Implementing machine-vision to track moving objects

Developing software for visual guided UAV Navigation

Rohan Yogesh Dwivedi

A thesis submitted for the degree of Master of Science in
Intelligent Systems and Robotics

Supervisor: Dr. Dongbing Gu
School of Computer Science and Electronic Engineering
University of Essex

August 2020

Abstract

Quadrotor navigation using computer vision is an open research problem. To be able to track objects using quadrotors autonomously has important applications. This thesis discusses a project to develop a computer vision-based tracking mechanism to detect and track objects from a commercially available quadrotor using only a monocular camera. The project is based on using YOLOV3 to detect objects by training the model on synthetic training data collected using simulation and implements a Kalman filter and PID to track the vehicle. The project utilises the sphinx simulation environment to test the implementation instead of testing in the real world. It employs the open-source implementations of bebop autonomy; a ROS driver developed by autonomy lab of Simon Fraser based on the Ardrone SDK provided by parrot. It also makes use of an open-source implementation of YOLOV3 to train the model and modifies its detector to pass the coordinates of the bounding box as a ROS message. A method to perform inverse projection of 2d image coordinates to 3d world coordinates is also discussed in this thesis. Fairly good tracking was observed on the simulated tests, the performance of the tracking and its limitation is thoroughly analysed.

The document is organised in a way that we first discuss the objectives of this project and why it is needed in the introduction section. It is followed by an extensive literature review to obtain an understanding of the problem, concepts and available solutions and their limitations. Further, the document gives a brief explanation of the technical concepts required to be known before attempting to understand the methodology discussed in the section that follows. The final sections deal with illustrating the performance of the implementation followed by an analysis of the weaknesses and challenges. The document concludes with a note on the future direction of this work.

Contents

1 Introduction **4**

2 Literature Review **8**

3 Background **15**

3.1 Mathematical model of a Quadrotor 16

3.2 Computer vision 18

3.2.1 Geometry of image formation: Pinhole Camera 18

3.2.2 Camera calibration 20

3.2.3 Relationship between world and image coordinates . . 21

3.2.4 Object detection using YOLO 21

3.3 Kalman Filter 24

3.4 PID 26

4 Methodology - software implementation **27**

4.1 Simulation environment: Parrot Sphinx 27

4.2 System Architecture 28

4.3 Implementation 30

4.3.1 ROS 30

4.3.2 Object detection 31

4.3.3 Tracking using Kalman Filter 32

4.3.4 Image coordinates to world Coordinates 34

4.3.5 Position control: PD controller 35

5 Results **37**

5.1 YOLOV3 Performance 37

5.2 Tracking Results 40

5.2.1 Experiment 1 40

5.2.2 Experiment 2 44

6 Analysis **47**

7 Conclusions and future work **49**

References

Appendices

1 Introduction

The term Robot first occurred in a play titled "Rossum's Universal Robots" by a Czech author Karel Chapek in 1920. The word Robot has roots in the Czech noun robota which means to do hard work or labor[1]. A robot is simply a machine that is capable of carrying out complex tasks (labour). Moreover, the definition of the robot can be extended to, for any machine to qualify as a robot, it must be able to perform its assigned labour 'automatically', which means without any external intervention. Also, the robot must consist of a control apparatus that guides its action. The closed-loop control apparatus involves sensory feedback and actuation of the end effector[2]. Usually, a robot is constructed in a configuration that is most suited to the task and its mechanical system is designed to have the most efficient way of completing the required labor[3].

According to [1] robots can be classified in many different ways. The classification can be based on its motion: stationary or mobile; or it could be based on the medium where it operates: land, water or air. It follows that any aircraft that is automated by the virtue of its control software will fall under the category of an aerial-mobile robot. [4] explains that an Unmanned Aerial Vehicle (UAV) is an umbrella term encompassing all aircraft's that have no human pilot's onboard. They come in different sizes and configurations depending upon their intended applications. UAV's can either be operated remotely or be entirely automated. He states that an "autonomous drone" or in other words a flying robot can operate without any external intervention such as by a human, that is, it should be able to take off, carry out its objective and land with the software acting as its pilot. [4] further clarifies that a flying robot is a type of UAV but not all UAV's are flying robots.

[5] points out that UAV's come in many different flavours, the most popular and widely available configuration is that of a quad-rotor. A quad-rotor comprises of four rotary blades configuration as shown by figure 1. A quad-rotor behaves more like a helicopter in contrast to a fixed-wing aircraft. The thrust generated due to the combination of all four rotors allows the quad-rotor to have vertical takeoff and landing, this proves advantageous for research projects carried out in indoor labs [1][5] which is a compulsion arising due to unfavourable laws [6]. Another advantage of a quadrotor is that they are lightweight, cheaper to produce and are commercially available with a system on chip (SOC) onboard and are equipped with various sensors such as inertial measurement units, camera(s), GPS etc [1].

Some commercially available quad-rotors are shipped with software development kits or SDK that allow the user to program/automate the quad-rotors and are mostly targeted towards customers that want to research different aspects of aerial robotics, for instance, parrot drones are a commercial drone company that has open-sourced their SDK [7]. Considering all of these advantages, a commercially available quad-rotor is the most popular choice for UAV research.

Figure 1: Parrot Bebop 2 Quadrotor [8]

A considerable amount of work has been carried out in the area of field robotics in the previous decade, it includes developing various kinds of legged or wheeled vehicles [9]. But, the application of wheeled robots can be constrained due to factors such as difficult terrain, unavoidable obstacles or vast area of operation, therefore, aerial robotics prove to be extremely important in such situations. Aerial robotics have been developed for a wide range of applications such as defence applications, for instance, carrying out surveillance, target tracking and strike missions to civilian applications such as e-commerce logistics, agriculture, remote sensing and disaster response[10]. But aerial robots are not without their unique challenges.

One of the many obstacles to automated UAV's is implementing a navigation system. Since there is a need for the aircraft to automatically navigate to the desired position, the aircraft has to rely on onboard sensors for the perception of its environment. However, in some complex environments, the sensors might not be entirely reliable due to communication and perception weaknesses of traditional sensors [11]. Usually, the UAV relies on GPS and inertial measurement unit on-board to navigate via waypoints. Waypoints are a set of GPS coordinates set along the desired path of operation, these are provided to the flight controller before the beginning of operation [12]. But in cases such as when the UAV has to operate in a GPS denied area; or if there is some interference with the sensor operation; or if the task is dynamic and real-time navigation is required, then there arises a need for having a robust navigation control system in place that does not solely rely on the GPS waypoint system. Early researchers relied on lasers or radar sensors for circumventing this challenge but these are complex and expensive solutions[11]. These days, almost all UAV's come with inexpensive cameras on board. The camera's primary purpose is aerial photography, but it provides with an opportunity to implement an efficient vision-based control paradigm for navigation, guidance and control[13]. The study on the navigation of birds and flying insects further motivates to implement a vision-based navigation system [13].

Machine vision is defined as the transformation of data from a still camera or a video camera into either a decision or new representation. All such transformations are done to achieve a particular goal [14]. [15] states that visual tracking is one of the fundamental challenges of machine vision. Furthermore, [15] defines visual tracking as automatically predicting the state of the object being tracked in a sequence of image frames, given the initial position and the size of the object. The main challenges lie in the partial occlusion of the subject during tracking[15], environmental factors such as bad lighting and uniform texture in the scene][16], fast and abrupt motion of the subject and large variations in the pose of the camera etc[15]. It has applications in the video surveillance, autonomous vehicle navigation and human-computer interaction [15].

