL & 0 \\ -B & B & -B & B \end{pmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} \quad (3.23)$$

Taking inverse of the above matrix, the following equations are obtained for individual rotor speeds:

$$\omega_1^2 = \frac{T}{4k} - \frac{m_\theta}{2kl} - \frac{m_\psi}{4b} \quad (3.24 \text{ a})$$

$$\omega_2^2 = \frac{T}{4k} - \frac{m_\phi}{2kl} + \frac{m_\psi}{4b} \quad (3.24 \text{ b})$$

$$\omega_3^2 = \frac{T}{4k} + \frac{m_\theta}{2kl} - \frac{m_\psi}{4b} \quad (3.24 \text{ c})$$

$$\omega_4^2 = \frac{T}{4k} + \frac{m_\phi}{2kl} + \frac{m_\psi}{4b} \quad (3.24 \text{ d})$$

These  $\omega$  are then used to calculate the current states of the quadcopter as mentioned before. This model is used to develop both the attitude and trajectory controller along with implementation of the trajectory planner. The model described in the following session is a special case, and is used only to test the failure detection module (Section 3.6).

### 3.2 Mathematical model of quadcopter with one failed rotor

The dynamics of the quadcopter remains almost the same in the case of failure of one rotor. However, the control problem becomes increasingly complex. It becomes impossible to control the full attitude of a quadcopter with only three functional rotors. Without loss of generality, it can be assumed that the failed rotor is rotor number 2. This means that the torque control input  $M_\phi$  is lost as the torque can now be provided only in one direction. So the spinning of rotor 4 now creates an unbalanced torque. In order to avoid the toppling of the quad, the rotor 4 velocity must be minimized and this creates an unbalance in yaw. Any attempt to maintain the quad in hover in such a case implies that the yaw control must be relinquished. The total number of control inputs reduces to three. This changes only the right-hand side of the dynamical equations which deals with external driving forces and torques which are the control inputs.

With rotor 2 encountering failure,  $\omega_2 = 0$ . Therefore net thrust is given by

$$T = K(\omega_1^2 + \omega_3^2 + \omega_4^2) \quad (3.25)$$

and thrust due to each rotor is  $T_i = K\omega_i^2$ ,  $i = 1, 3, 4$ , and the torques are given by

$$M_\phi = L * T_4 \quad (3.26)$$

$$M_\theta = L(T_3 - T_1) \quad (3.27)$$

$$M_\psi = B(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \quad (3.28)$$

As  $M_\phi$  is no longer a control input, the equation for  $P$  is modified to express  $M_\phi$  in terms of the other control inputs. The remaining equations stay the same. The modified equations governing the rotational dynamics are given below

$$\dot{P} = \left( \frac{I_{xx} - I_{yy}}{I_{zz}} \right) QR - \frac{I_R}{I_{xx}} Q\Omega + \frac{0.5l \left( T - \frac{M_\psi}{d} \right)}{I_{xx}} - \frac{A_r}{I_{xx}} P \quad (3.29 \text{ a})$$

$$\dot{Q} = \left( \frac{I_{zz} - I_{xx}}{I_{yy}} \right) PR - \frac{I_R}{I_{yy}} P\Omega + \frac{M_\theta}{I_{yy}} - \frac{A_r}{I_{yy}} Q \quad (3.29 \text{ b})$$

$$\dot{R} = \left( \frac{I_{xx} - I_{yy}}{I_{zz}} \right) PQ + \frac{M_\psi}{I_{zz}} - \frac{A_r}{I_{zz}} R \quad (3.29 \text{ c})$$

The angular velocities of rotors required for desired values of control inputs are given by:

$$\begin{bmatrix} T \\ M_\theta \\ M_\psi \end{bmatrix} = K \begin{bmatrix} 1 & 1 & 1 \\ -l & -l & 0 \\ -d & -d & d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \quad (3.30)$$

Taking the inverse, we get the equations for the three rotor speeds as

$$\omega_1^2 = \frac{T}{4k} - \frac{m_\theta}{2kl} - \frac{m_\psi}{4b} \quad (3.31 \text{ a})$$

$$\omega_3^2 = \frac{T}{4k} + \frac{m_\theta}{2kl} - \frac{m_\psi}{4b} \quad (3.31 \text{ b})$$

$$\omega_4^2 = \frac{T}{2k} + \frac{m_\psi}{2b} \quad (3.31 \text{ c})$$



### 3.3 Attitude controller

A quadcopter consists of mainly six outputs of interest ( $\Phi, \theta, \Psi, X, Y, Z$ ) with only four control inputs. This is solved by decoupling it into two distinct control loops (figure 5), inner loop dealing with the attitude variables and the outer variable dealing with the position variables. The angular motion of the quadcopter does not depend on the translational components, whereas the translational motion depends on the Euler angles. So the aim is to first control the rotational behavior due to its independence and then control the translational behavior.

Controlling vehicle attitude requires sensors to measure vehicle orientation, actuators to apply the torques needed to re-orient the vehicle to a desired attitude, and algorithms to command the actuators based on (1) sensor measurements of the current attitude and (2) specification of a desired attitude. Once the attitude control is designed and optimized, it can be integrated with the trajectory controller.

The block diagram for attitude controller is as shown below:

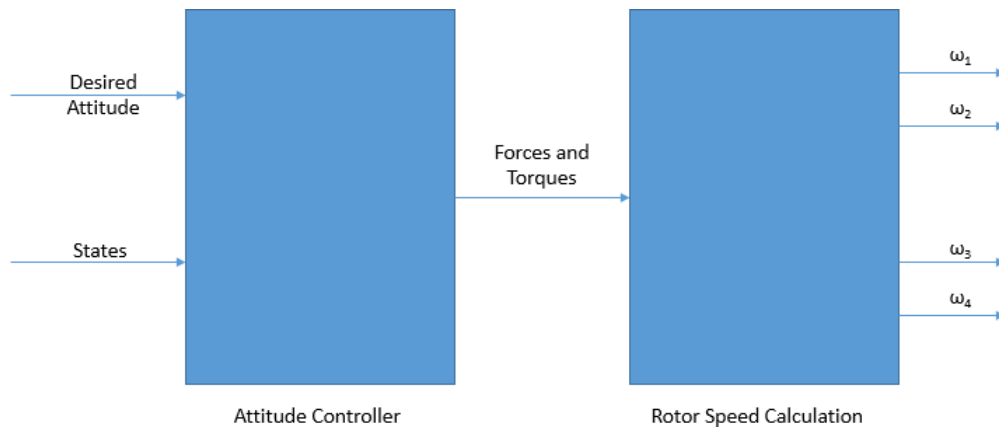


Figure 5: Block diagram of Attitude Controller

A lot of different methods have been studied to achieve autonomous flight, from which three methods (both linear and nonlinear) are discussed below. A comparative study is carried out to identify the most optimal controller for attitude stabilization.

#### 3.3.1 PID controller

Of all the controllers, a PID Controller is the easiest to implement. The general form of PID controller is

$$e(t) = x_d(t) - x(t) \quad (3.32 \text{ a})$$

$$u(t) = K_P e(t) + K_I \int e(\tau) d\tau + K_D \frac{d}{dt} e(t) \quad (3.32 \text{ b})$$

Where  $u(t)$  is the control input and  $e(t)$  is the error between desired state  $x_d(t)$  and present state  $x(t)$ , and  $K_P$ ,  $K_I$  and  $K_D$  are the parameters for the proportional, integral and derivative elements of the PID controller. The desired values of attitude are fed from an attitude command block.

The standard PID control technique is applied on the nonlinear system directly, with an individual PID block for each attitude variable to control it independently. This does not require the model to be linearized about the hover condition, and can thus stabilize the quadcopter on the advent of strong perturbations.

A phi controller is basically an attitude controller which is used to control the attitude of the quadcopter about the X axis. Using this controller, the  $\phi$  angle is stabilized. It can also be used to set  $\phi$  to a particular value, which would help in the motion of the quadcopter in Y direction. Similarly, the theta controller and psi controllers are used to stabilize  $\theta$  and  $\psi$  attitudes of the quadcopter. The corresponding torques are calculated using the following equations:

$$m_\phi = I_{xx} ( K_{\phi,D} \dot{e}_\phi(t) + K_{\phi,P} e_\phi(t) + K_{\phi,I} \int e_\phi(t) dt ) \quad (3.33 \text{ a})$$

$$m_\theta = I_{yy} ( K_{\theta,D} \dot{e}_\theta(t) + K_{\theta,P} e_\theta(t) + K_{\theta,I} \int e_\theta(t) dt ) \quad (3.33 \text{ b})$$

$$m_\psi = I_{zz} ( K_{\psi,D} \dot{e}_\psi(t) + K_{\psi,P} e_\psi(t) + K_{\psi,I} \int e_\psi(t) dt ) \quad (3.33 \text{ c})$$

Where  $e_\phi(t) = \phi_d(t) - \phi(t)$ ,  $e_\theta(t) = \theta_d(t) - \theta(t)$  and  $e_\psi(t) = \psi_d(t) - \psi(t)$ .

The Z controller is used to stabilize the altitude of the quadcopter to a desired value. Similar to the angular attitude controllers, this controller also employs a PID to control the altitude. Here, the thrust required is calculated using the following equation:

$$T = m C_\theta C_\phi [g + K_{z,D} \dot{e}_z(t) + K_{z,P} e_z(t) + K_{z,I} \int e_z(t) dt] \quad (3.34)$$

Where  $e_z(t) = z_d(t) - z(t)$ .

Since thrust is calculated in the body frame while  $g$  and other PID terms are in the inertial frame, a rotational matrix is applied to the terms in the inertial frame which is given by  $C_\theta C_\phi$  where  $C$  stands for cos function.

### 3.3.2 Feedback linearization controller

Feedback linearization control is a popular nonlinear control approach, where the nonlinear system is algebraically transformed into (fully or partly) linear ones by cancelling the nonlinearities. Most feedback linearization techniques are based either on input-output linearization or input-state linearization. We have adopted the input-output linearization in our work. Input-Output linearization involves the repeated differentiation of the output variables till the input term appears, the last derivative being the  $r^{\text{th}}$  one. This will help in obtaining a mapping between the transformed inputs and the outputs. We use the concept of dynamic inversion to the system given by [], which yields the inner loop that feedback linearizes the system from the control input to the output. The output variables not considered above is called residual or internal dynamics. Dynamic inversion need not necessarily yield the internal dynamics stable, which will then require another outer stabilizing loop.

Consider a SISO (Single Input Single Output) system with state  $x$ , input  $u$  and output  $y$  whose dynamics are given by

$$\dot{x} = f(x) + g(x)u \quad (3.35)$$

$$y = h(x) \quad (3.36)$$

The derivative of the output  $y$  can be expressed as

$$\dot{y} = \frac{\partial h}{\partial x} [f(x) + g(x)u] \quad (3.37 \text{ a})$$

The derivative of  $h$  along the trajectory of the state  $x$  is known as the Lie Derivate and equation (3.41) can be written in terms of lie derivative as

$$\dot{y} = \frac{\partial h}{\partial x} [f(x) + g(x)u] = L_f h(x) + L_g h(x)u \quad (3.37 \text{ b})$$

The output  $y$  is differentiated continuously until input terms appear in the differential equation. Generally, the  $i^{\text{th}}$  derivative of  $y$  is expressed in terms of lie derivative as

$$y_f^{(i)} = L_f^i h(x) + L_g L_f^{i-1} h(x) u \quad (3.38)$$

The above equation can be linearized through dynamic inversion, choosing  $u$  as

$$u = \frac{1}{L_g L_f^{i-1} h(x)} [-L_f^i h(x) + v] \quad (3.39)$$

This yields the simple output equation

$$y^i = v \quad (3.40)$$

The concepts used for the SISO systems can be extended to MIMO systems. In particular, we consider square systems having the same number of inputs and outputs. Suppose that an input term first appears in the  $r^{\text{th}}$  derivative of the  $i^{\text{th}}$  output. Then, the equation for the  $i^{\text{th}}$  output is expressed as

$$y_i^{(r_i)} = L_f^{r_i} h_i(x) + \sum_{j=1}^m L_{g_j} L_f^{r_i-1} h_i(x) u_j \quad (3.41)$$

The set of differential equations that corresponds to the input – output relations may be expressed in the matrix form as

$$\begin{bmatrix} y_i^{(r_i)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix} = \begin{bmatrix} L_f^{r_1} h_1(x) \\ \vdots \\ L_f^{r_m} h_m(x) \end{bmatrix} + E(x) \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \quad (3.42)$$

Where  $E(x)$  is an  $m \times m$  coefficient matrix of the inputs.

This set of equations can be converted to simple linear equations for the outputs by the following input transformation

$$u = -E^{-1} \begin{bmatrix} L_f^{r_1} h_1(x) \\ \vdots \\ L_f^{r_m} h_m(x) \end{bmatrix} + v \quad (3.43)$$

This yields equations of the form

$$y_i^{(r_i)} = v_i \quad (3.44)$$

Thus, the inversion-based control law has the capability in shaping the output response by simply designing the new controls  $v_i$  to get the desired output.

To implement this controller, we assume the system is decomposed into two sub models –  $M1$  with states  $X_1 = [Z, \Phi, \theta, \Psi, W, P, Q, R]$  and  $M2$  with states  $X_2 = [X, Y, U, V]$ .

Consider the inner loop with the sub model  $M1$  and output variables

$$Y_1 = [Z \ \Phi \ \theta \ \Psi]^T \quad (3.45)$$

The first derivative w.r.t time does not contain input terms, as evident from the following equations. The expressions are obtained from Equations 3.17 and 3.22.

$$\begin{bmatrix} \dot{Z} \\ \dot{\Phi} \\ \dot{\theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & 0 & \cos \phi & -\sin \phi \\ 0 & 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{bmatrix} \begin{bmatrix} W \\ P \\ Q \\ R \end{bmatrix} \quad (3.46)$$

The transformation matrix is denoted by  $MatW$ .

Differentiating this again gives:

$$\ddot{Y}_1 = \begin{bmatrix} \ddot{Z} \\ \ddot{\Phi} \\ \ddot{\theta} \\ \ddot{\Psi} \end{bmatrix} = \frac{d}{dt} (MatW) * \begin{bmatrix} W \\ P \\ Q \\ R \end{bmatrix} + MatW * \begin{bmatrix} \dot{W} \\ \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} \quad (3.47)$$

The expressions of  $W$ ,  $P$ ,  $Q$ , and  $R$  contain input terms are obtained from Equations 3.18 c and 3.21.

From the above equations, we obtain the general form for  $Y_1$

$$\begin{aligned}
 \begin{bmatrix} \dot{W} \\ \dot{P} \\ \dot{Q} \\ \dot{R} \end{bmatrix} &= \begin{bmatrix} -g \\ \left(\frac{I_{XX} - I_{YY}}{I_{ZZ}}\right) QR - \frac{I_R}{I_{XX}} Q\Omega \\ \left(\frac{I_{ZZ} - I_{XX}}{I_{YY}}\right) PR - \frac{I_R}{I_{YY}} P\Omega \\ \left(\frac{I_{XX} - I_{YY}}{I_{ZZ}}\right) PQ \end{bmatrix} \\
 &+ \begin{bmatrix} (\cos \Phi \cos \theta) \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_{XX}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{YY}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{YY}} \end{bmatrix} \begin{bmatrix} T \\ M_\Phi \\ M_\theta \\ M_\Psi \end{bmatrix} \\
 &= \text{MatC} + \text{MatD} * U
 \end{aligned} \tag{3.48}$$

$$\ddot{Y}_1 = A_1(X_1) + B_1(X_1) * U \tag{3.49}$$

$$\text{Where } A_1(X_1) = \text{MatA} = \frac{d}{dt} \text{MatW} * \begin{bmatrix} W \\ P \\ Q \\ R \end{bmatrix} + \text{MatW} * \text{MatC} \quad \text{and}$$

$$B_1(X_1) = \text{MatB} = \text{MatW} * \text{MatD}$$

The relative degree of the system is calculated as eight, whereas number of states of the system is twelve. To ensure the stability of the whole system, the remaining internal dynamics must be stabilized. But if we consider the sub model that has eight states, the sub model is stabilized.