[17] asserts that navigation of flying robots in GPS denied environments is still an open research problem. Robots are often utilised in situations such as search and rescue operations that require a high degree of autonomy and reliability [18], and in that case, the unreliable GPS or radio signals cannot be counted upon for successful navigation. Detection and tracking have very interesting applications says [19] - "including following a suspect or a fugitive in persecution, or video recording people in hazardous situations, like while practising extreme sports"

This thesis aims to demonstrate that a machine vision-based tracking software can be implemented that mitigates the above-discussed challenges. The objective here is to implement a closed-loop controller to detect and track the movement of a vehicle and follow it as closely as possible. This thesis argues that inexpensive and reliable tracking can be achieved based on novel deep learning methods for object detection such as YOLO in tandem with machine vision techniques, Kalman filter and PID controller for navigation of a quad-rotor equipped with nothing but a monocular camera in a GPS denied environment. This thesis seeks to support its claims by evaluating the performance of the aforementioned control software using simulated tests.

This document is divided into several sections. The thesis begins with a discussion and review of relevant research in this area and tries to ground the work presented in this document in the context of the literature available. It follows an attempt to briefly explain the technical background that forms the basis of the work carried out. The next section deals with the implementation of the software followed by the results of the evaluation. The thesis concludes with an analysis of the results achieved and possible future work.

2 Literature Review

Peter Corke [1] defines robot navigation as the problem of guiding the robot towards a goal. He observes that when it comes to humans for navigation, we tend to use maps and signs. Robots do not require maps for many tasks that have an element of navigation to them. [1] refers to it as reactive navigation. However, he contrasts that for some complex tasks robots do need to perform motion planning which requires self-localisation and the ability to map their environment. But, since the objective here is for the UAV to track the vehicle using machine vision, therefore, its position must be known to the controller relative to the position of the UAV throughout the course of the navigation. Thus the UAV controller must rely on reactive navigation wherein the UAV reacts to the information obtained directly from the environment. In this case, the position of the vehicle is obtained from the camera using machine vision.

Several control methods are being studied such as PID, back-stepping, nonlinear control etc. All of these control methods require precise position and attitude information usually obtained from a gyroscope, accelerometer and other devices such as GPS [20]. [16] and [17] discern that navigation of aircraft in GPS denied environments are an open research problem which according to [18] has become of increasing interest lately. Moreover, [16] writes that current applications of UAV's are limited to observation and military missions where the aircraft are mostly manually operated. [18] observes that if the applications of UAV's have to go beyond the military applications in areas such as search and rescue, flying robots need to have a very high degree of autonomy. [16] agrees to [18]'s premise and proposes that to extend the operation of UAV's they must be able to operate through environments without relying on GPS or other sensors. This is because the environment might be GPS denied, [18] adds that stable radio links with high bandwidths or reliable GPS connection's are not always available in some environments such as urban areas, which limit the applications of UAVs. [16] mentions that the current state of the art commercially available UAV automation relies on preloaded GPS way-points and the UAV is constrained to following the said way-points. [17] supports this premise and adds that previous work regarding UAV navigation had focused on developing reliable control strategies and therefore had to adopt the offline maps strategy because of the challenges of using low-cost hardware that leads to noisy sensors and inaccurate actuation's.

[21] agrees that GPS based navigation is a problem and proposes that navigation strategies must adopt cameras and develop vision-based controllers. His reasoning is grounded upon the applications of the UAV, for example, he states that for tracking moving ground-based objects, localisation methods such as GPS do not give information about the position of the object relative to the UAV, in cases like such, cameras prove to be extremely useful. But [16] puts a caveat for adopting vision-based methods, he argues that camera-based navigation is difficult mainly due to the monotonous textures of the scene or bad lighting conditions. He also takes into consideration that perception algorithms must be robust to outlier measurements that result from low-level image processing. [18] seconds the argument, he says that since the control of the UAV must be real-time, the algorithm must be fast, robust and accurate since any delays or incorrect measurements may lead to crashes.

[18] also asserts that the choice of the sensor is the most crucial to the problem of visual navigation. [16] is of the view that depth sensors are the ideal solution. [16], [17] and [18] all agree that active sensors like laser scanners and radars are heavy and consume a lot of power but they disagree on what kind of passive senor would be the best. [18] believes that Kinect sensors would be a good replacement since they are lightweight and consume less power but they only work indoors. Therefore he becomes a proponent of using stereo camera rigs, they are light and have great resolution. [17] advocates using monocular cameras, he argues that the amount of information obtained is quite high in comparison to the low weight and power consumption, size and cost. [16] sides with [18], he contends that monocular camera does not provide depth information without knowing additional parameters like the scale of the object, which is not a requirement for stereo setups. [17] agrees that it is a drawback.

But, since the commercial UAV's already come equipped with a monocular camera, the function of which is mostly aiding the user in remote control or videography, the solutions to visual navigation should be based on using monocular cameras. Also, according to [22], stereo rigs made by using two cameras have an inherent problem which is that post-calibration the cameras must not even slightly move from its position, since even a small shift can cause large variations in measurement.

[16] summarises different approaches to monocular visual navigation. First of these methods is based on optical flow. Optical flow is based on the premise that the difference between frames of a video sequence is due to the motion of the pixel(s) from one image position to another. The method he presents is based on compensating the loss of velocity information by estimating the 3d position and velocity of both the UAV and Object by using optical flow and extended Kalman filter. He claims the accuracy of this method is similar to GPS, but additional laser-based sensor for height estimation is required. The weakness of the optical flow method is the assumption that the variation in image sequences is caused by the motion of pixels which is not always true, motion blur or occlusions etc can also cause this differences. This method also partially relies on sensors to obtain attitude and altitude of the UAV, hence also contended by [17] the navigation is subjected to drift and therefore not suitable for long term navigation.

The second method presented by [16] is visual odometry. This method relies on exploiting the 3d measurements of a stereo camera to get the depth information. However, this method can be modified to obtain the depth information from the SFM technique that requires scale information as shown by [19]. This method can obtain the rotation and translation using the 3d information. The main advantages of this method over other methods are that it can be applied to any scene, and it is fast if there is a GPU based implementation. But it does not take into account successive error accumulation over time due to transformations. [21] argues that this is a good approach only if estimating pose relative to starting position, he prefers visual servoing strategy which is based on optical flow. [18] suggests using a combination of the way-point following and obstacle avoidance strategy and stereo odometry based path planning.

Another approach discussed in the article [16] is that of V-SLAM. This is a mapping-based technique that takes the aid of landmarks in the environments to create for itself a map of the environment. It takes into the picture the accumulation of errors by the visual odometry method but it is a computationally expensive continuous update and correction of the environment is required. Even then the map built is not completed if there a lot of obstacles or the robot is operating in a congested area. This is not suited for an onboard computations perspective.

[23]'s approach to tracking a vehicle is to obtain the position and velocity of the drone using an external tracker (VICON motion capture system) and then design a PID controller to follow the target. [17] asserts that advanced external tracking is not very useful in real-life situations, they perfect the control strategies but do not solve the navigation problem.

An interesting strategy to vision-based navigation is shown by [19]. He presents a system for tracking a mobile object with unknown dynamics. The principle idea is to detect the target of interest using any object detector in the image sequences and then track the object using Kalman filter thus obtaining an estimated position of the object with respect to the UAV. Later a linear controller is used to maintain the drone position relative to the position of the target. Demonstrating the concept [19] trains a haar cascade classifier to detect a human face and shows that the drone can follow the target on interest while maintaining a constant distance from it. As far as tracking concerned, a summary of related work is available from [19]. The summary highlights that the existing work mostly concerns with tracking people's faces or gesture recognition with or without markers. However, the approach by [19] can be adapted to non-humans especially in the case of tracking a vehicle from UAV which this thesis attempts to do.

Several approaches to object detection exist with varying degrees of performance both in terms of accuracy and speed. [24] states that feature descriptors such as SIFT and SURF are major well-known strategies in computer vision for object detection. [25] expresses that Convolutional Neural Networks (CNN) has made a big contribution to the ability of computer vision to recognise objects. He also contends that this has been possible due to increasing computing power and availability of data. [25] explains that traditional approaches such as SIFT, SURF, BREIF etc that are well established for object detection involve many steps such as edge detection, corner detection, thresholding etc to obtain meaningful features. The difficulty is to get features that are important to each image every time. As the number of classes increases, feature extraction becomes more and more difficult and the CV engineer must fine-tune every parameter to get acceptable results. However, he contends that in the case of deep learning-based approaches trained data can be used to train the model once, where neural networks discover the best and salient underlying features for each class automatically and he asserts that it has been accepted that the performance of neural networks far outweighs the traditional approaches.

[26] agrees with [25] and points out that Haar-cascade classifiers used to be the best methods available for object detection before the availability of deep learning-based methods. He justifies his assertion by carrying out a series of tests and he finds that both CNN and Haar cascade classifier have a high precision and recall rate, Haar cascade classifier outperforms CNN in scale invariance and requires less time to set up. However, CNN beats Haar cascade in processing time. CNN is also much better with orientation variance. He concludes that is better to use CNN if training time is not a problem and objects do not scale too much. [27] also support this argument in their comparison of Haar and CNN methods for detection of surgical instruments, they conclude that both methods have drawbacks in changing lighting conditions, however, CNN performs better whereas Haar-cascade has more number of false positives. [27] agrees with [26] that for real-time applications Haar-cascade is very slow in processing whereas CNN can extract features in very short periods. [27]'s results also show that CNN has better accuracy although it is not a very huge difference.

[28] argues that ideal detectors should have a higher accuracy of detection and localisation as well as must be time and memory efficient. [28] explains that CNN based object detectors can be classified mainly in two categories, single-stage detectors and two-stage detectors. Single-stage detectors are faster, consume less memory but are relatively less accurate. Single-stage detectors treat object detection as a regression problem, they take the training image and learn the class probabilities and bounding box coordinates whereas two-stage detectors generate regions of interest based on having higher probabilities of being the object and then in the second step apply the regression, this makes them more accurate but slower. He lists and compares several single and two-stage detectors. Out of which detectors of most interest are the single-stage detector YOLO and multi-stage detectors RCNN and Faster RCNN. He finds that Tiny YOLO is the fastest of all the methods on the Pascal VOC data set but less accurate, YOLOv2 is a good compromise between speed and accuracy.

YOLO was presented by [29] in 2016 as a state of the art object detection method. YOLO works on detecting class probabilities of multiple bounding boxes all at once, it trains on full images and optimises its performance accordingly. As claimed, the advantages of YOLO lies in the fact that it is extremely fast in comparison to other real-time object detectors. The author also claims that YOLO makes fewer than half of the background errors made by Fast RCNN. However, YOLO lacks in accuracy than other two-stage detectors like Fast RCNN. It also struggles with localising small

objects. An improvement over YOLO has been suggested in YOLOV3 by the authors of YOLO [30]. According to [31] the major improvements in YOLOV3 are that the average precision for the small object has improved with fewer localisation errors and better mean average precision. Predictions on different scales improved.

Figure 4 shows the performance of the original YOLO algorithm vs RCNN on the VOC 2007 dataset. It shows the percentage of localisation and background errors in the top N detection of various categories.

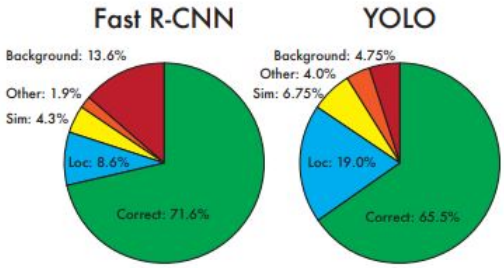

Figure 2: YOLO Performance [30]

Once the object has been detected, it can be tracked. There are several ways tracking can be performed using vision as an input. [32] lists a few of them; one of them is mean shift which is a probabilistic method based on hill-climbing posterior kernel [33]. It is fast but cannot be used when the object tracking is lost. The other techniques are Kalman filter, extended Kalman filter and particle filter etc. Kalman filter is a probabilistic method based on Bayesian networks where the error is assumed to Gaussian and the system is linear. There are also other variations of Kalman filter like the extended Kalman filter which assumes the error to be non-Gaussian [33]. For systems where the errors are non-Gaussian and non-linear, there exists a Monte Carlo method as well. Kalman Filter is the most widely used in signal analysis and guidance systems which has proven to work best with vision-based inputs as well [34]. [35] in their work compare particle filters and Kalman filters and conclude that particle filter is more reliable and robust for multi-tracking applications in complex situations but a Kalman filter performs better for tracking single objects.

[32] explains Kalman filter as follows: "The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process in several aspects: it supports estimations of past, present, and even future states, and it can do the same even when the precise nature of the modelled system is unknown. The Kalman filter estimates a process by using a form of feedback control. The filter estimates the process state at some time and then obtains feedback in the form of noisy measurements. The equations for Kalman filters fall in two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimate for the next time step. The measurement update equations are responsible for the feedback. That is used for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate. The time update equations can also be thought of as predictor equations, while the measurement update equations can be thought of as corrector equations."

3 Background

This section explains the relevant concepts and defines terms that make the basis of the implementation of the project. The first subsection details the mechanics of a flying robot in the context of its motion and control. That is followed by a discussion on deep learning-based object detection methods elucidating the implementation of YOLOV3 algorithm that has been chosen for this project. The final subsection deals with the theory of Kalman filter followed by a PID controller.

Before we delve into discussing the mathematics of a quadrotor, we need to define and understand a few terms related to the motion of flying machines.

1. Inertial frame of reference: The frame of reference that is fixed to the ground on the origin. Aeronautical inertial frame convention is x-axis points north, y-axis points east, and the z-axis points down[20].

2. Vehicle Frame: The origin is at the centre of mass, the axes are aligned to the inertial frame

3. Vehicle-1 frame: Origin is the same as vehicle frame, the axes are rotated by the yaw angle ψ [20].

4. Vehicle-2 frame: Origin is the same as vehicle frame, the axes are obtained by rotating the axes of the vehicle-1 frame by pitch angle θ [20].

5. Body frame: This is the reference frame that is aligned to the sensor. The origin of it is in the centre of mass of the quadrotor, the axes are obtained by rotating the axes of the vehicle-2 frame by roll angle ϕ [20].