Based on the general form, the input to the system can be written as:

$$U = \alpha(X_1) + \beta(X_1) * v \tag{3.50}$$

where  $\alpha(\mathbf{X}_1) = -B_1(\mathbf{X}_1)^{-1} * A_1(\mathbf{X}_1)$

and  $\beta(\mathbf{X}_1) = B_1(\mathbf{X}_1)^{-1}$

This on simplification yields  $\ddot{Y}_1 = \vartheta$ , which is a linear system.

The new input  $\vartheta = [\vartheta_1 \ \vartheta_2 \ \vartheta_3 \ \vartheta_4]^T$  can be designed using any of the standard linear control techniques.

In the first approach, we used PD controller to design the 4 linear control inputs, where the error term is given by  $e = Z_d - Z$  and so on. The control gains are tuned manually to obtain the desired response.

### 3.3.3 Linear Quadratic Regulator

In the second approach, the control inputs are designed using the Linear Quadratic Regulator (LQR) approach. LQR is an optimal control technique used to determine the control signal which drives the system states to the desired value along with minimizing a cost function. Hence, the control effort in case of LQR is the least.

Consider a dynamic system of the form:

$$\dot{\mathbf{X}} = \mathbf{A} \cdot \mathbf{X} + \mathbf{B} \cdot \mathbf{U} \quad (3.51 \text{ a})$$

$$\mathbf{Y} = \mathbf{C} \cdot \mathbf{X} \quad (3.51 \text{ b})$$

The cost function for this optimal problem is given by:

$$J = \int_{t_0}^{\infty} \{ \mathbf{U}(t)^T \cdot \mathbf{R} \cdot \mathbf{U}(t) + [\mathbf{X}(t) - \mathbf{X}_d(t)]^T \cdot \mathbf{Q} \cdot [\mathbf{X}(t) - \mathbf{X}_d(t)] \} dt \quad (3.52)$$

Where  $\mathbf{R}$  is the cost of actuators and  $\mathbf{Q}$  is the cost of the state.

The control input  $\mathbf{U}$  that minimizes the cost function is a static linear feedback as:

$$\mathbf{U} = -\mathbf{K} \cdot [\mathbf{X}(t) - \mathbf{X}_d(t)] \quad (3.53)$$

The value of  $K$  is obtained by solving the Riccati's algebraic equation, performed by MATLAB using the LQR function:

The decomposition of the dynamic model into sub models  $M1$  and  $M2$  is done in this case too, with the controller being designed for the sub model  $M1$ . The equations in state variable form for the eight states is the same as Equations 3.18 c, 3.21 and 3.22.

The sub model  $M1$  is linearized around the equilibrium point (near hover condition) using the Jacobian approach (  $A_r = \frac{\partial}{\partial x} A(0)$  and  $B_r = B(0)$  ).

The linearized plant dynamics is given by

$$\dot{X}_1 = A_r X_1 + B_r U \quad (3.55 \text{ a})$$

$$Y_1 = C X_1 \quad (3.55 \text{ b})$$

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



$$\text{where } A_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\frac{A_z}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{A_r}{I_{XX}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{A_r}{I_{YY}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{A_r}{I_{ZZ}} \end{bmatrix},$$

$$B_r = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_{XX}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{YY}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{ZZ}} \end{bmatrix} \text{ and}$$

A function called `linriz.m` (Appendix A7) was written in MATLAB to solve the algebraic equation using Riccati's method, where the LQR in-built function was employed.

$$K = LQR(A_r, B_r, Q, R) \quad (3.56)$$

The  $Q$  matrix is then modified manually, the above equation is again solved to find the control gains until the desired response is achieved.

### 3.4 Trajectory Controller

In this controller, the deviation from the desired path is calculated (in the body frame) at every instant and is fed to the succeeding blocks as the desired velocity. Using this, the desired roll and pitch angles are calculated. This is the underlying principle of this controller.

The errors in X and Y positions are transformed from the inertial frame to the body frame.

$$\text{Error in } X \text{ in body frame} = (X_d - X) \cos(\psi) + (Y_d - Y) \sin(\psi) \quad (3.64 \text{ a})$$

$$\text{Error in } Y \text{ in body frame} = (Y_d - Y) \cos(\psi) - (X_d - X) \sin(\psi) \quad (3.64 \text{ b})$$

These errors are taken as the desired velocities in body frame.  
Therefore,

$$U_d = \text{Error in } X \text{ in body frame} \quad (3.65 \text{ a})$$

$$V_d = \text{Error in } Y \text{ in body frame} \quad (3.65 \text{ b})$$

The next step is calculation of desired attitudes in order to feed it to the attitude controller.

$$\theta_d = K_{p,\theta}(U_d - U) - K_{d,\theta}(U) \quad (3.66 \text{ a})$$

$$\Phi_d = -\{K_{p,\Phi}(V_d - V) - K_{d,\Phi}(\dot{V})\} \quad (3.66 \text{ b})$$

The above equations are devoid of  $\dot{V}_d$  and  $\dot{U}_d$  terms. The absence of these terms permits the presence of non-differentiable points in the path function.

## Chapter 4 - Simulation

The dynamic model, controllers and the trajectory planners are implemented in MATLAB/Simulink. Separate simulations are carried out for attitude control comparison, trajectory following to the nearest landing point and trajectory following based on the planner. The complete Simulink model including the path commands, trajectory controller, attitude controller and the quadcopter plant is shown in Appendix A15.

The values for the parameters in the quad plant are given in Table 1.

Parameter	Value	Unit
$g$	9.81	$\text{m/s}^2$
$L$	0.225	m
$m$	0.468	kg
$K$	$2.98 \times 10^{-6}$	
$d$	0.0382	
$B$	$0.114 \times 10^{-6}$	
$I_{XX}$	$4.856 \times 10^{-3}$	$\text{kg m}^2$
$I_{YY}$	$4.856 \times 10^{-3}$	$\text{kg m}^2$
$I_{ZZ}$	$8.801 \times 10^{-3}$	$\text{kg m}^2$
$I_R$	$3.357 \times 10^{-5}$	$\text{kg m}^2$

Table 1: Parameter values for quad plant

The initial conditions given to the quadcopter states for the simulation with feasible landing point is mentioned in Table 2.

State	Value	State	Value
$X$	0 m	$\Phi$	10 rad
$Y$	0 m	$\theta$	12 rad
$Z$	2 m	$\Psi$	10 rad
$U$	0 m/s	$P$	0 rad/s
$V$	0 m/s	$Q$	0 rad/s
$W$	0 m/s	$R$	0 rad/s

Table 2: Initial conditions for trajectory control simulation 1

The initial conditions given to the quadcopter states for the simulation with trajectory planner is mentioned in Table 3.

State	Value	State	Value
$X$	1 m	$\Phi$	0 rad
$Y$	1 m	$\theta$	0 rad
$Z$	2 m	$\psi$	0 rad
$U$	0 m/s	$P$	0 rad/s
$V$	0 m/s	$Q$	0 rad/s
$W$	0 m/s	$R$	0 rad/s

Table 3: Initial conditions for trajectory control simulation 2

## 4.1. Simulation for attitude controller comparison

The three controllers – Feedback linearization with PD controller (FBL+ PD), Feedback linearization with LQR (FBL+LQR), and PID controller are implemented as separate Simulink models and simulated. The Simulink block layout is shown in Appendices A3- A6.

A simulation time of 50s with a variable step size is given, and Ode-45 Dormand-Prince method is used to solve the numerical problem.

### 4.1.1. Attitude commands

The attitude commands are provided from a block, which contains step functions for each of the attitude variables. The step function starts with the initial condition as shown in Table 2, and falls to 0 with a step time of 5s for the 3 angles and rises to 3 with the same step time for height command.

### 4.1.2. Control gains

For the FBL+ PD controller combination, the values of all control gains are determined through manual tuning.

Controller	$K_P$	$K_D$
Roll	3	2
Pitch	3	2
Yaw	2	2
Height	3	2

Table 4: Gain values for attitude controller 1

For LQR, the values of all control gains are determined through inbuilt LQR Matlab function, as shown in linriz.m (Appendix A7).

Controller	$K_1$	$K_2$
Roll	10	0.1703
Pitch	10	0.1703
Yaw	10	.0267
Height	10	2.8195

Table 5: Gain values for attitude controller 2

For the PID controller, the values of all 3 gains are determined through manual tuning.

Controller	$K_P$	$K_I$	$K_D$
Roll	6	1.5	1.75
Pitch	5	3	3
Yaw	6	1.5	1.75
Height	15	10	10

Table 6: Gain values for attitude controller 3

## 4.2. Simulation for trajectory following – feasible landing point

### 4.2.1. Path commands

The path commands are provided by two blocks, one that guides the quadcopter to the final goal location and the other that guides it to the nearest landing point determined through the image processing module. The quadcopter's mission is to follow the specified trajectory and stop at its destination with the desired attitude values. A switch is used to change the source of path commands, which is to be activated by the failure detection module. Here, a clock input is given and switching at the end of 50s is performed assuming that the failure occurs then.

For the initial path commands,  $X$  and  $Y$  are fed as a ramp input such that the final goal having coordinates (100,200) is reached within 100s.  $Z$  and  $\Psi$  are given as constants 2m and 0 rad respectively.

For the alternative path commands block, the current state is taken as input. The image processing module returns back the coordinates of the nearest landing location. Based on these two points, a ramp is created till the landing point for both  $X$  and  $Y$ . Once the actual  $X$  or  $Y$  reaches within 0.005% of the desired value, the ramp is replaced by a step with the desired value as magnitude.

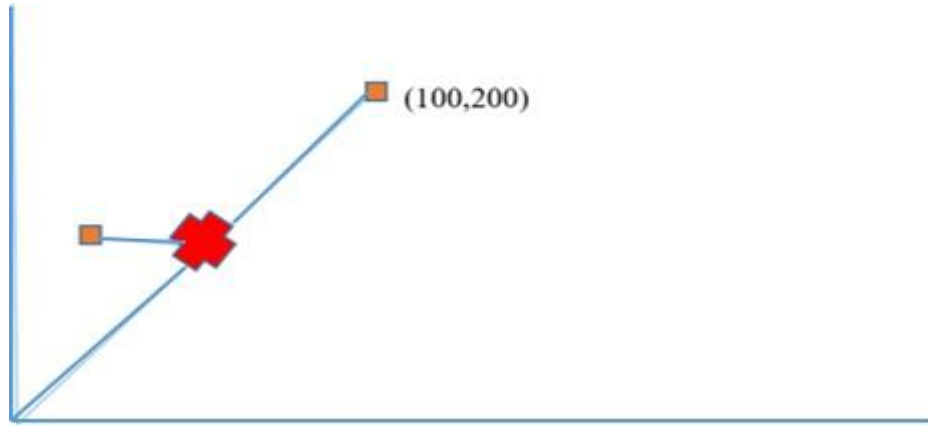


Figure 6: Desired path with switching

#### 4.2.2. Control gains

For the position PD controller, the values for proportional and derivative gains are determined through manual tuning till the desired performance is achieved.

Controller	$K_P$	$K_D$
$\Phi$ command	0.5	0.4
$\Theta$ command	0.36	0.45

Table 7: Gain values for trajectory controller

### 4.3. Simulation for trajectory following – using a trajectory planner

#### 4.3.1. Path commands

The pathcr.m file, which performs the trajectory planning, is first run to obtain the path command data. While the desired  $X$  and  $Y$  commands are obtained as a time series data, the  $Z$  and  $\Psi$  values

are given as a constant function. Compared to the previous simulation, the quadcopter is made to follow a more complex trajectory.

#### 4.3.2. Control gains

For the position PD controller, the values for proportional and derivative gains are determined through manual tuning till the desired performance is achieved.

Controller	$K_P$	$K_D$
$\Phi$ command	0.5	0.4
$\Theta$ command	0.36	0.45

Table 8: Gain values for trajectory controller

#### 4.3.3. VR Simulation

The Simulink 3D Animation package provides apps for linking Simulink models and MATLAB algorithms to 3D graphics objects. This package can be used to visualize and verify dynamic system behavior in a virtual reality environment. Objects are represented in the Virtual Reality Modeling Language (VRML), a standard 3D modeling language. A 3D world can be animated by changing position, rotation, scale, and other object properties during desktop or real-time simulation.

A VR simulation model of the quadcopter was created using 3D World Editor in MATLAB. This model could be rotated or translated about any axes. It was then implemented into the various SIMULINK models of the quadcopter using a VR Sink block. The phi, theta and psi states of the quadcopter were connected to the rotation control of the VR model and the X, Y and Z states were connected to the translation control. Once the SIMULINK model is run, the VR model achieves the corresponding motion in a given terrain which can be seen through a VR simulator.

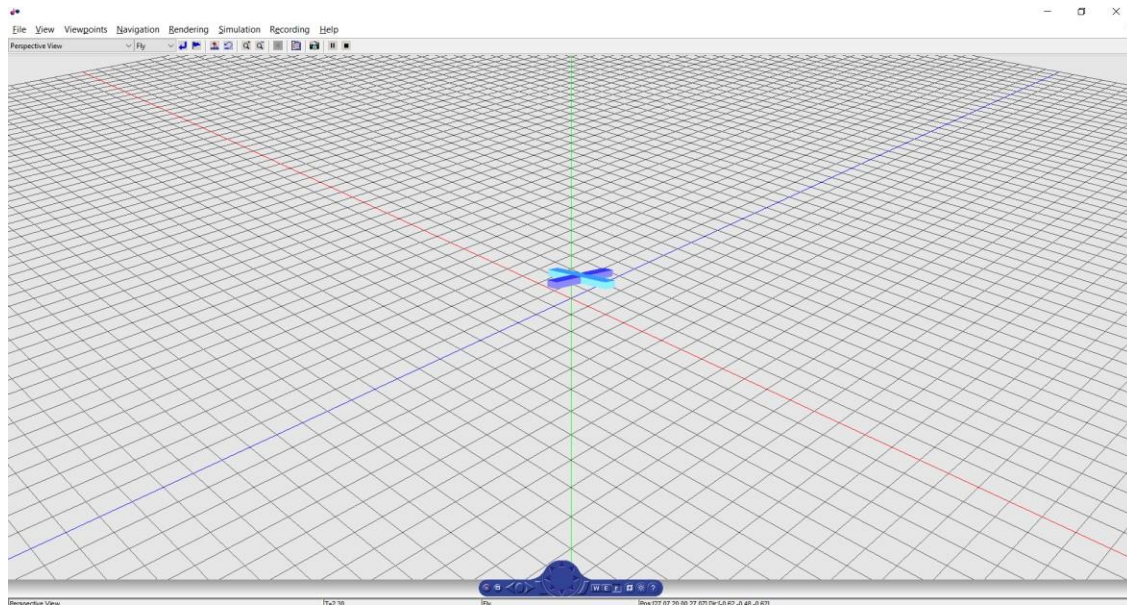


Figure 7: A VR model of quadcopter during simulation (viewpoint 1)

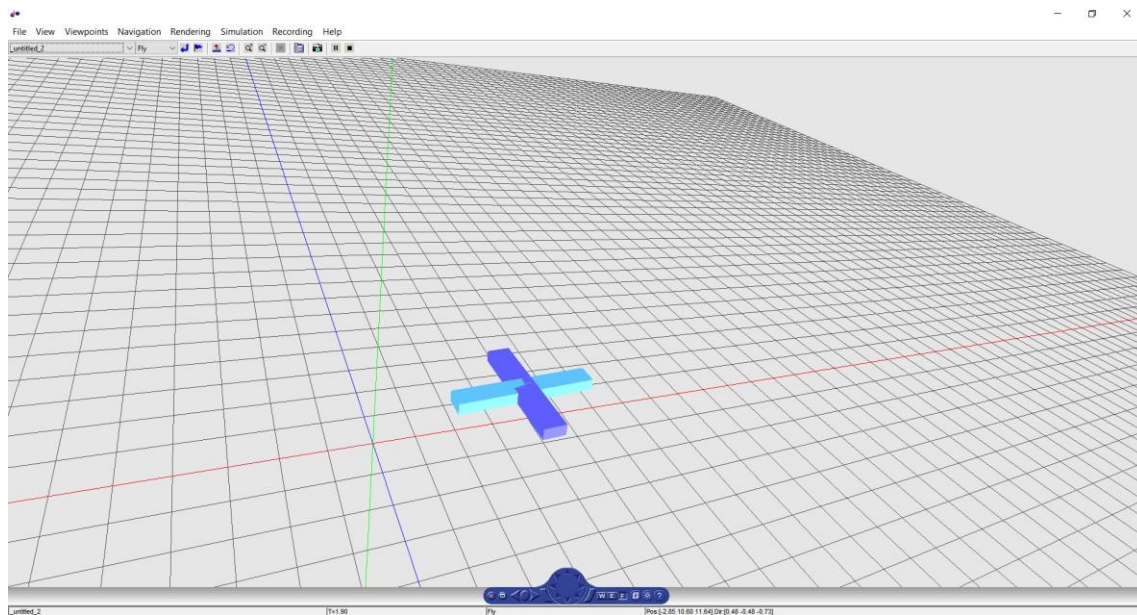


Figure 8: A VR model of quadcopter during simulation (viewpoint 2)



## Chapter 5 - Results and Discussions

### 5.1 Attitude controller comparison

To compare the three different controllers used for the attitude control, a step input is provided as the desired value for each of the attitude variables. The comparison is done based on parameters like rise/fall time and percentage overshoot/undershoot.

Looking at the step response for  $\Phi$  as seen in figure 9, it is evident that LQR controller shows the best performance as the fall time is least and there is no significant undershoot. The combination of FBL and PD controller shows a comparatively slower response with some undershoot. PID controller shows comparatively poor performance as the fall is very gradual and it takes a long time to settle, though with no oscillations. All the three controllers reach the steady state value with no oscillations, and so settling time is not considered here.

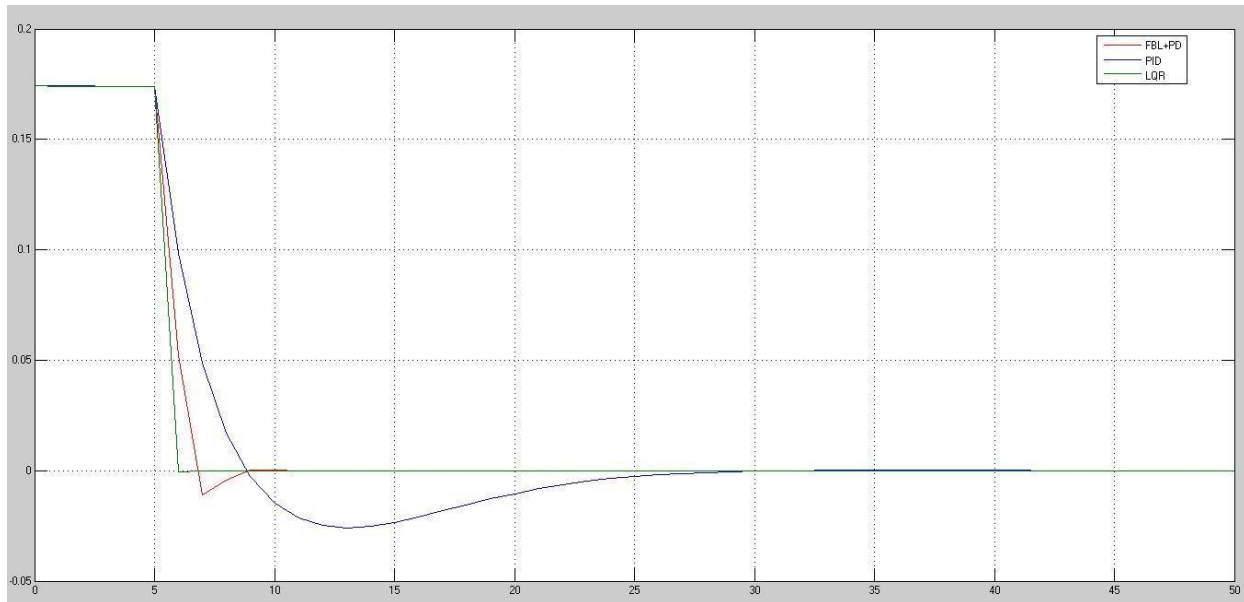


Figure 9: Step responses of controllers –  $\Phi$

Table 9 gives a quantitative comparison between the parameters mentioned above, which clearly shows the superior performance of LQR.

$\Phi(t)$	FBL + PD	PID	LQR
Fall time (s)	1.076	2.792	0.059
Undershoot (m) %	5.851	14.368	0.556

Table 9: Characteristic parameters to a step input for  $\Phi$

Consider the step response for  $\theta$ , where LQR again shows the better performance. The combination of FBL and PD controller has almost similar fall time, but shows some undershoot. The PID controller shows a large undershoot and some oscillations leading to a poor response. Based on settling time, both LQR and FBL+PD combination show similar performance but PID lags far behind.

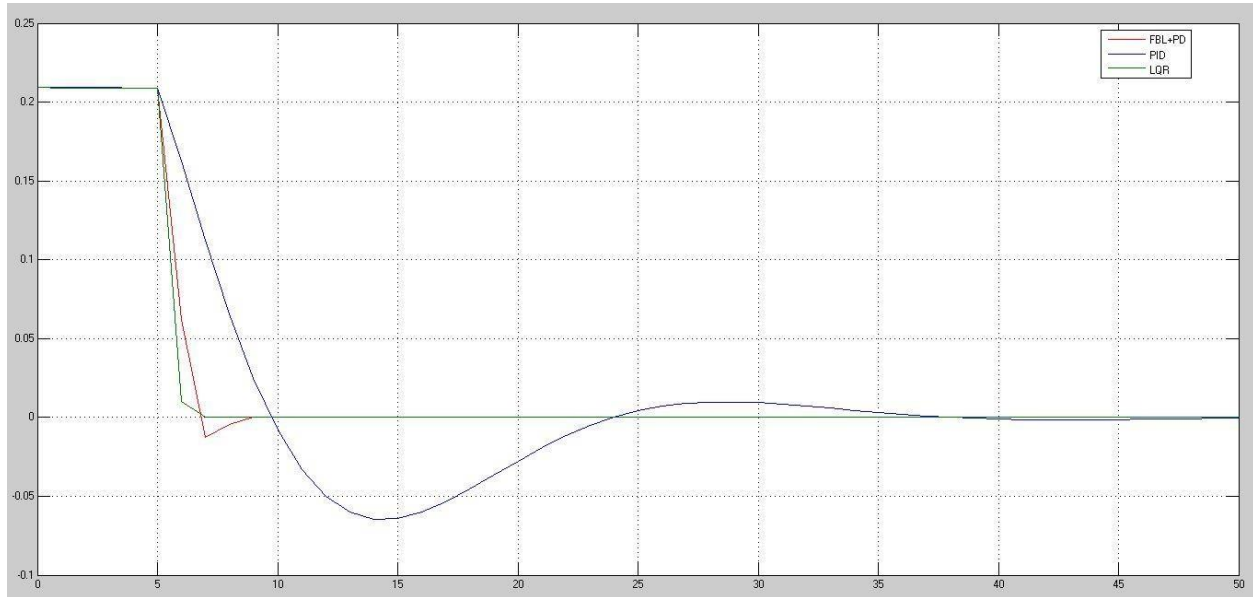


Figure 10: Step responses of controllers –  $\theta$

Table 10 gives a clearer picture about the  $\theta$  response of the three controllers for a step input.

$\theta(t)$	FBL + PD	PID	LQR
Fall time (s)	1.076	3.612	0.074
Undershoot (m) %	5.851	30.921	0.505

Table 10: Characteristic parameters to a step input for  $\theta$

Figure 11 shows the step response for  $\Psi$  for the three controllers. Here, the combination of FBL and PD controller show the best performance with very small undershoot and a decent fall time. LQR shows the least fall time, but has a large undershoot and rapidly varying response before settling. PID also shows a very undershoot and the slowest response among the three controllers.

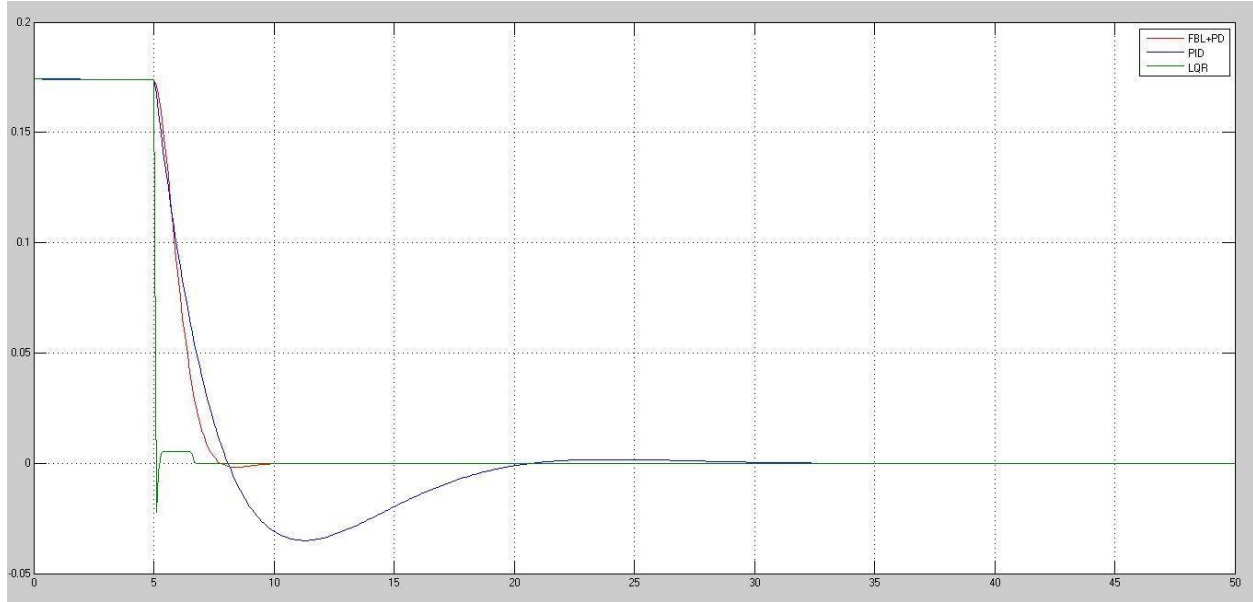


Figure 11: Step responses of controllers –  $\Psi$

Table 11 contains the values of the characteristic parameters used for comparison, and clearly shows that the combination of FBL and PD controller has the best performance.

$\Psi(t)$	FBL + PD	PID	LQR
Fall time (s)	1.561	2.32	0.053
Undershoot (m) %	1.531	19.88	15.698

Table 11: Characteristic parameters to a step input for  $\Psi$

For Z control, since a positive step was given as input the comparison is made on the basis of rise time and percentage of overshoot. Figure 12 shows the response of the three controllers, from which it is evident that the combination of FBL and PD controller has the best performance. It shows the least overshoot and remains steady before the step input is given. The PID controller has a faster rise, but higher overshoot due to the step input and a deviation before the step is provided. LQR shows a very poor performance in this case, as it saturates below the desired height. It also shows deviations before the step is given.

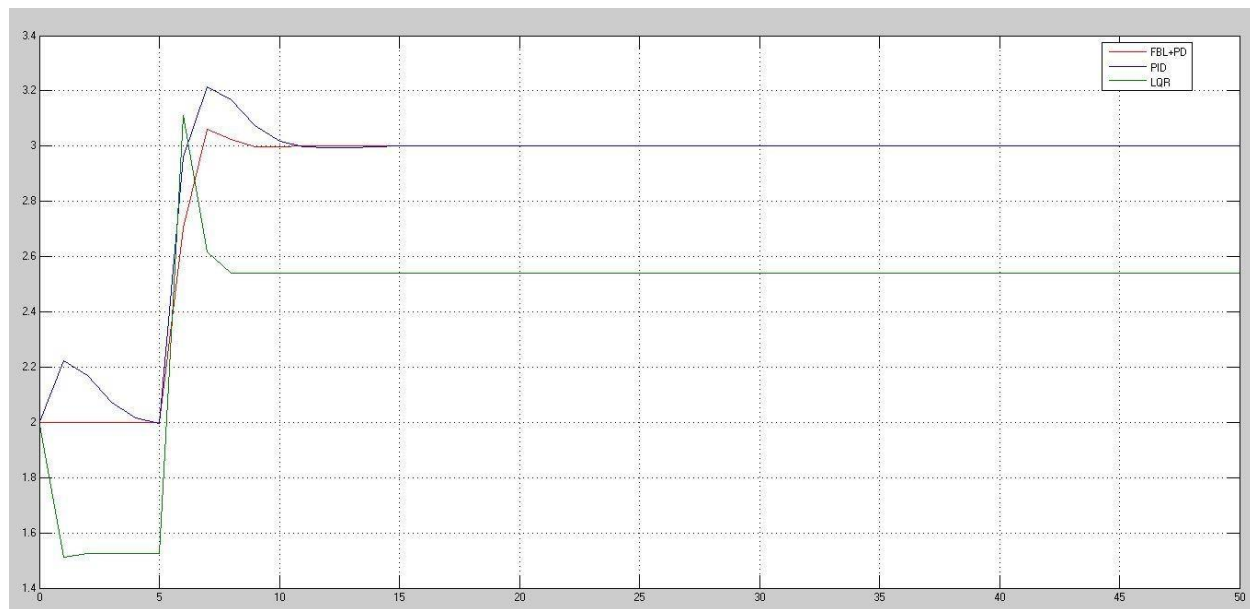


Figure 12: Step responses of controllers –Z

From table 12, it is clear that the combination of FBL and PD controller has the best performance.

$Z(t)$	FBL + PD	PID	LQR
Rise time (s)	1.07	0.788	0.33
Overshoot (m) %	5.851	21.341	58.871

Table 12: Characteristic parameters to a step input for Z

Finally, a comparison can also be made based on the computational time. All the 3 controllers were simulated for 50s in MATLAB. As seen in the table below, the computational time is least for the combination of FBL and PD controller.

Controller	Simulation time (sec)
FBL + PD	2.98
PID	4.08
LQR	5.10

Table 13: Computational time of controllers

From the comparative study presented above, it is very evident that the combination of Feedback Linearization and PD controller shows the best performance. Hence, a trajectory controller is integrated with this to achieve the desired trajectory tracking.

## 5.2 Trajectory planner

### 5.2.1 Selecting the most feasible landing point

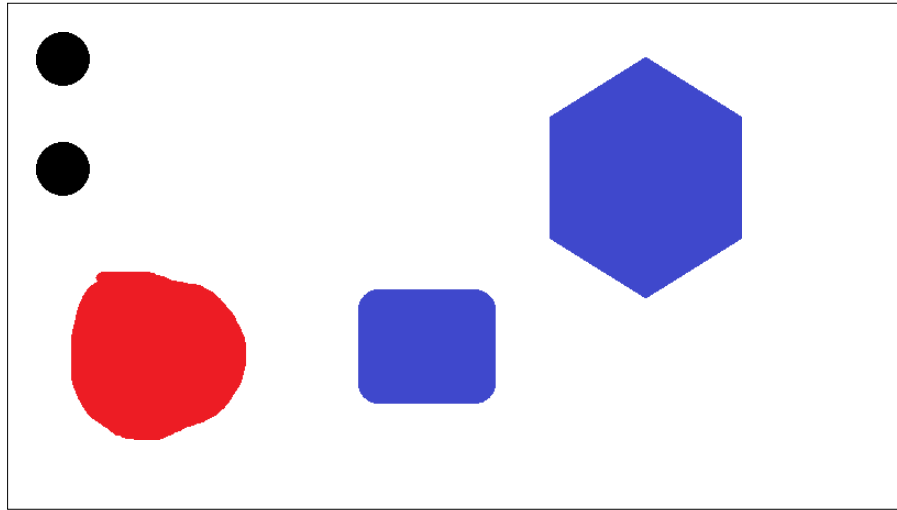


Figure 13: Image containing different shapes in different colors

The input image consists of various shapes in various colors. The final image consists of only the black circles. The centroids of these circles are the feasible landing points.



Figure 14: Black circle given as output

### 5.2.2 Trajectory Planning algorithm



Figure 15 : Maze input to the trajectory planner

The input maze consists of walls which are no fly zones shown in Figure 15. The Figure 16 shows the path generated by the trajectory planner.