6. Center of Mass (COM), there is a certain point "inside" the body, called the centre of mass, such that the net resulting external force produces an acceleration of this point, just as though the whole mass were concentrated there."

7. UAV's have 6 degrees of freedom available to them, these are up-down, left-right, clockwise-counter clockwise. Technically these axes are termed as roll, pitch and yaw[1].

    - Pitch: It is the orientation of the nose of the aircraft in either facing upwards or downwards.
    - Yaw: It is the freedom of movement clockwise or anticlockwise around the vertical axis.
    - Roll: It is the freedom of movement towards left or right around the horizontal axis.

8. Rotation Matrix: The matrix that rotates the drone from the inertial frame of reference to the body frame.

3.1 Mathematical model of a Quadrotor

The mathematical model of any robot travelling through a fluidic medium, be it airborne or waterborne is subjected to their model is expressed in terms of forces, torques and acceleration rather than their velocities. Hence a dynamic model is constructed instead of a kinematic model[1]. Figure 2 [a] is a representation of a dynamic model of the quadrotor being considered[27] and fig 2 [b] demonstrates roll, pitch and yaw axes of the quadrotor.

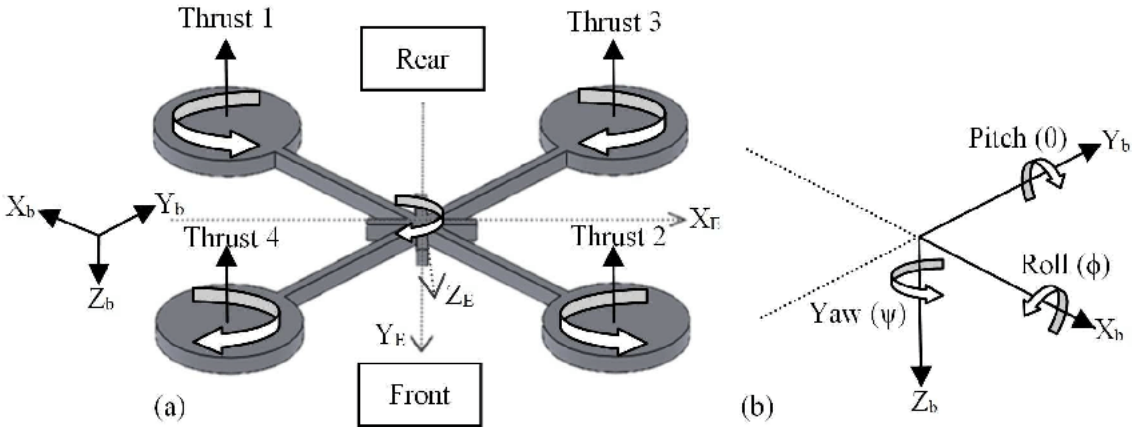

Figure 3: Roll,Pitch and Yaw [27]

An important detail [36] to note is that quadrotors have four rotors at fixed angles to each other, the motions are caused due to the thrust generated by these rotors. Different configurations can be arranged by varying the angles between the rotors, but the most stable and widely accepted configurations are the '+' and 'x'. Moreover, in comparison to plus-config, x-config is considered more stable. Bebop 2 used for this project is also an x-config drone. As shown in figure 2 [a], the equal thrust generated by all rotors allows the quadrotor to lift off. The caveat is that propellers 1 and 3 that are opposite to each other must rotate counter-clockwise and propellers 2 and 4 clockwise so that the net torque around the yaw axis becomes zero, otherwise the quadrotor will yaw in the direction opposite to the direction of the propeller's rotation. This allows the quadrotor to maintain forward-backwards movement by increasing-decreasing the speed of front rotors and back rotors respectively. Pitch and roll changes are achieved by applying more thrust to one set of rotors than to the opposite pair of rotors. Yawing can be achieved by applying more thrust to the rotors that are spinning in the same direction.

Equation 1 [1] derived from newtons second law of motions explains the mechanics for the translational motion of the quad-rotor in inertial coordinates.

$$m\dot{v} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} - 0_{R_B} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} - Bv \tag{1}$$

Derived from Newton's law of motion $f = m * a$, equation 1 gives the force required to lift the quadrotor against gravity and air resistance. $\dot{V}$ is the acceleration of the centre of mass of the aircraft in the inertial frame; it is the derivative taken of velocity with respect to time. 'g' is the acceleration due to gravity that is acting downward taken to be 9.81 m/s^2, m is the mass of the aircraft and B is the aerodynamic friction. T is the total upward thrust produce by all the propellers of the aircraft combined given by the equation

$$T = \sum T_i \tag{2}$$

In the equation downward force of gravity on the aircraft is subtracted from the total thrust produced by the vehicle rotated into the inertial frame by multiplying it with the rotation matrix 0_{R_B}. Rotation matrix provides the transformation of mass B with respect to the inertial frame. Aerodynamic drag caused by the air resistant is subtracted from the resultant.

Quadrotors can be represented by the coordinates [37]

$$q = (x, y, z, \psi, \theta, \phi)\epsilon R^6 \tag{3}$$

here (x,y,z) represents the position of the centre of the mass of the quadrotor with respect to the inertial frame and (ψ,θ,ϕ) are the Euler angles yaw, pitch and roll respectively that represent the orientation of the UAV. Using Euler-Lagrange equations the simplified dynamic equation for the motion of the aircraft can be obtained [37] [19].

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \approx \frac{T}{m} * \begin{bmatrix} sin\psi * sin\phi + cos\psi * sin\theta * cos\phi \\ -cos\psi * sin\phi + sin\phi * sin\theta * cos\phi \\ cos\theta * cos\phi \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{4}$$

3.2 Computer vision

This section attempts to explain the basic concepts required to understand for tracking using computer vision. We attempt to understand how an image is formed on the sensor of the camera, and how can we utilise this knowledge to obtain meaningful information from it to utilise it for vision-based navigation.

3.2.1 Geometry of image formation: Pinhole Camera

A very simplistic model to explain how an image is formed on the camera is given by the Pinhole camera model. A pinhole camera is set up with a barrier between the object and the film with a very small aperture that allows enough light in to form an image. The small aperture allows every point on the object to be mapped to the screen forming an image [38]. Figure 3 shows the formal construction of the camera. Equation 5 [14] provides the relation between the size of the object and the focal length of the pinhole camera.

$$-x = f\frac{X}{Z} \tag{5}$$

Here x is the object size on the image plane, X is the actual size of the object, f is the focal length, and z is the distance between the camera and the object.