Figure 16 : Path generated by the trajectory planner

## 5.3 Trajectory tracking

### 5.3.1 Trajectory tracking – Most feasible landing point

A comparison can be made between the desired trajectory and the actual trajectory using Figure 17. It can be inferred that the quadcopter is able to follow the trajectory with reasonable accuracy throughout. At the point of switching, the inertia of the quadcopter prevents it from making a sharp change in the trajectory. So it consumes some time before again following the desired trajectory. When it approaches the end point, the change in input from ramp to step occurs, but the quadcopter due to its inertia oscillates about that point before finally resting there. The final location of the quadcopter is exactly in the specified landing point, which can be concluded from Figure 18 where  $X$  and  $Y$  comparisons are plotted.

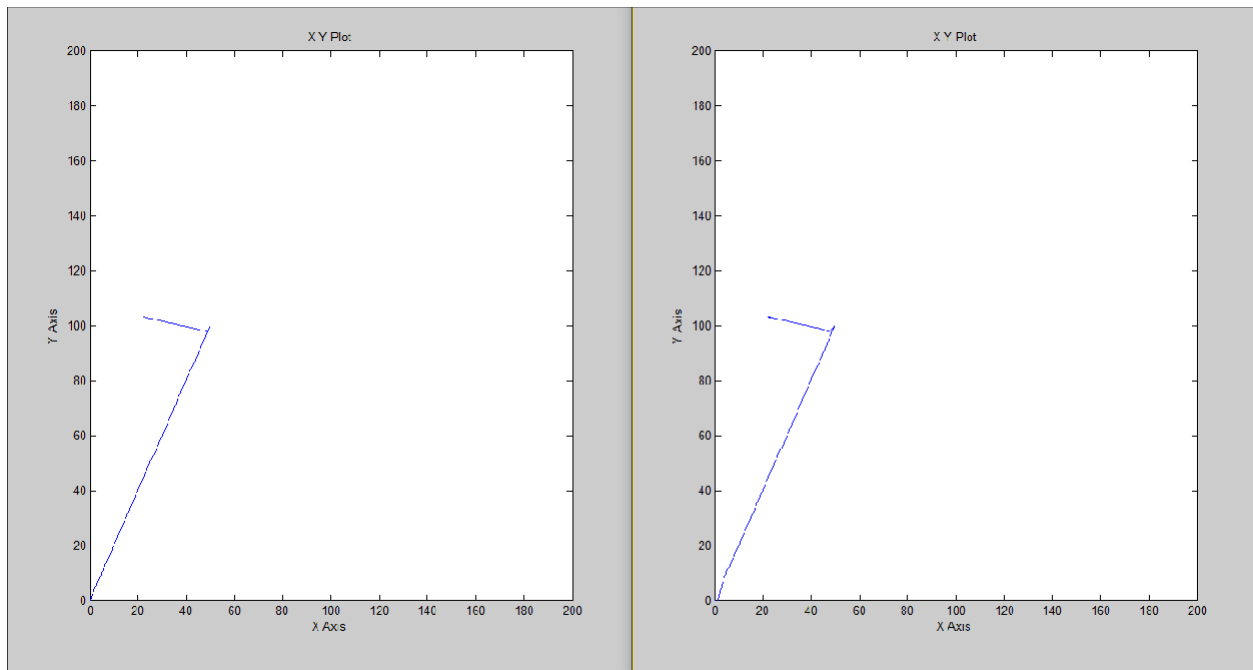


Figure 17: Actual path followed

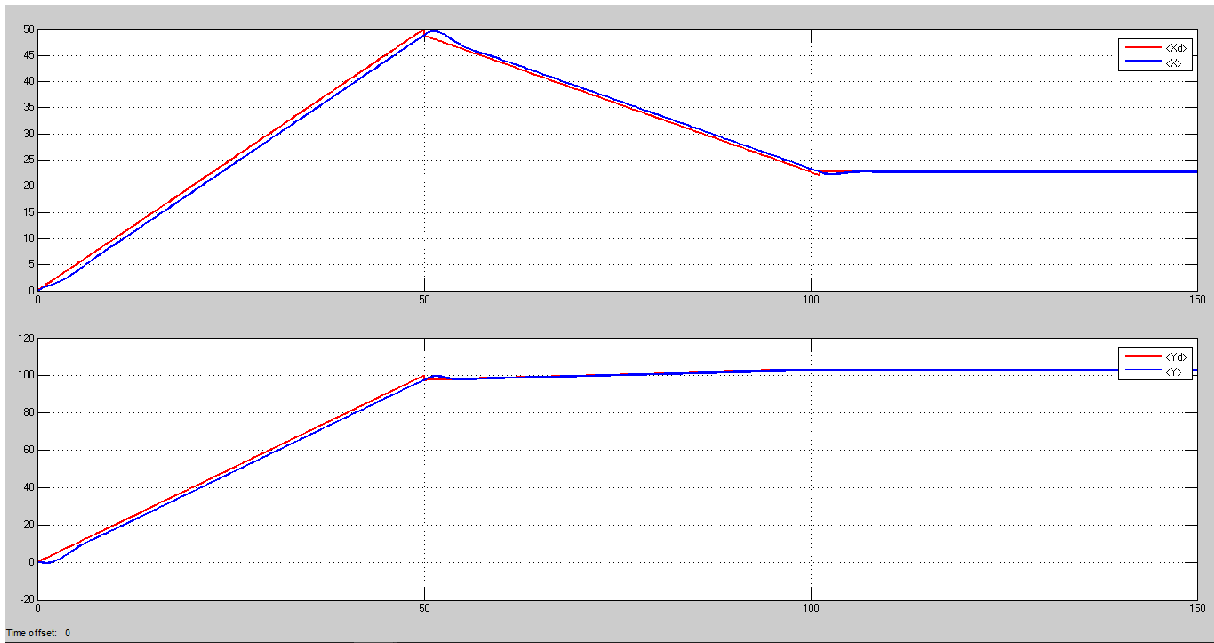


Figure 18: Coordinate wise comparison of desired and actual paths

The plots of attitude variables vs time for the above simulation is given in Figure 19. There are three regions of interest – the starting point when a path command is given, during switching and finally after reaching the landing point. The attitude variables show changes only in these three regions.

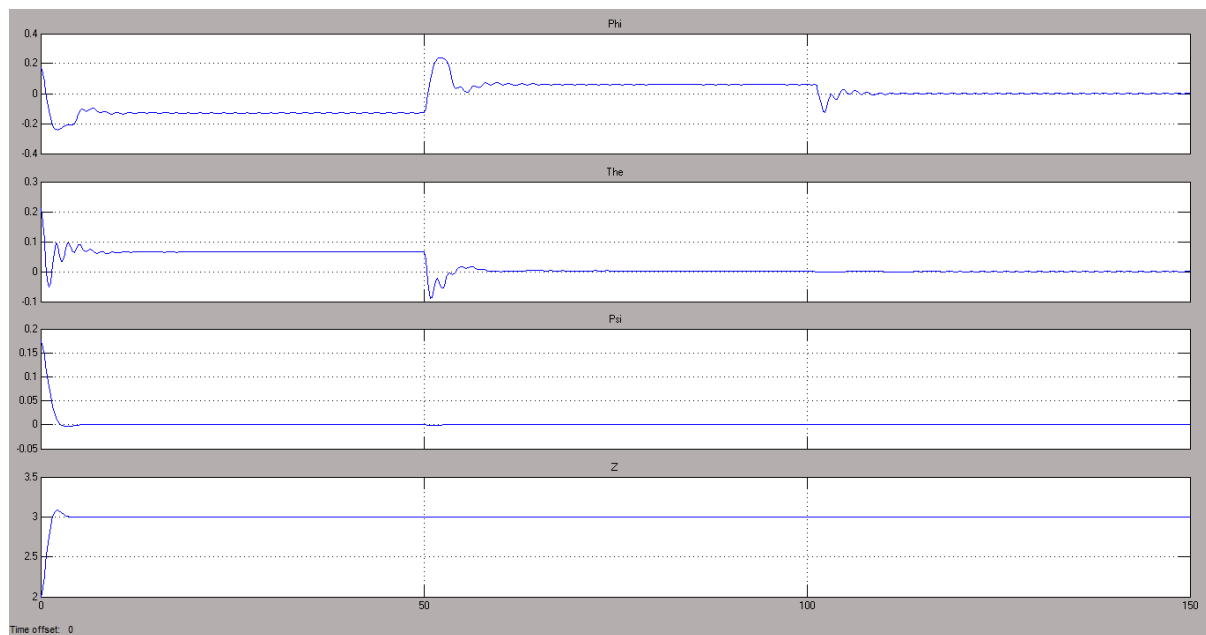


Figure 19: Plots of attitude variables vs Time



### 5.3.2. Trajectory tracking – traversing the maze

In this simulation, the quadcopter is traversing a complex maze. The path planning is done based on the image of surroundings. The image is processed to identify obstacles and safe flying zones. The Dijkstra's algorithm is employed to determine the shortest path from the starting point to destination. Then, trajectory tracking is used to closely follow this path.

The shortest path from the left top corner of the image to the right bottom corner of the image is generated using Dijkstra's Algorithm. The path is efficient and is devoid of any obstacles. This path can be seen in Figure 20.

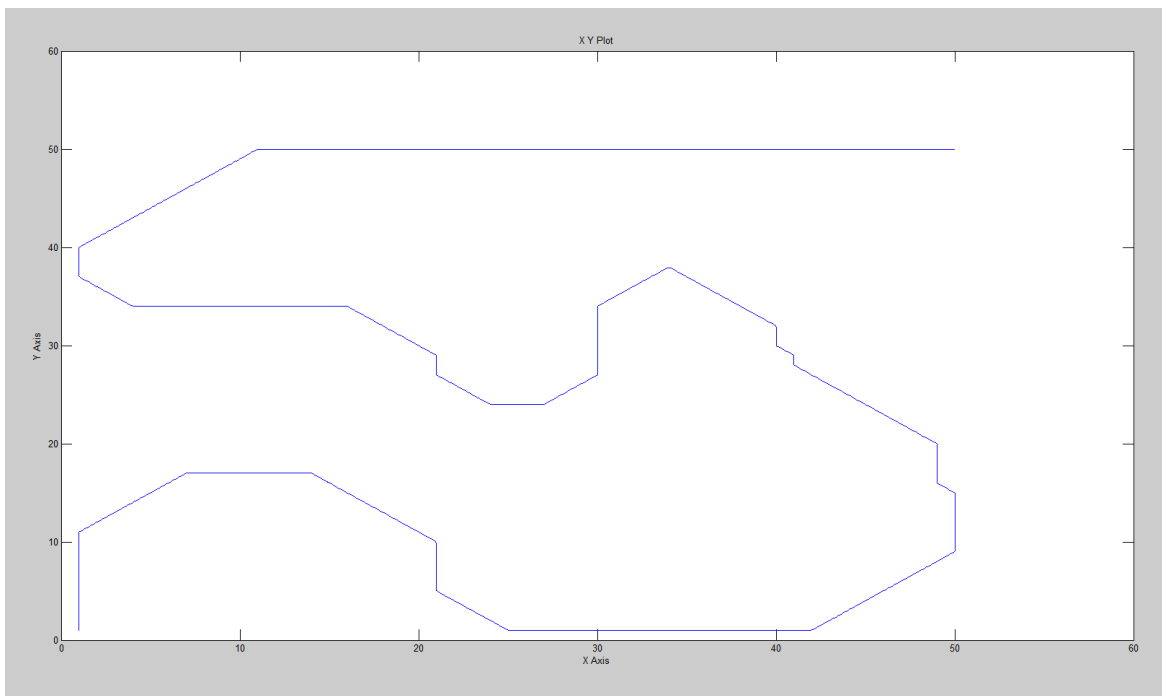


Figure 20: Desired path generated by trajectory planner

This trajectory is fed in the form of a time series data to the controller. The trajectory is followed with reasonable accuracy. It is visible in Figure 21 that all the sharp turns in the desired trajectory are transformed to smooth curves in the actual path due to the inertia of the quadcopter.

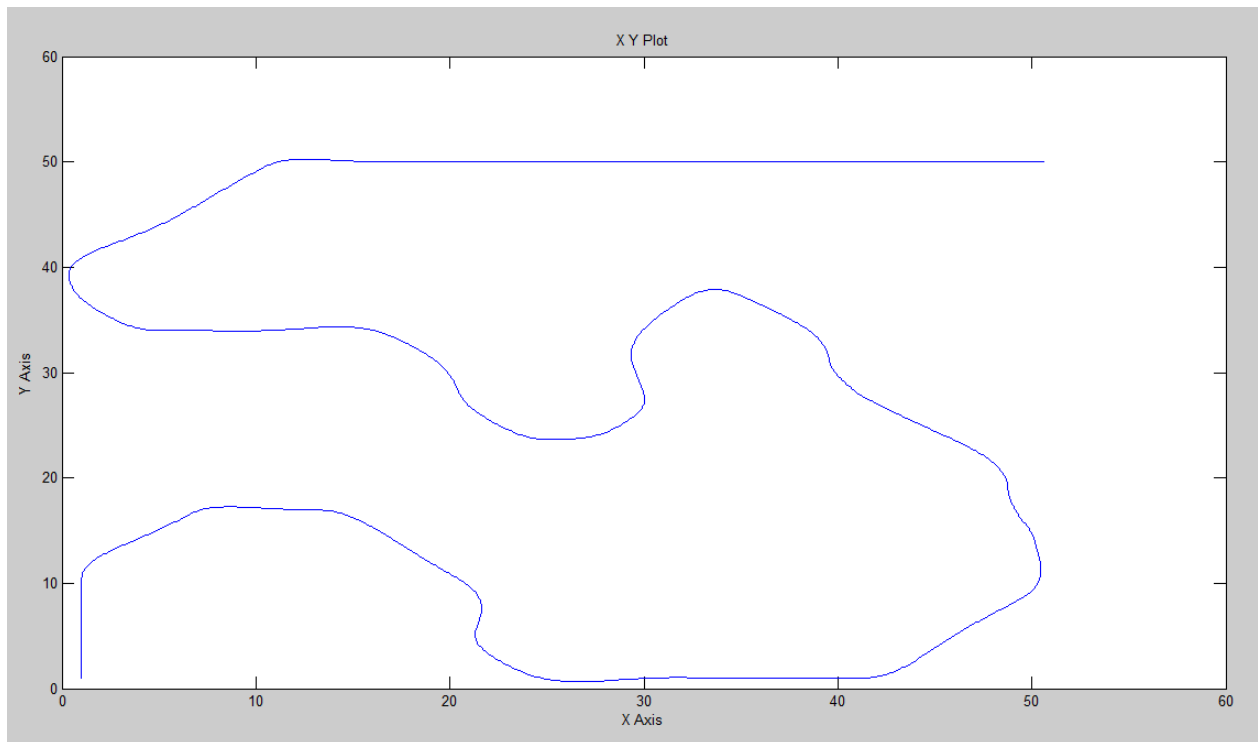


Figure 21: Actual path followed by quadcopter

The trajectory controller is successful in guiding the quadcopter to traverse such a complex path, which is clearly seen in Figure 22 through a coordinate wise comparison. Although there is a slight lag between the desired and actual paths due to the slow response of the trajectory controller, it finally ceases at the end when the destination is reached.

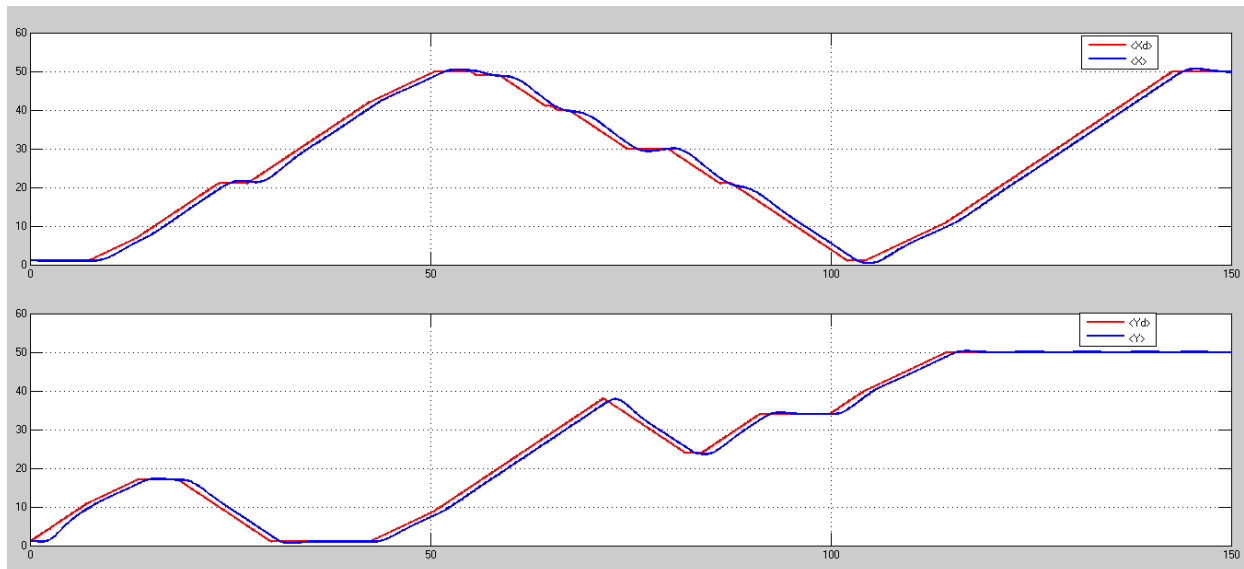


Figure 22: Coordinate wise comparison of desired path and actual path

The attitude plots, especially the roll and pitch angles show changes throughout the course of the quadcopter motion because of the complexity in path. Theta and x are correlated whereas Phi and y are correlated. Z settles at 3m soon after the start of the simulation.