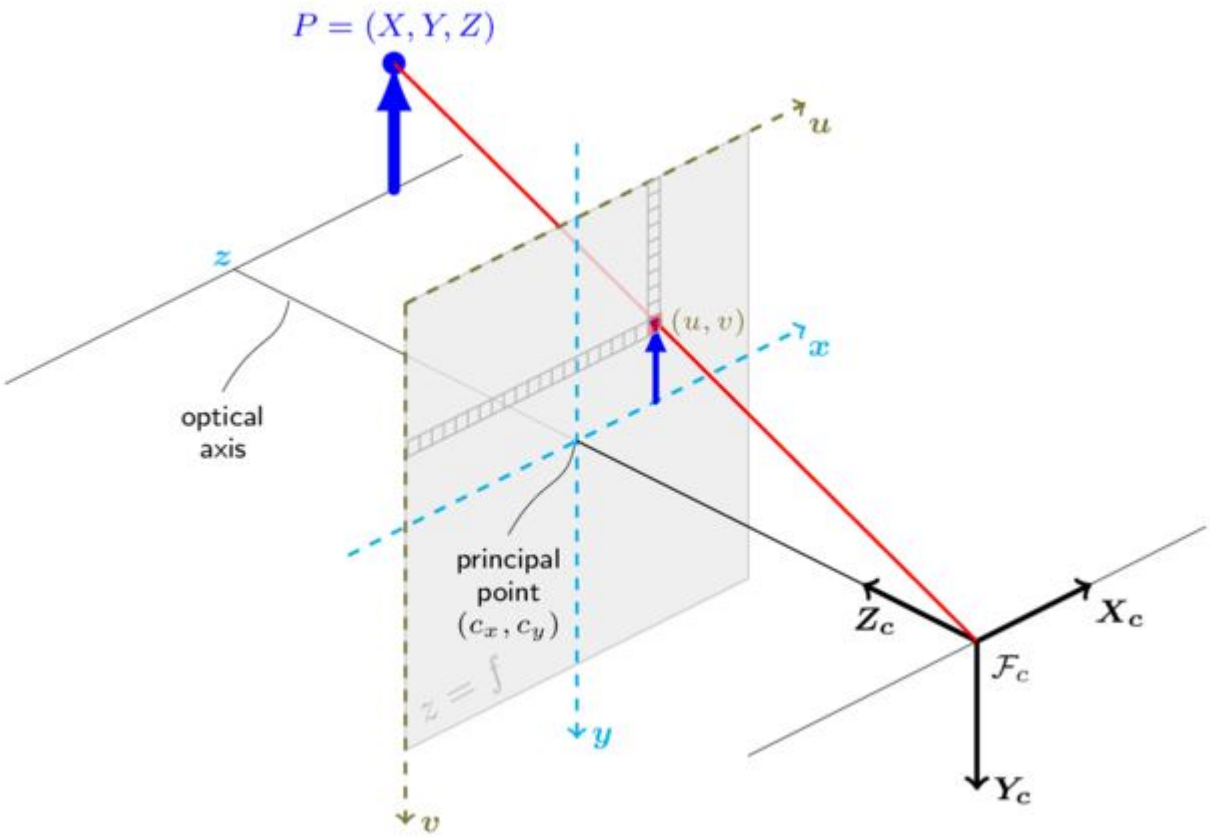

Figure 4: Pinhole camera model [39]

Practically speaking, a pinhole camera is a generalisation and doesn't work very well in real-life applications because due to the very small size of the aperture it cannot capture enough light rays for rapid exposure [14]. With a larger aperture, more light passes through resulting in every point in 3d space being represented on the image plane resulting in a blurry image, if we reduce the size of the aperture sharp and clear image can be obtained but the resulting image will be darker [38]. To solve this problem we introduce lenses to our pinhole generalization.

[38] explains that lenses are devices that can focus or disperse light. refraction caused due to the lenses allows all rays of light emitted from a point to converge to a single point on the image frame. Lenses allow cleared image to be obtained for all points that are in its depth of field; depth of field is the property of the lens. But lenses can also cause distortions. But since we want to use the pinhole model for mathematical simplification in our com-

puter vision application, we need to correct for this distortion by a process called calibration. Camera calibration is also required to find out the relation between the points in the 3D world and pixels in the image 2d world. Camera calibration gives us the parameters for camera's geometry and the lens distortion model which constitutes the intrinsic parameters of the camera.

3.2.2 Camera calibration

3d points are projected into the image plane using perspective transformation in the case of a pinhole model. The transformation is given by the equation [39] below.

$$s * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & Cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{6}$$

Here (X, Y, Z) are 3D Coordinates of the point being photographed. (u,v) are the coordinates of the point in the image plane after transformation. (Cx, Cy) is the point of the image centre and (fx,fy) are the focal lengths in pixels. The matrix comprising of parameters (Cx,Cy,fx,fy) is called the intrinsic parameter matrix [39]. The second matrix r—t is the extrinsic parameter matrix comprising of the rotation and translation of the camera with respect to the inertial origin [39]. The whole point of the calibration exercise is to obtain these parameters which can then be used to do the inverse transformation for computer vision applications [39]. OpenCV provides calibration function to obtain parameters using checkerboard method [39], however, most drone manufacturers provide the parameters with their SDK [7].

3.2.3 Relationship between world and image coordinates

If we consider the pinhole camera model and thereafter obtain the calibration parameters, the relationship between world coordinates (Xw, Yw, Zw) and image coordinates(Xim, Yim) is given by the following equations [19]

$$X_i m = f_x * \frac{X_w}{Y w} + C_x \tag{7}$$

$$Y_i m = f_y * \frac{Z_w}{Y w} + C_y \tag{8}$$

3.2.4 Object detection using YOLO

YOLO[30] is a deep learning-based object detection model recently developed. The salient features of YOLO is that it performs single regression-based detection, that allows it to look at the image only once and find out all the image present and their probabilities. The biggest advantage of YOLO over other detectors is that it fast yet fairly accurate to other models hence can be used for real-time detection which we attempt to do in this project.

A few important terms [40]:

- Classification refers to categorising objects into different classes based on their shared attributes
- localisation refers to pointing out the location of the object in the image.
- object detection refers to localising the object in the image and then classifying it into a category based on its probability.

YOLO is based on regression rather than classification, it predicts the classes and bounding box (location) of the object(s) at once rather than the slow method of classification which is selecting a region of interest (ROI) in the first step and then finding out the class probabilities in the next step [40].

A bounding box can be described by 6 parameters. These are Any 2 corner points; left/right and top/bottom along with the width and height of the box. The other parameters are the confidence of the detection and the probability Pc that the bounding box contains an object [40].

The image is split into grids, and each grid predicts multiple bounding boxes to cover cases if there is more than one object present in the grid. The parameter Pc allows us to discard all the grids that have a very low probability of containing the object, hence reducing the vast search space. This is also termed as non-max suppression[40]. It also removes all the bounding boxes that have a very high intersection of shared area [40].