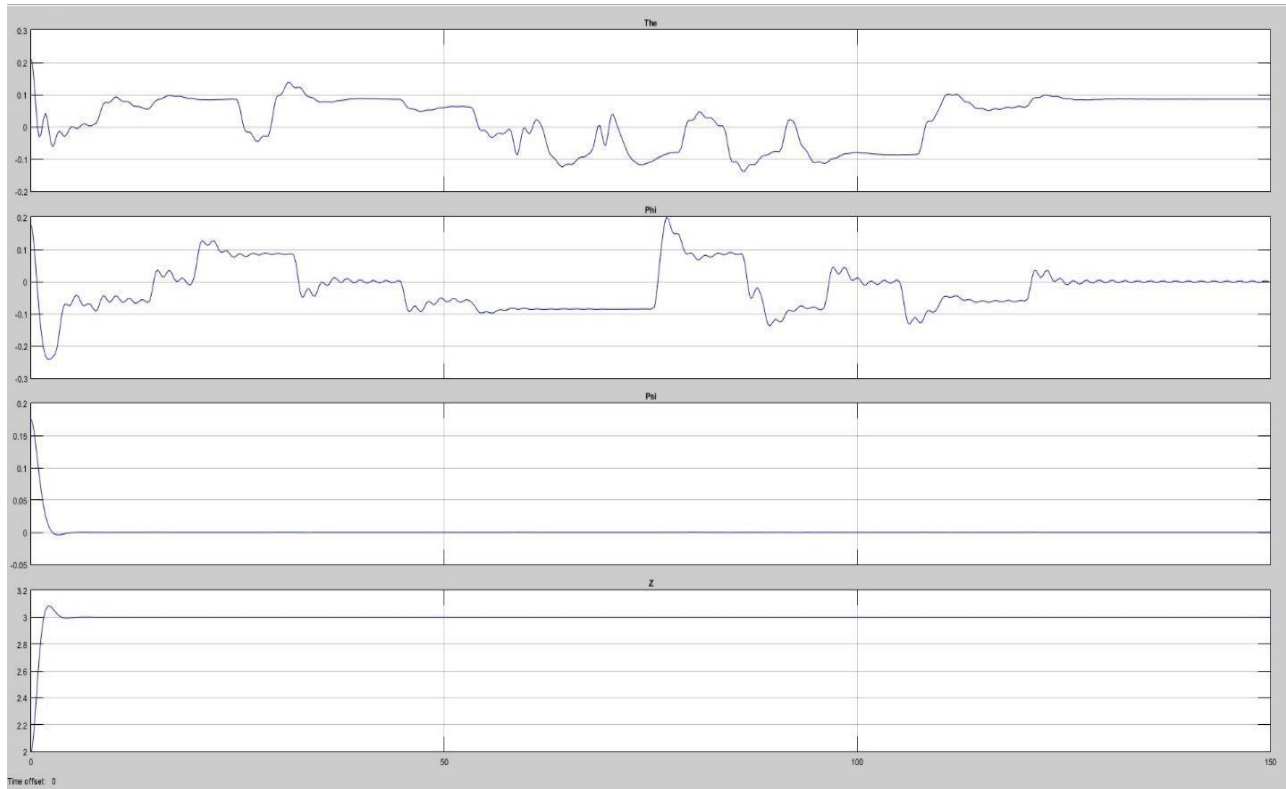


Figure 23: Plots of Attitude variables vs Time

## Chapter 6 - Object detection with Raspberry Pi 4 on the Drone

Code: <https://github.com/AbhishekTyagi404/Object-Detection-on-Raspberry-Pi-with-obstacle-avoiding-rover>

### Introduction: -

Object detection involves detecting instances of objects from a particular class in an image. The goal of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image. Typically, only a small number of instances of the object are present in the image, but there is a very large number of possible locations and scales at which they can occur and that need to somehow be explored [12]. The capability of a computer to locate and identify objects in an image. With the recent advances in deep learning and convolutional neural networks (CNN) [13].

### Background: -

Object detection is the process of detecting and defining objects of a certain known class in an image. Only a few years back, this was seen as a hard problem to solve. Before Krizhevsky presented the CNN AlexNet [14] at the ImageNet Large Scale Visual Recognition Competition in 2012 researchers struggling to find an optimal solution to finding the image classification with a very low error rate. There are many objects detection methods applied by CNN and has been showing the best optimal performance and accuracy. However, much computing power still needed to run the visual task effectively and for real-time purposes, and on a device (Embedded Systems) with limited hardware resources, running this object detection system can be challenging.

### Problem Statement: -

Computer vision is an interdisciplinary field that deals with how a computer can gain a high-level understanding from images or videos. Many problems in computer vision were saturating on their accuracy before a decade. However, with the rise of computer science and deep learning technique, the accuracy improved exponentially. The major problem is image classification and predicting the class. There are many complications of computer vision in image classification in which an image contains one object. While, on the other hand, the tough one is one frame with many objects.

In this case, the input system will be a Raspberry pi [15] camera image, and the output will be the bounding box corresponding to all the objects in the Raspberry pi camera frame, along with the class of objects corresponding to each box.

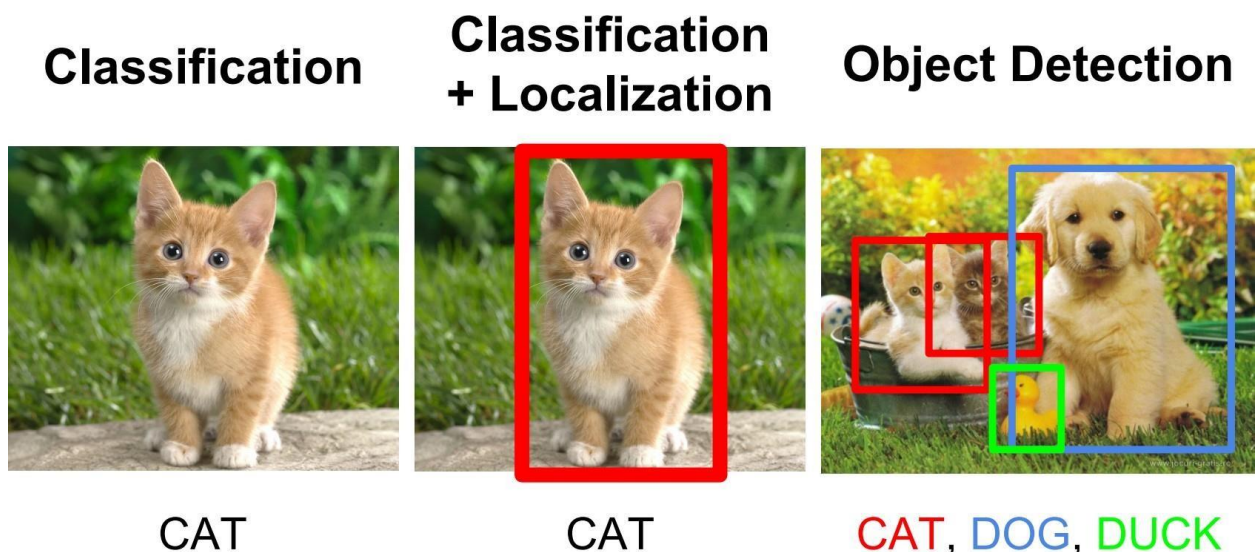
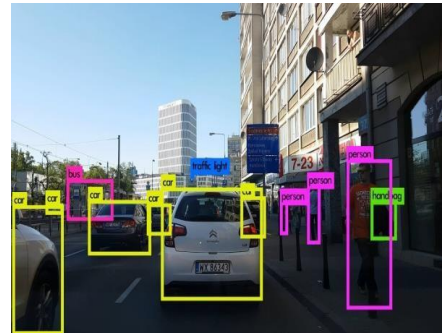
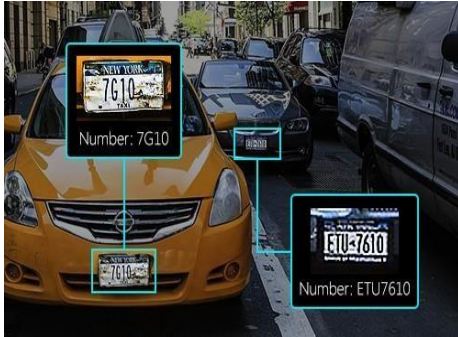


Figure 1: Computer Vision Task

## Applications: -

A well-known application of object detection is human face recognition, that used in almost every smartphone. A more multiclass application can be used in Autonomous vehicles and surveillance systems. These systems are integrated with many tasks such as image annotation, activity recognition and co-segmentation. It is also useful in tracking objects for example tracking a ball during a cricket match, also useful for traffic monitoring and changing the time of red light with the



number of vehicles passing by the road.

(A): Surveillance

(B): Autonomous Vehicle

Figure: 2

## Challenges: -

The major challenge in this project is the real-time working of object detection with the help of the Raspberry pi camera and its hardware support. Also, a problem that challenged us was variable dimensions of the output in different illumination with different backgrounds and which caused due to the variable number of objects that can be presented in the given number of the input frame. And other than that, the major challenge we faced while building our rover is deploying this whole object detection to the rover through Raspberry Pi and performing a whole lot of tasks, also receiving the data in real-time and analyzing that simultaneously running the rover program.

## Related Works: -

Huang et. al. wrote a paper about speed and accuracy trade-offs for modern object detectors where they discussed some of the main aspects that influence the speed and accuracy of object detectors [16]. There are several working projects and traditional computer vision techniques are taking place simultaneously as well as some models are working on real-world projects around the globe. However, the lack in the accuracy of deep learning-based techniques, two classes of models is prevalent. First is “Two-stage detection” which will be RCNN, Fast RCNN, Faster RCNN [17] and second is “Unified detection” which is “You only look once” aka Yolo [18] and “Single-shot detection” aka SSD [19].

In a study by Velasco-Montero et al. [20] they investigated the performance of real-time DNN inference on a Raspberry Pi. They took four different types of frameworks and four popular DNN models used for image classifications. The authors demonstrated that it was possible to achieve a real-time interference speed on a Raspberry pi Model B. Object detection was the focus of this paper is a more computationally demanding task.

These concepts have been explained below:

## Bounding Box: -

The bounding box is an imaginary rectangle box, that serves as a reference point or rectangle for every instance of every object that is present in the camera frame or an image. Data annotators draw these rectangles over the object that will be identified by the algorithm. For every single box, four numbers need to be predicted (Center X, Center Y, Width and Height). This can be trained using a distance measure between predicted and ground-truth bounding boxes. The distance measure is Jaccard distance [\[21\]](#) which measures the intersection over the union between the predicted and ground truth boxes.

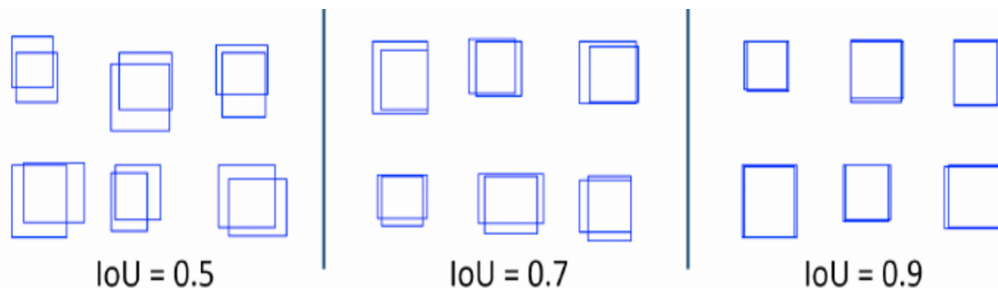


Figure 3: Jaccard Distance

## Approach:

### Classification and Regression: -

The bounding box is predicted using regression and the class within the box is predicted using classification. The overview of the architecture is shown in the figure below:

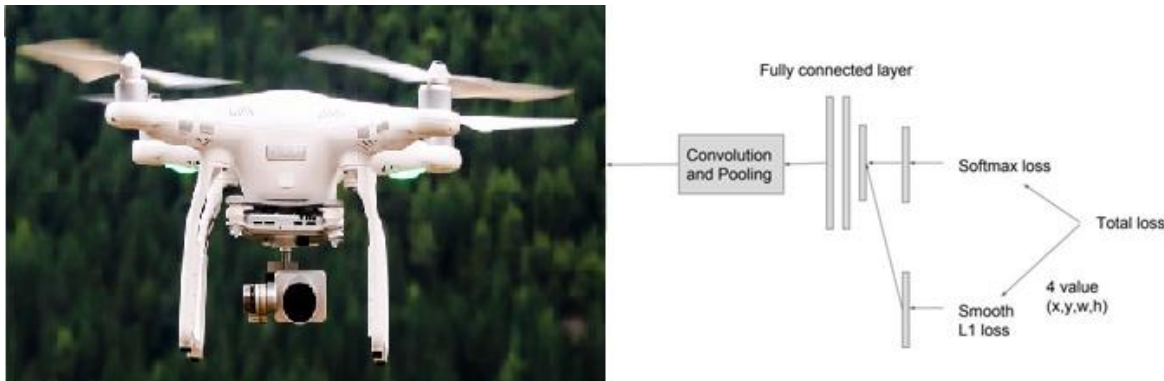
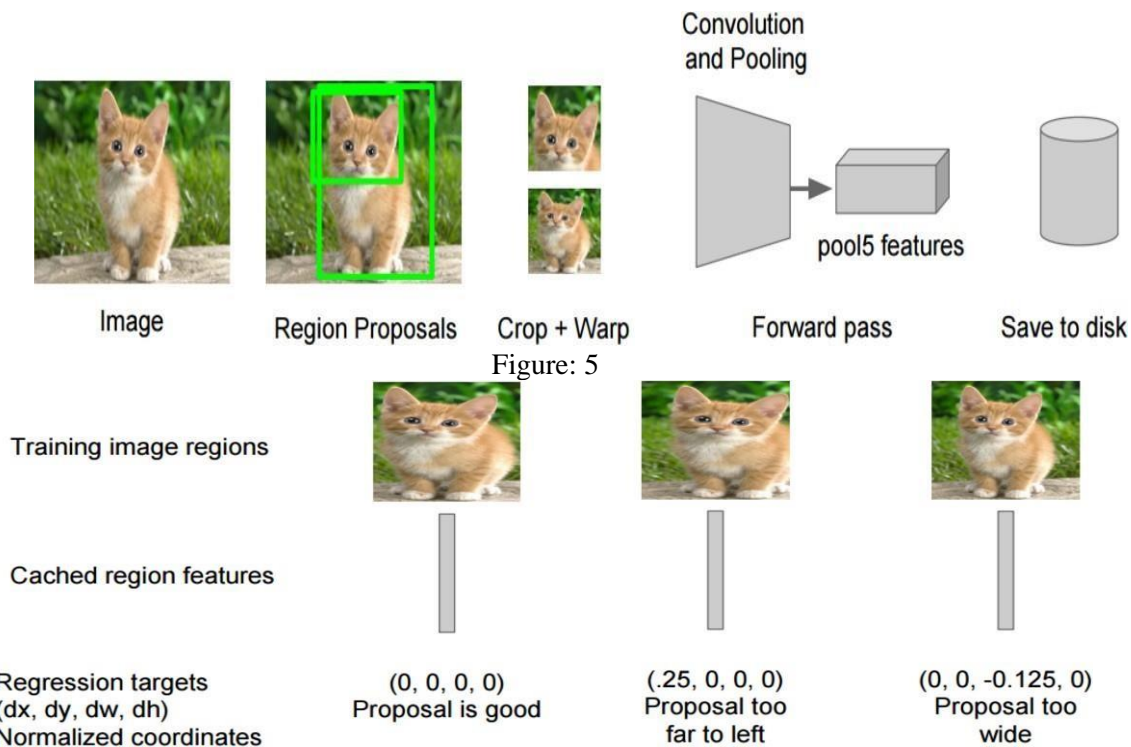


Figure: 4

### Two-Stage Method: -

In this case, the bounding box is generated using the same other computer vision techniques and then resized to desired /fixed output for classification, which acts as a feature extractor. SVM [\[22\]](#) is good for linear as well as non-linear regression problems and work well for practical problems. The algorithm is simple to understand, it separates the data with a hyperplane line distanced with some margin ( $D+$  and  $D-$ ) and turns it into the classes. Here, we shall use SVM (Support Vector Machine) to trained to classify between background and objects (single SVM for each class). Also, a bounding box regressor trained in a way that it can give some correction to the proposal boxes (Offset). The overall approach is shown in the figure.





### Unified Method: -

In this method, we are not producing a pre-defined set of boxes or proposals to look for an object, but we are using convolutional layers of the network, then we run another network over these feature maps to predict the class and bounding box offsets. The idea delineated in the figure below:

- Train convolutional neural network (CNN) with regression and classification objectives.
- Convened activation from later layers to infer classification and location with fully connected layers.
- When training the module, use Jaccard distance to relate ground truth with predictions.
- During interference with many objects in a single frame, use non-maxima suppression to filter multiple boxes around the same objects.

This technique followed by SSD (Single Shot Detection) [23]. Uses many different activations maps on multiple scales for the prediction of bounding boxes and classes. Whereas YOLO (You Only Look Once) [24] use single activation maps for the prediction of both bounding boxes and classes. In this project, we are using SSD for prediction because using multiple scales benefits to achieve higher “mean average precision (mAP)”. It can detect objects of different sizes on the image better than YOLO.

### Comparison

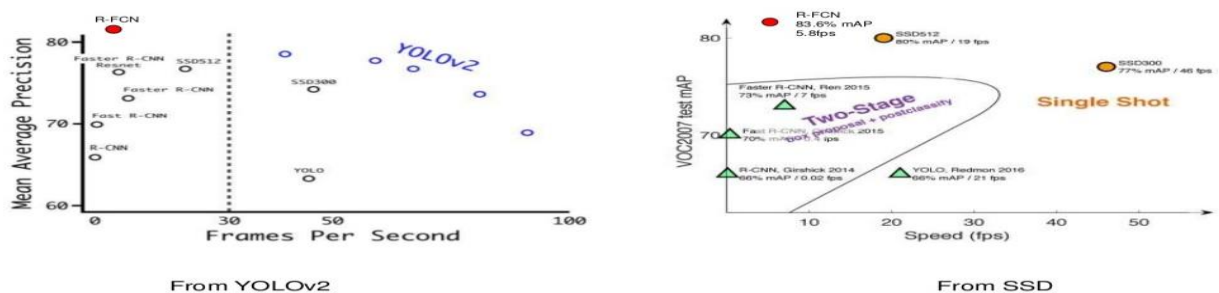


Figure: 7

## Architecture Design:

The network which we are using in this project is SSD (Single Shot Detection). The architecture of SSD is shown below in figure 8.

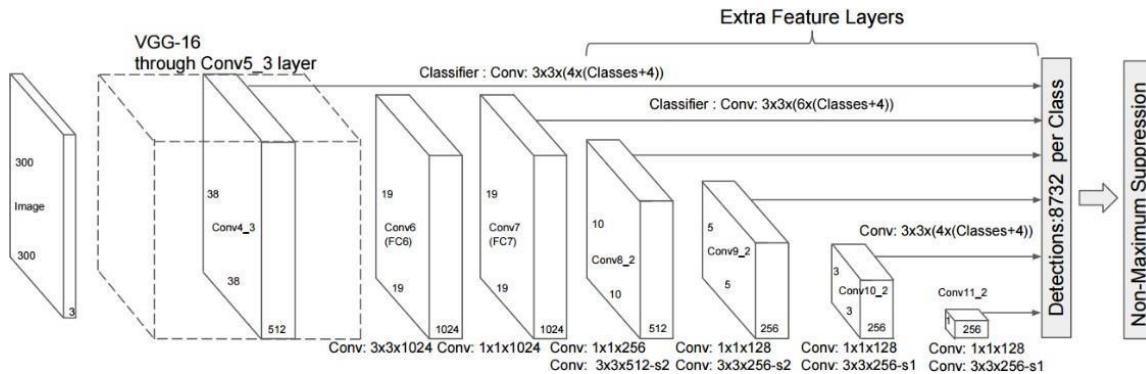


Figure: 8

The SSD starts with a VGG model, the VGG Net is the name of the pre-trained convolutional neural network (CNN) model. The VGG model is a CNN model proposed by K. Simonyan and A. Zisserman jointly [25]. The model can achieve 96.5% of the top 5 test accuracy in ImageNet, which is a dataset of 14 million images belonging to 1000 different classes. After the VGG model, it is converted to a fully convolutional network. Then we attach some extra convolutional layer that helps us to handle bigger objects in the real world. The output at the VGG network is a 38x38 feature map. The added layers produce 19x19, 10x10, 5x5, 3x3 and 1x1 feature maps. These feature maps are useful for predicting various bounding boxes and various scales, meanwhile, the other later layers predict larger objects if needed.

Thus, the overall algorithmic approach is shown in the figure below. Subnetworks passed to the activations that act as a classifier and localizers.

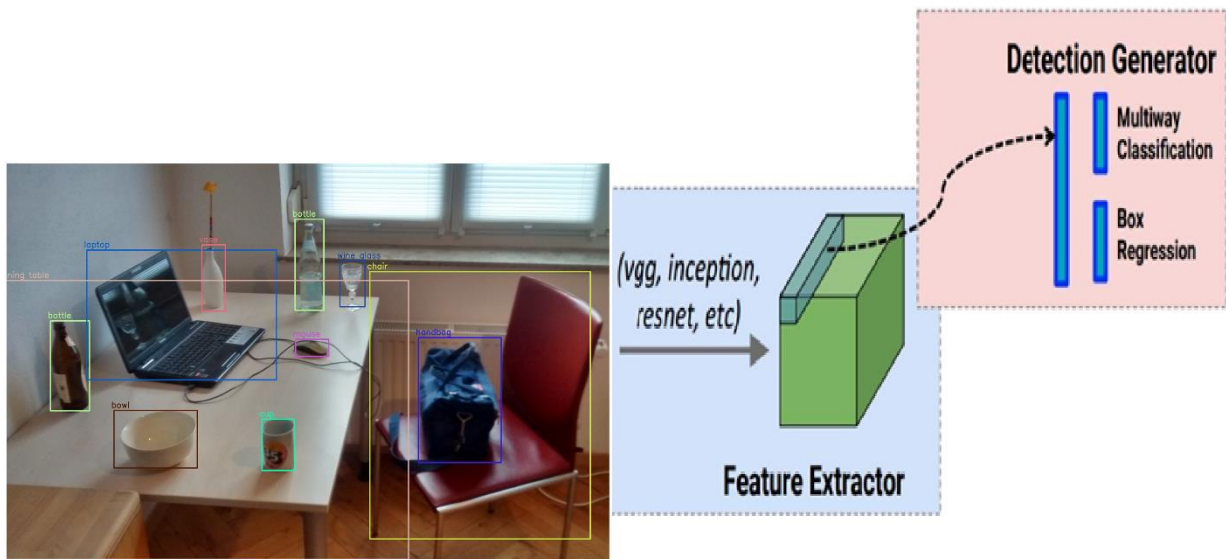


Figure: 9



The collections of boxes veneer on an image at different spatial locations, scales and aspect ratios is known as anchors. These anchors act as reference points on ground truth images as shown in the figure.

## Region based methods - Faster R-CNN

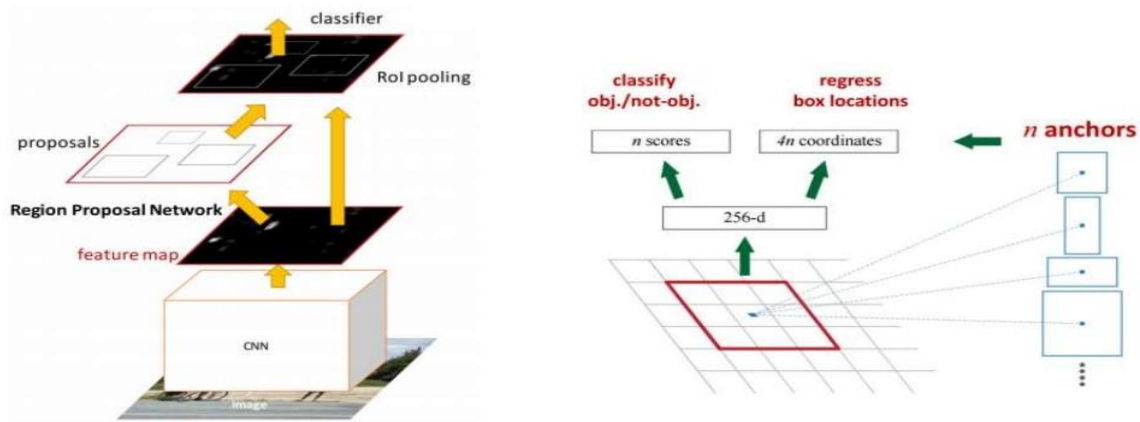


Figure: 10

- Discrete Class
- Continuous offset by which the anchor needs to be shifted to fit the ground truth of the bounding box.

During training, SSD matches ground truth annotations with anchors. Each and every element of the feature map has many anchors associated with it. Any anchor with a 0.5 Jaccard distance or greater than 0.5 is considered a match. Consider a case in the image shown below, here the cat has two anchors overlapped on each other whereas the dog has only one anchor matched. Note that even in this case both matched at different feature maps.

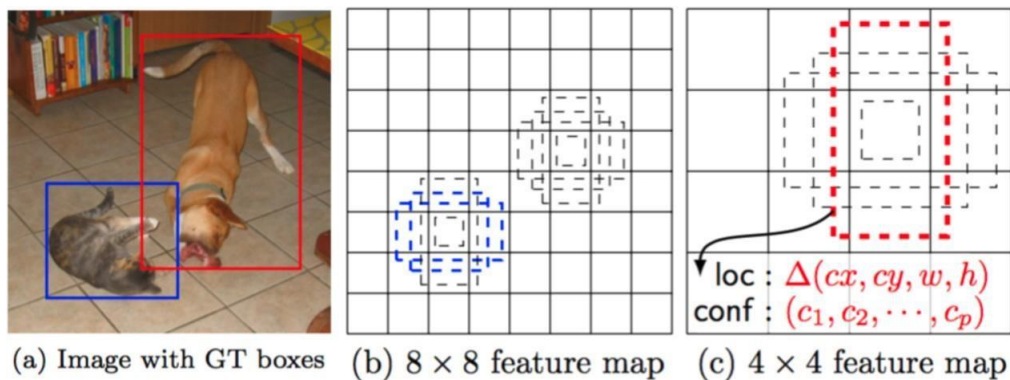


Figure: 11

**Loss Function:** The loss function used is the combination of multi-box classification and regression loss. The classification used is the SoftMax cross-entropy, whereas for regression we use the smooth L1 loss. During the prediction to filter the different and multiple boxes per object we used non-maxima suppression that may be matched as shown in the figure below.

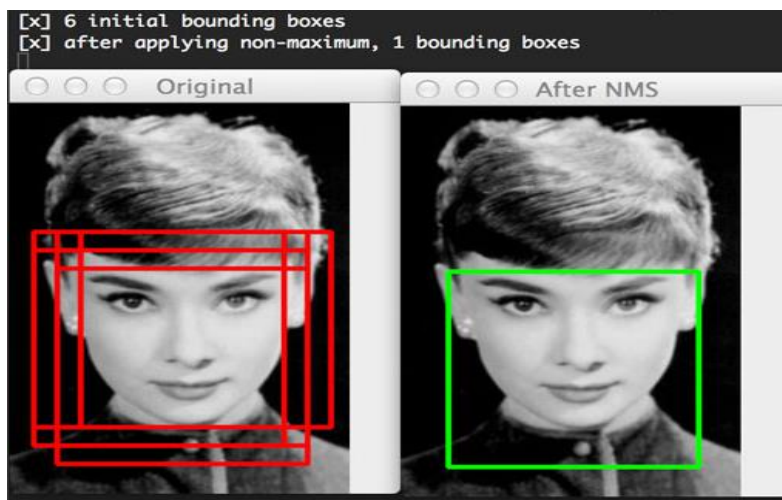
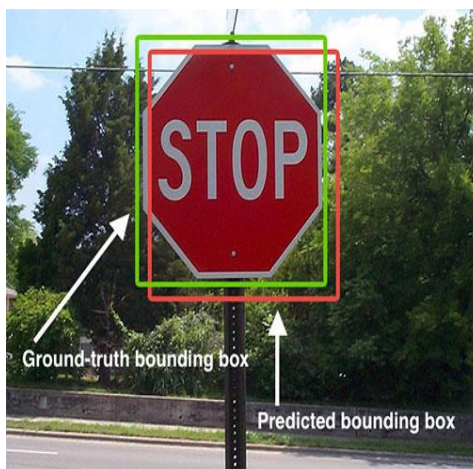


Figure: 12



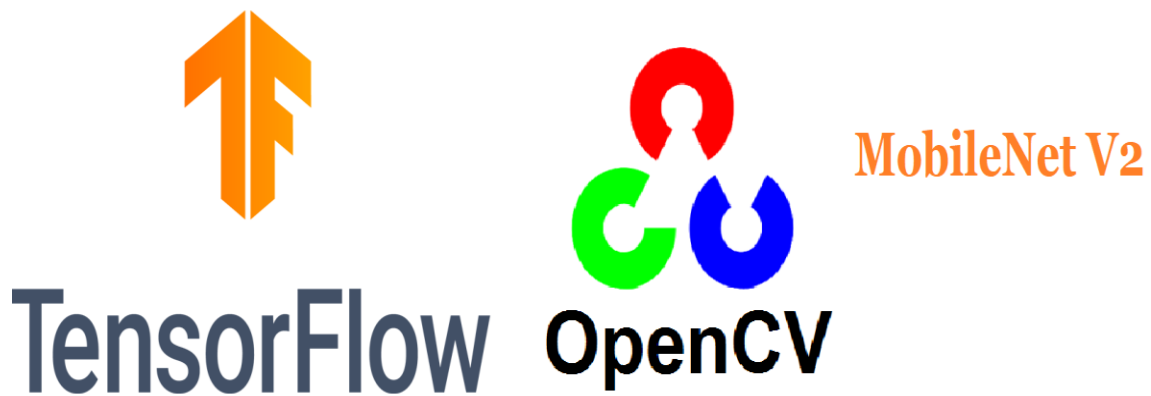
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Figure: 13

## **Experimental Setup:**

### **Dataset:**

For this project, and to do objects detection we are using publicly available SSDLite\_MobileNet\_V2\_COCO model from Tensorflow [\[26\]](#) Detection Model Zoo (The model zoo is Google's collection [\[27\]](#) of pre-trained object detection models that have various speed and accuracy levels). The specific model we are using in Raspberry pi 3 is "SSDLite\_MobileNet\_V2\_COCO\_2018\_05\_09" [\[15\]](#) because Raspberry pi has a weak processor, so we need to use a model that takes less processing power in real-time usage. Though the model will run faster enough and it comes at a tradeoff of having lower accuracy. This model has been trained on the COCO [\[28\]](#) dataset which has 90 different objects categories. Here, we are using the lite version, which consists of 10,000 images with 20 different objects classes and the label objects structure is defined in string\_int\_label\_map.proto file in protobuf [\[29\]](#) format. The TensorFlow object detection API uses protobuf, a package that implements Google's protocol buffer data format.