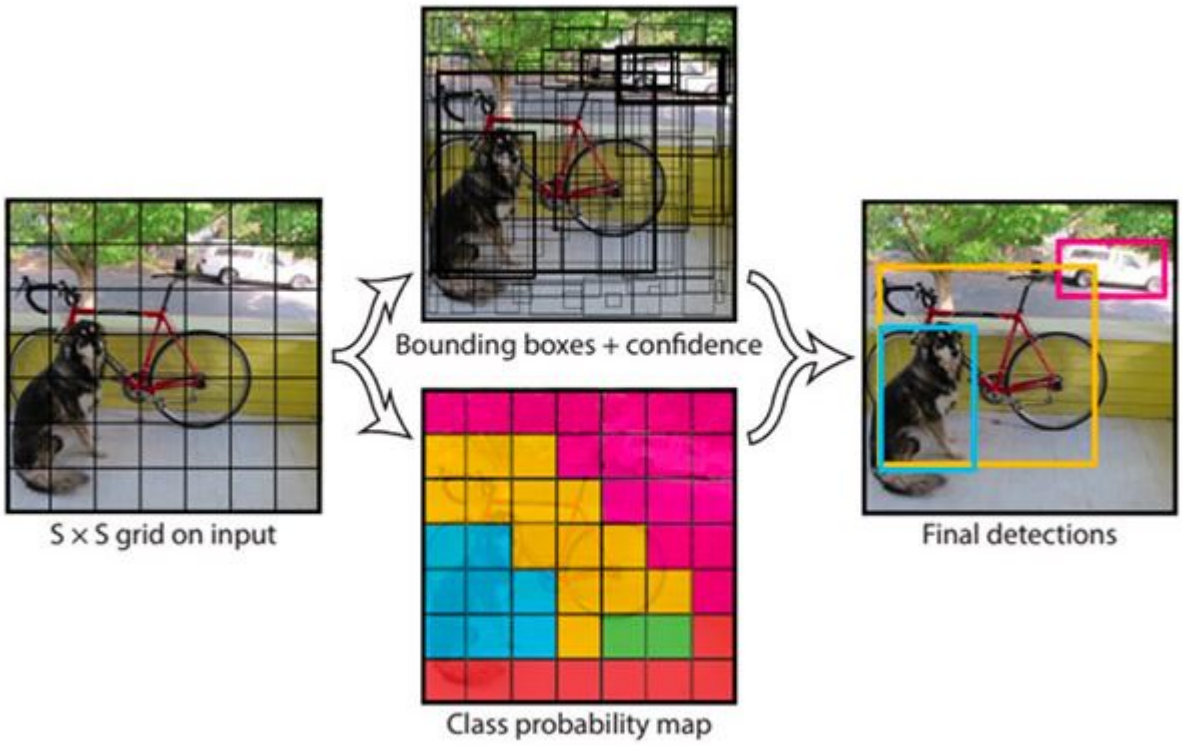

Figure 5: YOLO Detector Pipeline [29]

YOLO utilises the CNN architecture[29]. A convolution is a mathematical function that expresses how the shape of one function is modified by another function. A CNN has an input layer and output layer with multiple hidden layers in between, these hidden layers are usually a series of convolutional layers. The initial layers extract the features from the image an while the later fully connected layers predict the output probabilities and Coordinates.

The input to the CNN is a tensor with shape (number of images) x (image height) x (image width) * (image depth). Upon going via a convolution layer, the picture is abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). The convolution layer convolves this input and passes it to the next layer. Each part of the CNN only gets input according to what it's 'receptive field' is. This allows CNN to have more depth without having a very high number of neurons required to process images if using a shallow architecture. This also allows solving for the vanishing gradient problem seen during backpropagation in traditional neural networks[41]. The figure below describes a CNN network used by YOLO (2016).

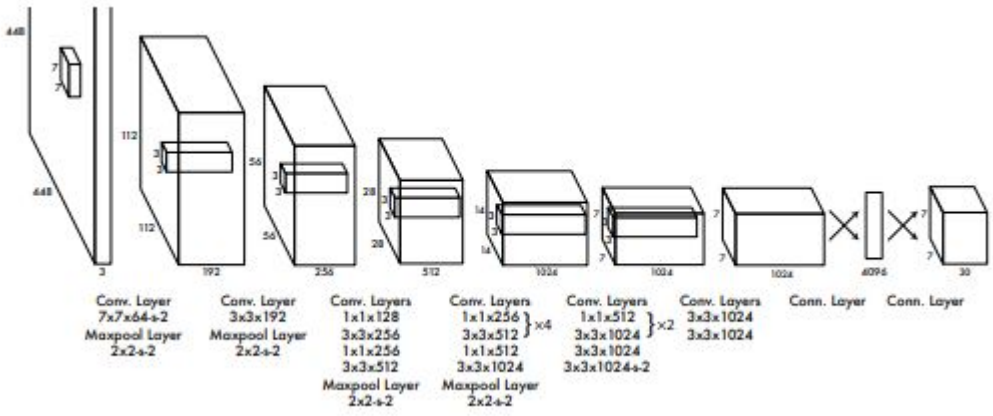

Figure 6: CNN Architecture for YOLO(2016) [29]

The first version of YOLO used the google net model, the YOLOV3 implementation used in this project uses the Dark-Net model [30].

3.3 Kalman Filter

To track moving objects (vehicle) using computer vision with video frames from the drone's camera as the input, the position of the object has to be continuously predicted keeping in mind the weaknesses of the object detection model [YOLO] and by using maximum information that is available from the previous frames [42]. This has to be done to prevent the unnecessary search for the object in the case object detection model fails to provide the position of the object due to occlusion or weakness in its prediction model. Also, we need a way to remove noise, multiple predictions or wrong predictions in a short window. This is where a Kalman filter can be implemented to track the object. The solution is to have equations to update the position of the object assuming a constant velocity model and to factor the process noise. This is possible by implementing a Kalman filter.

[42] defines kalman filter as: "The Kalman filter is the optimal estimator for a linear system for which the noise is zero mean, White and Gaussian, though it will often provide good estimates even if the noise is not Gaussian." Figure 7 beautifully explains the process, courtesy of [43]

The following equations that summarise Kalman filter are also obtained from the derivation by [43]

Let us then call the best estimation of the state of the object as $\hat{x_k}$ and its covariance matrix P_K. To predict the next state at time K, the prediction step can be represented by F_k.

$$\hat{X_k} = F_K * X_k - 1 \tag{9}$$

$$P_k = F_K * (P_k - 1) * F_K^T \tag{10}$$

Kalman gain is given by

$$K = H_K P_k H_K^T (H_k P_K H_K^T + R_K)^{-}1 \tag{11}$$

Predicted measurement mean and covariance are $(\mu_0, \sum_0) = (H_k\hat{x_k}, H_kP_kH_K^T)$ and observed measurement mean and covariance are $(\mu_1, \sum_1) = (\vec{Z_k}, R_k)$. Then the equations for update cycle are,

$$\hat{X_k^{'}} = X^k + K'(\vec{z_k} - H_k\hat{x}_k) \tag{12}$$

$$P_k^{'} = P_k + K'H_KP_K \tag{13}$$

Kalman Gain update is given by

$$K' = P_k * H_k^T(H_kP_kH_k^T + R_K)^-1 \tag{14}$$

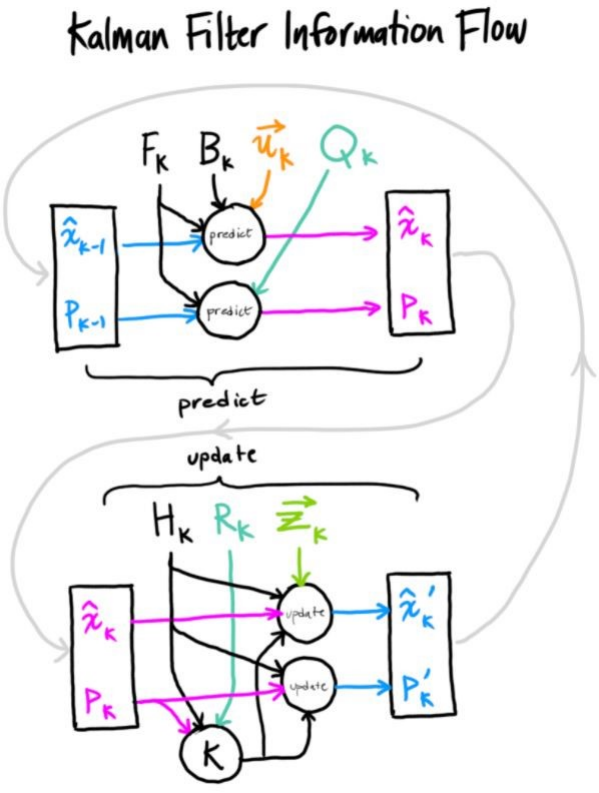

Figure 7: Kalman filter Algorithm [43]

3.4 PID

PID is the abbreviation of a proportional, integral and derivative control. They are widely used for process control in industrial settings. A controller's basic premise is to make the error signal tend to zero; error is the difference between a reference value and the value observed from a sensor. Figure 8 represents the PID process in pictorial form.