### **Implementation Details:**

This project is implemented in Python 3. Tensorflow used for training and classification purposes of the objects, it helps us in detecting, locating and tracking an object in an image or video. And we used OpenCV (Open-Source Computer Vision Library) [\[30\]](#) for feature selection to build an input model for TensorFlow.

The system specifications on which the model is trained and evaluated are as follows:

- Raspberry Pi 3 Model B having 1 GB RAM and a processor with a 1.4 GHz clock speed.
- 64 GB Flash Drive for storage
- Raspberry Pi Camera 5 MP.
- Operating System: NOOBS ("New Out of the Box Software")

### **Pre-processing**

The annotated data is structured in protocol buffer data format and the Tensorflow [\[31\]](#) object detection API relies on Protocol Buffer. These protobuf needs to structure the given data once after successful compilation it can be used with any programming languages like C, C++, Python etc.

## **Environment**

To use the object detection API we need to add it to our PYTHONPATH along with slim which contains code for training and evaluating several widely used Convolutional Neural Network (CNN) image classification models [\[32\]](#).

### Network

The entire network architecture is shown in fig. 13. This model consists of a base network derived from VGG and then modified with CNN layers for fine-tuning and then the classifier and localizer network. These all create Deep-Network which is trained on datasets end to end.

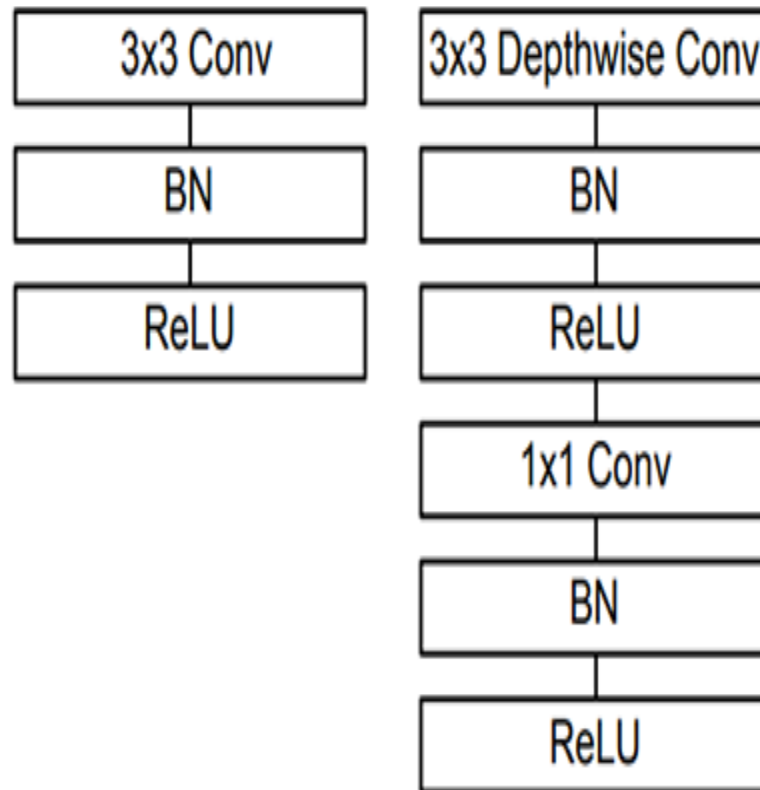


Figure: 14

*(Left)* Standard convolutional layer with batch normalization & ReLU.

*(Right)* Depth wise separable convolution with depth wise & pointwise layers followed by batch normalization & ReLU.



# Experimental Result Analysis:

## Qualitative Analysis:

The result from SSDLite MobileNet V2 Coco is shown in Table below:



Table 1: Detection results on the dataset.  
Figure: 15

The results on the custom dataset are shown in Table 2.


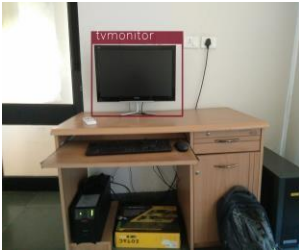





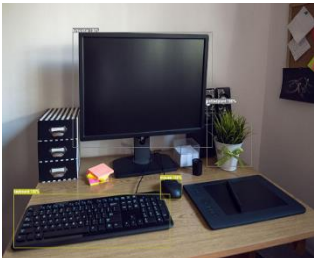

Input	Ground Truth	Prediction
		
		
		

Table 2: Detection results on the custom dataset.

The system handles illumination variations thus providing vigorous detection.



a. High illumination



b. Low illumination

Figure 14: Detection robust to illumination variation

As in the below figure the object is in a sunny environment vs in a shade. However, obstruction/Occlusion creates a problem sometimes. As shown in the figure below. Many persons are not detected by the model and do not have any bounding boxes around them.





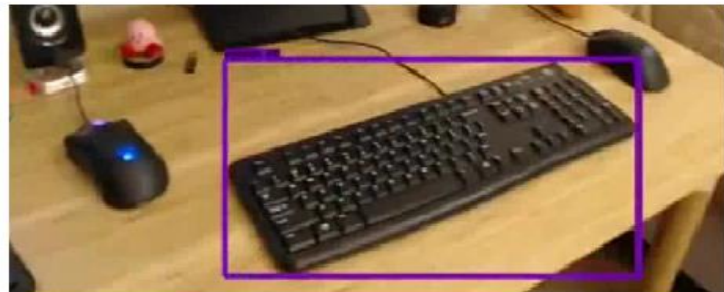
Figure: 17

Also, larger objects are more dominated when presented along with smaller ones. This has been reported in the very next image.

Figure: 18



(A): Only Small Object in Image



(B): Two small and large Object in Image

## Quantitative Analysis:

The evaluation metric used in “mean average precision (mAP). For every given class precision-recall curve is computed. Recall can be defined as the proportion of all positive examples ranked above a fixed given rank. On the other hand, precision is the proportion of all the given examples that rank which is from the positive class of the objects. Here, AP will summarize the shape of the curve (Precision-Recall Curve) and it will be defined as a mean precision at a set of eleven equally spaced recall levels [0, 0.1, 0.2,..... 1].

Thus, to obtain a high score a high precision will be the need of the hour at all levels of recall. This measure is way better than the AUC (Area Under the Curve) because it gives priority to the sensitivity.

The detection was assigned to the ground truth objects and judged to be true or false positives by measuring the bounding box overlapping/occlusion. For a well veracious detection, the area of overlap between ground truth and foretold bounding box must exceed a predefined threshold. The ranking order of the confidence output assigns in decreasing order to those output of the detections assigned to ground-truth objects satisfying the overlap criterion.

Multiple detections of the same objects in an image must be considered as false detection, i.e. 6 detections of a single object counted as 1 true and the other remaining as the false positives.

If no predictions made for a given image, then it considered a false negative.

$$Precision = \frac{TP}{TP + FP}$$

*TP* = True positive

*TN* = True negative

$$Recall = \frac{TP}{TP + FN}$$

*FP* = False positive

*FN* = False negative

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The average precision for all the object categories is reported in table 3. The mAP for the MobileNet dataset was found to be 0.633. The current state-of-the-art best mAP value is reported to be 0.736.

CLASS	AVERAGE PRECISION
MOTORBIKE	0.724
BOTTLE	0.272
BIRD	0.635
CAT	0.909
AEROPLANE	0.727
CHAIR	0.360
PERSON	0.542
DINING TABLE	0.534
BOAT	0.544
TRAIN	0.909
CELL PHONE	0.710
SOFA	0.731
BICYCLE	0.636
BUS	0.726
HORSE	0.726
TV MONITOR	0.633
COW	0.632
CAR	0.634
DOG	0.818
POTTED PLANT	0.359

Table: 3



## Chapter 7 - A Real-time Face Recognition System using PCA and various Distance Classifiers on Raspberry pi 4

### 1 Literature Review

The project on face recognition had helped the author to have a detailed survey of many face recognition algorithms along with their advantages and limitations. Some of the important methods studied will be described in this section. Face recognition systems architecture broadly consists of the three following tasks:

1. Acquisition (Detection, Tracking of face-like images)
2. Feature extraction (Segmentation, alignment & normalization of the face image)
3. Recognition

#### 2.1 Face Detection Approaches

Some of the main face detection methods are discussed here.

- 1) Knowledge-based methods are developed on the rules derived from the researcher's knowledge of human faces. The problem in this approach is the difficulty in translating human knowledge into well-defined rules.
- 2) Featured-based methods: Invariant features of faces are used for detecting texture, skin colour. But features from such algorithm can be severely corrupted due to illumination, noise and occlusion.
- 3) Template matching: Input image is compared with a predefined face template. But the performance here suffers due to variations in scale, pose and shape.
- 4) Appearance-based method: In template matching methods, the templates are predefined by experts. Whereas, the templates in appearance-based methods are learned from examples in images. Statistical analysis and machine learning techniques can be used to find the relevant characteristics of face and non-face images.

#### 2.2 Face Recognition Approaches

LFA method of recognition analyzes the face in terms of local features, e.g., eyes, nose, etc. referred to as LFA kernels. LFA technique offers better robustness against local variations on the facial image in carrying out a match but does not account for global facial attributes. Neural Network are based on learning of the faces in an example set by the machine in the 'training phase and carrying out recognition in the 'generalization phase. But to succeed in a practical set-up, the examples should sufficiently large in number to account for variations in real-life situations. Model Matching methods of face recognition (like Hidden Markov Model (HMM)) train a model for every person during model learning and choose the best matching model, given a query image. Here also big realistic representative models are necessary for good results. A recognition system based on sparse representation computed by  $l^1$ -minimization works with the basic idea of casting the recognition as a sparse representation problem. The main concern in this approach is the presence of a sufficiently large number of features and correct computation of sparse representation. It is a robust and scalable algorithm for face recognition based on linear or convex programming.

### 2 Proposed System

We propose a real-time face recognition system based on PCA and Mahalanobis distance. The main challenge for a face recognition system is of effective feature extraction. The proposed system utilizes

the Eigenface method is information reduction for the images. There is an incredible amount of information present even in a small face image.

A method must be able to break down pictures to effectively represent face images rather than images in general. 'Base faces are generated and then image being analyzed can be represented by the system as a linear combination of these base faces. Each face that we wish to classify can be projected into face-space and then analyzed as a vector. A k-nearest-neighbor approach, a neural network or even a simple Euclidean distance measure can be used for classification.

The proposed system uses Principal Component analysis for feature extraction and various distance classifiers such as the Euclidean distance, the Manhattan distance and the Mahalanobis distance. The technique used here involves generating the 'eigenfaces' then projecting training data into face-space to be used with a predetermined classification method and evaluation of a projected test element by projecting it into face space and comparing to training data.

### 3.1 Steps Involved

The various steps to calculate eigenfaces are:

A. Prepare the data A 2-D facial image can be represented as a 1-D vector by concatenating each row (or column) into a long thin vector. Let's suppose we have M vectors of size N (= rows x columns of image) representing a set of sampled images Then the training set becomes:  $\Gamma_1, \Gamma_2, \Gamma_3 \dots \Gamma_M$

B. Subtract the mean The average matrix  $\Psi$  has to be calculated, then subtracted from the original faces ( $\Gamma_i$ ) and the result stored in the variable  $\Phi_i$

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n$$

$$\Phi_i = \Gamma_i - \Psi$$

C. Calculate the covariance matrix In the next step the covariance matrix A is calculated according to:  
 $A = \Phi^T \Phi$

D. Calculate the eigenvectors and eigenvalues of the covariance matrix In this step, the eigenvectors (eigenvectors)  $X_i$  and the corresponding eigenvalues  $\lambda_i$  should be calculated.

E. Calculate eigenfaces

$$[\Phi]X_i = f_i$$

where  $X_i$  are eigenvectors and  $f_i$  are eigenfaces.

Classifying the faces The new image is transformed into its eigenface components. The resulting weights form the weight vector

$$\Omega_k^T : \quad \Omega_k = \Omega_k^T (\Gamma_k - \Psi)$$

where  $k = 1, 2, 3, 4, \dots$  and  $\Omega_k^T = [\Omega_1 \Omega_2 \dots \Omega_M]$

The Euclidean distance between two weight vectors  $d(\Omega_i, \Omega_j)$  provides a measure of similarity between the corresponding images  $i$  &  $j$ .



**Figure 1: Sample Eigenface images of normal face images**

## 3.2 Various Distance metrics

Let  $X, Y$  be eigenfeature vectors of length  $n$ . Then we can calculate the following distances between these feature vectors

### 3.2.1 Mahalanobis Distance

Mahalanobis space is defined as space where the sample variance along each dimension is one. Therefore, the transformation of a vector from image space to feature space is performed by dividing each coefficient in the vector by its corresponding standard deviation. This transformation then yields a dimensionless feature space with unit variance in each dimension. If there are two vectors  $x$  and  $y$  in the unscaled PCA space and corresponding vectors  $m$  and  $n$  in Mahalanobis space. First, we define  $\lambda_i = \sigma_i^2$  where  $\lambda_i$  is the PCA eigenvalues,  $\sigma_i^2$  is the variance along those dimensions and  $\sigma_i$  is the standard deviation. The relationship between the vectors is then defined as:

$$m_i = \frac{x_i}{\sigma_i} \quad n_i = \frac{y_i}{\sigma_i}$$

$$d(x, y) = \sqrt{\sum_{i=1}^k (m_i - n_i)^2}$$

Where  $\lambda_i$  is the  $i^{th}$  Eigenvalue corresponding to the  $i^{th}$  Eigenvector.

### 3.2.2 Manhattan Distance

Also known as the L1- norm or the Manhattan Distance or the City Block Distance. It is defined as follows:

$$d(x, y) = |x - y| = \sum_{i=1}^k |x_i - y_i|$$

### 3.2.3 Euclidean Distance

Also known as the L2-norm or Euclidean Distance. It is defined as follows:

$$d(x, y) = ||x - y||^2 = \sum_{i=1}^k (x_i - y_i)^2$$

### 3.3 The decision on the test

Having calculated the distance between the two feature vectors, the training image closest to the given test image is returned as the result of the query. If the subject of the test image and the subject of the training image closest to the given test image are the same then a correct match is said to have occurred or else it is taken as an incorrect match. The above approach is tested on the whole database mentioned below, and the results are presented.

## 3 Database

The developed face recognition codes have been tested against the following two standard databases. A total of about 1200 face images of 78 test subjects with varying illumination and pose variations were used in the project. The databases are described in the following sections.

## 4 Codes

The face recognition systems presented here can extract the features of the face and compare this with the existing database. The faces considered here for comparison are still faces. The codes for the project are provided in the folder <https://github.com/AbhishekTyagi404/Deep-Learning-Specialization-Coursera/tree/master/Convolutional%20Neural%20Network/Week%204%20Face%20Recognition>. A README.txt file inside it provides the instructions of compiling and running the system. The code is written in Python and utilizes the TensorFlow and Keras framework.

## 5 The Yale Face Database B



Figure 2: Sample images from Yale Face Database B

The cropped Yale B database used contains 856 grayscale face images in raw PGM format of 38 individuals.[33]. The image resolution is 168(width) x 192 (height) pixels. There are 31 images

per subject, one per different facial expression or configuration: centre-light, with glasses, happy, left-light, with no glasses, normal, right-light, sad, sleepy, surprised, and wink. Sample images are shown in fig 5. The file names of the images in this database have been named in a special order mentioning the pose and illumination detail. The first part of the filename begins with the base name 'yale B' and is followed by the two-digit number signifying the subject number and then by the two-digit number signifying the pose. The rest of the filename deals with the azimuth and elevation of the single light source direction. An example image yaleB03 P06A+035E+40.pgm belongs to subject 3 seen in pose 6, and the light source direction with respect to the camera axis is at 35 degrees azimuth (A+035) and 40 degrees elevation (E+40). Here a positive azimuth implies that the light source was to the right of the subject while negative means it was to the left. Positive elevation implies above the horizon, while negative implies below the horizon. A complete description of the pose and illumination is provided on the web portal of the database.[34] The acquired images are 8-bit (grayscale) captured with a Sony XC-75 camera (with a linear response function) and stored in PGM raw format.

## 5.1 The AT & T Database of Faces (ORL)

The "AT & T Database of Faces" was formerly "The ORL Database of Faces" [35]. It consists of face images 40 distinct subjects with 10 images per subject. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open/closed eyes, smiling / Not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). A preview image of the Database of Faces is available on the official website. A sample preview of images of four different subjects are also shown in figure 3

The files are in PGM format, and can conveniently be viewed on LINUX systems using the default image viewer. The size of each image is 92(width) x 112 (height) pixels, with 256 grey levels per pixel. The images are organized in 40 directories (one for each subject), which have names of form X, where X indicates the subject number (between 1 and 40). In each of these directories, there are ten different images of that subject, which have names of the form Y.pgm, where Y is the image number for that subject (between 1 and 10).



Figure 3: sample images from the AT & T Database of Faces

## 6 Results

### 6.1 Overall Results

The results of running the codes on the two databases are described below in Figure 4

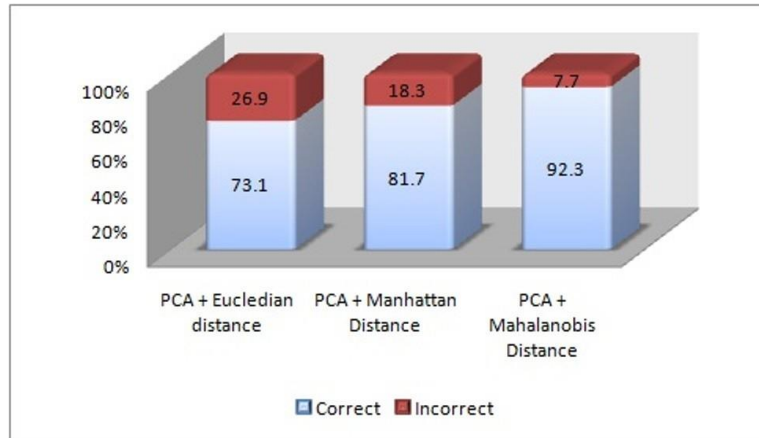


Figure 4: overall Recognition rate (in %) on a total of 1176 face images

### 6.2 The Yale Face Database B

The results of running the codes on the Yale Face Database B are described below in Figure 5 and Table 1

Method Used	Correct	Incorrect	Recognition accuracy
PCA + Euclidean distance	574	282	67.1%
PCA + Manhattan Distance	677	179	79.1%
PCA + Mahalanobis Distance	785	71	91.7%

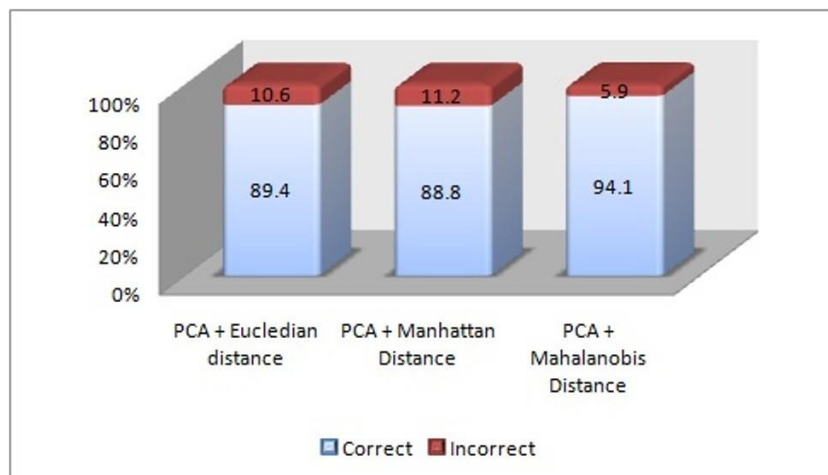
Table 1: Results on The Yale Face Database B

### 6.3 AT& T Face Database

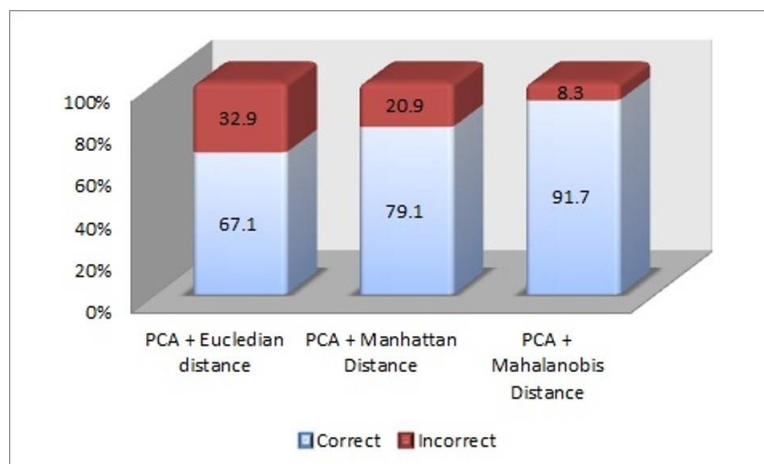
The results of running the codes on the AT& T Face Database are described below in Figure 6 and Table 2

Method Used	Correct	Incorrect	Recognition accuracy
PCA + Euclidian distance	286	34	89.4%
PCA + Manhattan Distance	284	36	88.8%
PCA + Mahalanobis Distance	301	19	94.1%

**Table 2: Results on The AT& T Face Database**



**Figure 5: Recognition rate (in %) on Yale Face Database B on a total of 856 images**



**Figure 6: Recognition rate (in %) on AT& T Face Database on a total of 320 images**



## **6.1 A Realtime Application: Process time**

The processing time of the presented system was also measured. On a Linux based Raspberry Pi (Kernel Linux: NOOBS) operating system with 1.6GHz Processor and 4GB memory, running on our database of 1200 images containing images size 168 x 192 pixels from 78 subjects under various lighting conditions, facial expressions etc of the Yale B database, our OpenCV based C++ implementation of the proposed system using Mahalanobis distance takes 20 seconds for training on the face images and 30 seconds for testing all the images of the database to recognise the test subject.

Once trained the system responds to single face recognition queries in less than 3 seconds. The system completed a query stream of 900 test images in 30 seconds, taking a per-query time slot of just 3 milliseconds.

## **Conclusions of Various Computer Vision Surveillance Techniques on Drone System**

### ***Object Detection on Raspberry Pi 4:***

A precise, accurate and efficient object detection system has been developed with Raspberry Pi 4, which achieves comparable metrics with the existing state-of-the-art system. This project uses contemporary techniques and algorithms to achieve real-world applications of object detection in the field of computer vision and deep learning. This can be used for real-time applications for the traffic management at Red Lights for civilians as well as for surveillance purposes to the armed forces. The raspberry pi achieves 1.2 or 2 FPS with a pi camera (5 MP). Which is efficient to use for real-world tasks. Also, an important scope would be to train the system on a video sequence for usage in tracking applications. And the addition of a temporary network would enable smooth detection and more optimal per-frame detection.

We are using this object detection with Raspberry pi 4 with a drone to come in handy to the user for real-time usage. The Drone's raspberry pi can be controlled by the laptop/cell phone with the help of the VNC Server.

The range of the drone surveillance depends upon the range of the Wi-Fi. And it can be increased infinitely with the help of the 4G GPS, GSM Module and access to the internet.

### ***Face Recognition Real-Time on Raspberry Pi 4:***

This project on Face recognition had given us an opportunity to study many popular methods used in the field of face recognition. The elaborate literary survey provided us with the pros and cons of many recognition systems and the trade-off associated with them. We also came to know that combining two or more techniques can improve the accuracy of the system greatly.

In this project, we have developed a PCA based face recognition system for feature extraction and matching using various distance classifiers. The Distance classifiers used are Euclidean distance, Manhattan Distance and Mahalanobis distance. The results for all three have been presented. The results clearly show that a recognition system based on Mahalanobis distance performs far better than the conventional Euclidean distance-based classifier. The run time of the codes is also considerably fast with a single query response time of fewer than 0.2 seconds.

## Further work

Miniature Unmanned Aerial Vehicles (UAVs) with ability to vertically take-off and land (as in quadrotors) exhibit advantages and features in maneuverability that has recently gained strong interest in the research community. All the simulations were carried out ensuring that the quadcopters entire motion occurs at sufficient height above the ground, and take-off and landing are not performed by the quadcopter. The phenomenon of ground effect has also not been considered at low altitudes. Incorporating the ability to perform vertical take-off and landing overcoming ground effects is a key issue which will be addressed in the future.

Reliability of control systems require robustness and fault tolerance capabilities in presence of anomalies and unexpected failures in actuators, sensors or subsystems. Designing a controller that can combat the failure of one or more rotors is a further step in this project. Based on the inputs from the FDI module, the quadcopter must be equipped to switch to the failsafe controller on the onset of failure. Implementing an adaptive control strategy of this kind is another key issue to be addressed.

## SLAM Based Autonomous Drone:

This paper will further propose a novel UAV autonomous landing approach based on monocular visual SLAM. In the proposed approach, we exploit a feature-based method to estimate the drone's pose and attain the fixed point in three-dimensional space. Regarding grid map construction, we'll first establish an image stream with a corresponding time stamp and barometric altitude information. Then, through the extraction of robust feature points and descriptors, the motion of the drone, the flight path, and the visible three-dimensional point cloud map are obtained by matching and tracking the features. A visible grid map can be built by putting the three-dimensional point cloud to the grid map and calculating the height of each grid after removing noise. After that, a method based on image segmentation is used to divide the height region of the grid map. On the basis of the divided grid map, the system can obtain the appropriate landing place after decision optimization. Finally, the drone will take the shortest path to reach the destination and starts the descent mode near the landing area. After finishing this, the UAV autonomous landing system will successfully be executed. Extensive experimental results on multiple real scenes confirmed that the landing area selection and navigation based on visual technology is effective and efficient. In addition, it can be used to find a suitable path in the field of automated driving where unmanned vehicles and UAVs are used in combination. The drone circles in the air to detect the environment, and then a three-dimensional map is constructed to select the appropriate driving path for the unmanned vehicle. Furthermore, the current work shows great potential in various visual domains. In future work, we will consider extending our work to multiple fields and develop it to the general level.

## References

1. Luukkonen, Teppo. "Modelling and control of quadcopter." *Independent research project in applied mathematics, Espoo* (2011).
2. Hossain, M. Raju, D. Geoff Rideout, and D. Nicholas Krouglicof. "Bond graph dynamic modeling and stabilization of a quad-rotor helicopter." *Proceedings of the 2010 Spring Simulation Multiconference*. Society for Computer Simulation International, 2010.
3. Mahony, Robert, Vijay Kumar, and Peter Corke. "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor." *IEEE robotics & automation magazine* 19.3(2012): 20-32.
4. Dikmen, I. Can, Aydemir Arisoy, and Hakan Temeltas. "Attitude control of a quadrotor." *Recent Advances in Space Technologies, 2009. RAST'09. 4th International Conference on*. IEEE, 2009.
5. Hoffmann, Gabriel M., Steven L. Waslander, and Claire J. Tomlin. "Quadrotor helicopter trajectory tracking control." *AIAA guidance, navigation and control conference and exhibit*. 2008.
6. S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2451–2456, 2004.
7. Zhou, Qing-Li, et al. "Design of feedback linearization control and reconfigurable control allocation with application to a quadrotor UAV." *Control and Fault-Tolerant Systems (SysTol), 2010 Conference on*. IEEE, 2010.
8. Sabatino, Francesco. "Quadrotor control: modeling, nonlinear control design, and simulation." (2015).
9. Das, Abhijit, Kamesh Subbarao, and Frank Lewis. "Dynamic inversion with zero-dynamics stabilisation for quadrotor control." *IET control theory & applications* 3.3 (2009): 303- 314.
10. Lee, Daewon, H. Jin Kim, and Shankar Sastry. "Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter." *International Journal of control, Automation and systems* 7.3 (2009): 419-428.
11. Slotine, Jean-Jacques E., and Weiping Li. *Applied nonlinear control*. Vol. 199. No. 1. Englewood Cliffs, NJ: prentice-Hall, 1991.
12. Y. Amit and P. Felzenszwalb, "Object Detection", *Computer Vision*, pp. 537-542, 2014.
13. "Convolutional neural network", *En.wikipedia.org*, 2019. [Online]. Available: [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network).
14. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 017.
15. "What is a Raspberry Pi?", *Raspberry Pi*, 2019. [Online]. Available: <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>. [Accessed: 06- Mar- 2019]
16. J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer [online]. Available: <https://arxiv.org/abs/1611.10012>.
17. Gandhi, Rohith (July 9, 2018). "[R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms](#)". Towards Data Science. Retrieved March 12, 2020.
18. Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2017.690
19. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu and A. Berg, "SSD: Single Shot MultiBox Detector", *Computer Vision – ECCV 2016*, pp. 21-37, 2016.
20. D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán and Á. Rodríguez-Vázquez, "Performance analysis of real-time DNN inference on Raspberry Pi", *Real-Time Image and Video Processing* 2018.
21. "Jaccard Index", *En.wikipedia.org*, 2011. [Online]. Available: [https://en.wikipedia.org/wiki/Jaccard\\_index](https://en.wikipedia.org/wiki/Jaccard_index)
22. Cortes, Corinna; Vapnik, Vladimir (1995-09-01). "Support-vector networks". *Machine Learning*. **20** (3): 273–

297. [CiteSeerX 10.1.1.15.9362](#). doi:[10.1007/BF00994018](#). ISSN 0885-6125. S2CID 206787478.
23. K. S. a. A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in International Conference on Learning Representations (ICLR), San Diego, 2015.
24. Abadi, M. et al., "Tensorflow: A system for large-scale machine learning," in [12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)], 265–283 (2016).
25. "With Lookout, discover your surroundings with the help of AI", *Google*, 2019. [Online]. Available: <https://www.blog.google/outreach-initiatives/accessibility/lookout-discover-your-surroundings-help-ai/>.
26. GitHub. (2019). TensorFlow detection model zoo. [online] Available at: [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md#coco-trained-models](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#coco-trained-models) .
27. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," *Computer Vision – ECCV 2014 Lecture Notes in Computer Science*, pp. 740–755, 2014.
28. "[Releases - google/protobuf](#)" – via [GitHub](#).
29. S. Mallick, "CPU Performance Comparison of OpenCV and other Deep Learning frameworks | Learn OpenCV", *Learnopencv.com*, 2019. [Online]. Available: <https://www.learnopencv.com/cpu-performance-comparison-of-opencv-and-other-deep-learning-frameworks/#object-detection>. [Accessed: 08- Sep- 2019].
30. *Raw.githubusercontent.com*, 2019. [Online]. Available: [https://raw.githubusercontent.com/adagun/detector/master/models/ssdlite\\_mobilenet\\_v2.pbtxt](https://raw.githubusercontent.com/adagun/detector/master/models/ssdlite_mobilenet_v2.pbtxt). [Accessed: 31- Sep- 2020].
31. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)
32. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
33. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Over feat: Integrated recognition, localization and detection using convolutional networks. In: ICLR. (2014)
34. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Scalable object detection using deep neural networks. In: CVPR. (2014)
35. Szegedy, C., Reed, S., Erhan, D., Anguelov, D.: Scalable, high-quality object detection. arXiv preprint arXiv:1412.1441 v3 (2015)
36. Wright, J. and Yi Ma and Mairal, J. and Sapiro, G. and Huang, T.S. and Shuicheng Yan, "Robust Face Recognition via Sparse Representation," *IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2009*. pp.597 - 604, 2009
37. Fazl-Ersi, E.; Tsotsos, J.K.; "Local feature analysis for robust face recognition," *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, page.1-6 July 2009
38. Delac K., Grgic M., Grgic S., "Independent Comparative Study of PCA, ICA, and LDA on the FERET Data Set, *International Journal of Imaging Systems and Technology*, Vol. 15, Issue 5, 2006, pp. 252-260
39. D. Bryliuk and V. Starovoitov, "Access Control by Face Recognition using Neural Networks and Negative Examples, 2nd International Conference on Artificial Intelligence, pp. 428-436, Sept 2002.
40. Wright, J. and Yi Ma and Mairal, J. and Sapiro, G. and Huang, T.S. and Shuicheng Yan "Sparse Representation for Computer Vision and Pattern Recognition" 2010 Proceedings of the IEEE, volume 98 pp 1031 -1044
41. Massimo Tistarelli, Manuele Bicego, Enrico Grosso, Dynamic face recognition: From human to machine vision, *Image and Vision Computing*, Volume 27, Issue 3, Special Issue on Multimodal Biometrics - Multimodal Biometrics Special Issue, Pages 222-232, February 2009