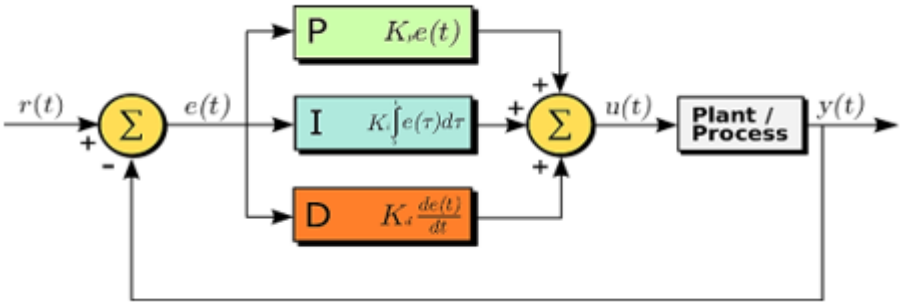

Figure 8: PID Process [44]

$$u(t) = k_p + k_i * e_i + k_d * e_d \tag{15}$$

Here, error E = desired value - current value,
Integral error Ei = Ei + E
Derivative error Ed = E – E previous,
and Kp, Ki and Kd are PID constants.

The following steps can be used to tune the PID constants:

1. Set KP, KI, KD to zero.
2. Increase the value of KP until the response gives a steady oscillation.
3. Increase the value of KD until there are no more oscillations.
4. Repeat 2 and 3 until increasing KD does not reduce oscillations any more.
5. Set KP and KD to the last known good value.
6. Increase KI until the desired value is obtained with minimal oscillations.

4 Methodology - software implementation

4.1 Simulation environment: Parrot Sphinx

A simulation environment was required to train and test the vision-based tracking method developed. Parrot Sphinx was chosen as the environment since it's developed and supported by the parrot team which allows us an easy integration of the simulated model of the Parrot Bebop 2 drone within the environment. It also provides with the option to model and simulate a vehicle actor which is the target chosen for running the experiment.

The main features of a Parrot sphinx environment are that it is backed by Gazebo. Gazebo is an open-source simulation tool that is well integrated with Robotic Operating System ROS which is the middleware that is selected for the implementation of the software. This allows us to operate the drone within the environment from software implemented via ROS. Sphinx also allows to record and visualise flight data which is required for fine-tuning the operation and verify results.

The environment shown below was designed in Gazebo-sphinx. The environment has wind and collision enabled to ensure that the simulation is as close to real as possible. The simulation physics in sphinx is based on Open dynamics engine (ODE) physics engine. The environment is populated with roads, buildings, trees etc to make it visually realistic. The assets for the same were obtained from open-source models available with the sphinx package. Figure 9 is the screengrab of the environment designed for this project. The parrot drone is spawned at the beginning of the simulation at the origin at rest.

The parrot bebop 2 chosen for this project is a lightweight drone, with a 14-megapixel front-facing camera which offers 1080p video recording. However, within the simulation, the video does not render at 1080p due to resource limitation. The camera has a fish-eye lens.

Figure 9: Simulation Environment designed in Gazebo-sphinx

4.2 System Architecture

The implemented control loop is divided into 2 layers, the software layer and the physical hardware layer. In this case, the hardware layer is abstracted to the simulation. The software layer is made up of individual ROS nodes, each with a specific function. A ROS Driver sits between the software layer and the hardware layer and acts as a communication interface for the command signals to be given to the Bebop drone simulation. The driver also provides the software layer with sensor information and video recorded in real-time from the bebop drone. Since the project aims to develop a vision-based control mechanism, only the video data is accepted from the driver.

From the documentation of bebop autonomy " bebop_autonomy is a ROS driver for Parrot Bebop 1.0 and 2.0 drones (quadrocopters), based on Parrot's official ARDroneSDK3. This driver has been developed in Autonomy Lab of Simon Fraser University by Mani Monajjemi and other contributors (List of Contributors). This software is maintained by Sepehr MohaimenianPour (AutonomyLab, Simon Fraser University), Thomas Bamford (Dynamic Systems Lab, University of Toronto) and Tobias Naegeli (Advanced Interactive Technologies Lab, ETH Zürich)."

Traditionally a drone is piloted using a joystick. Commands from an external joystick are sent to the internal controller, which regulates the motors to move the aircraft. Since the project aims to develop autonomous navigation, the control signals are generated by the software and these commands need to be communicated to the drone's internal pilot. The software development kit developed by the manufacturer of the drone allows us to do exactly that, the SDK can be integrated to make web apps/ mobile apps etc to communicate with the drones. Since we are using ROS environment for this project, bebop autonomy which is based on the SDK provided by parrot allows us to control the drone by setting the desired pitch, roll and yaw angles of the aircraft to the angles desired by the navigation software.

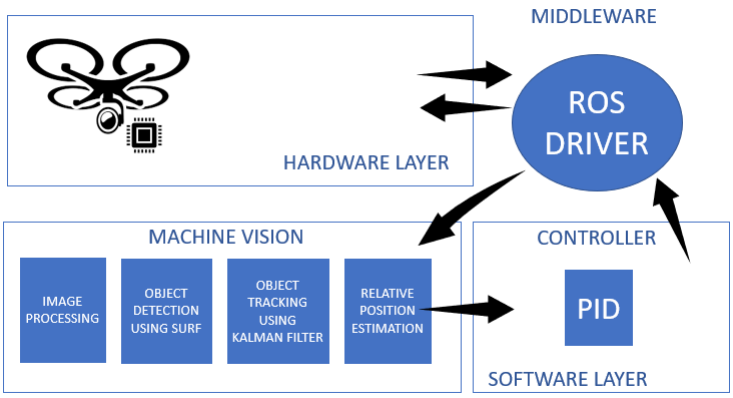

Figure 10: System Architecture [45]

4.3 Implementation

4.3.1 ROS

ROS stands for Robotic Operating System. It is not technically an operating system but a middleware. It has tools and libraries that help the creation of software for robotics tasks. It allows for a system to publish and subscribe messages, therefore different sensors/hardware can talk to software nodes as well as software node can talk to each other. This project uses ROS to simplify the communication of the various parts of the software. The design goal of ROS is to support open source. Therefore various tools and libraries can collaborate to build up the larger application-based software. In the case of this project, we rely on the combination of the open-source implementation of the YOLOV3 object detection neural network, Parrot sphinx Gazebo simulation environment, bebop autonomy ROS driver and the software stack designed for tracking and control as explained in the next subsection.

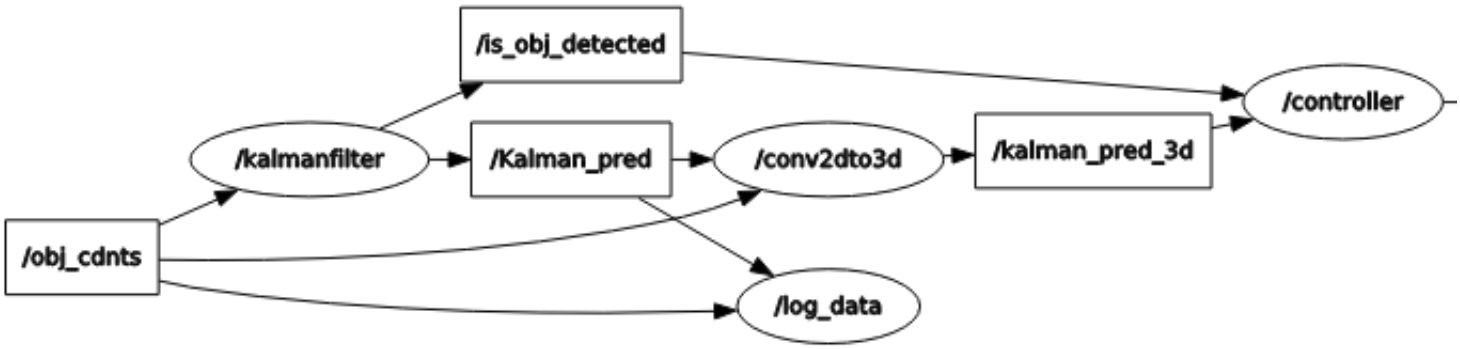

Figure 11: ROS Graph explaining the communication of Nodes

4.3.2 Object detection

Object detection here is performed using the open-source implementation of YOLOV3 available at https://github.com/AntonMu/TrainYourOwnYOLO.git. The first step was to collect sufficient data to train the model on the vehicle to be detected during the experiment. Since there is no dataset available that could be used to train the model to detect the very specific vehicle model which is also a simulated model and not realistic. Therefore, it was decided to generate a synthetic dataset using the simulation environment designed; the vehicle movement was simulated and a video was recorded while changing the simulation view to different heights and viewpoints to collect sufficient good quality data. The dataset was annotated using Microsoft VOTT annotation tool. The model was trained for 3 epochs and its performance on the testing dataset is discussed in the next section. The detector part of the code was modified to communicate with ROS nodes.

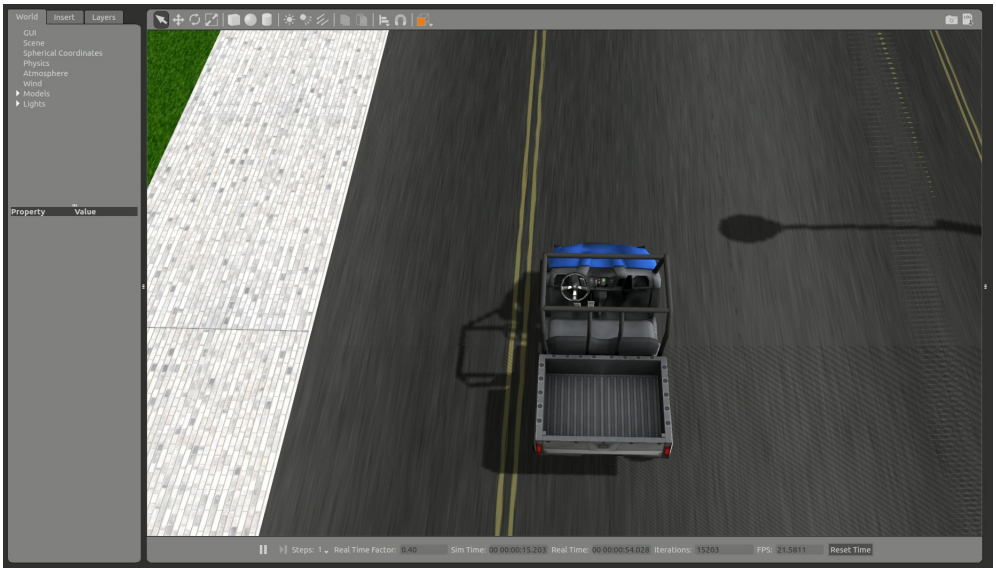

Figure 12: A frame from testing dataset

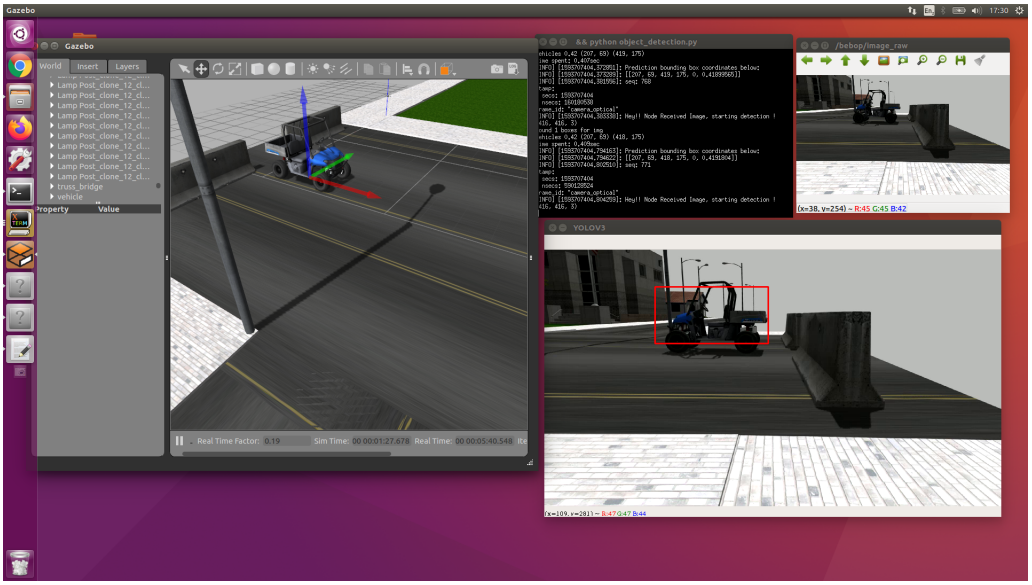

Figure 13: Vehicle detected from the drone video stream

The YOLOV3 detector's implementation was modified to run on the raw video stream received from the bebop drivers ROS node. The detector than outputs a bounding box coordinates of all the detected vehicle in the scene with their confidence levels. This is then used to draw a bounding box around the vehicle using OpenCV as shown in figure 13; it runs a detector on the stream and displays the detected object.

This is done by the object detection ROS node. This node subscribes to the video stream from the ROS driver node, runs the object detection model and publishes the coordinates and the resulting image with the bounding box.

4.3.3 Tracking using Kalman Filter

The Kalman filter node subscribes to the published coordinates data from the object detection node and calculates the centre of the bounding box, assuming that the centre of the bounding box is the centre of mass of the vehicle; the point which that needs to be tracked over time. This is done to overcome false detection or multiple detections that may occur with object detection. Moreover tracking using Kalman filter makes the object detection scheme robust to occlusion.

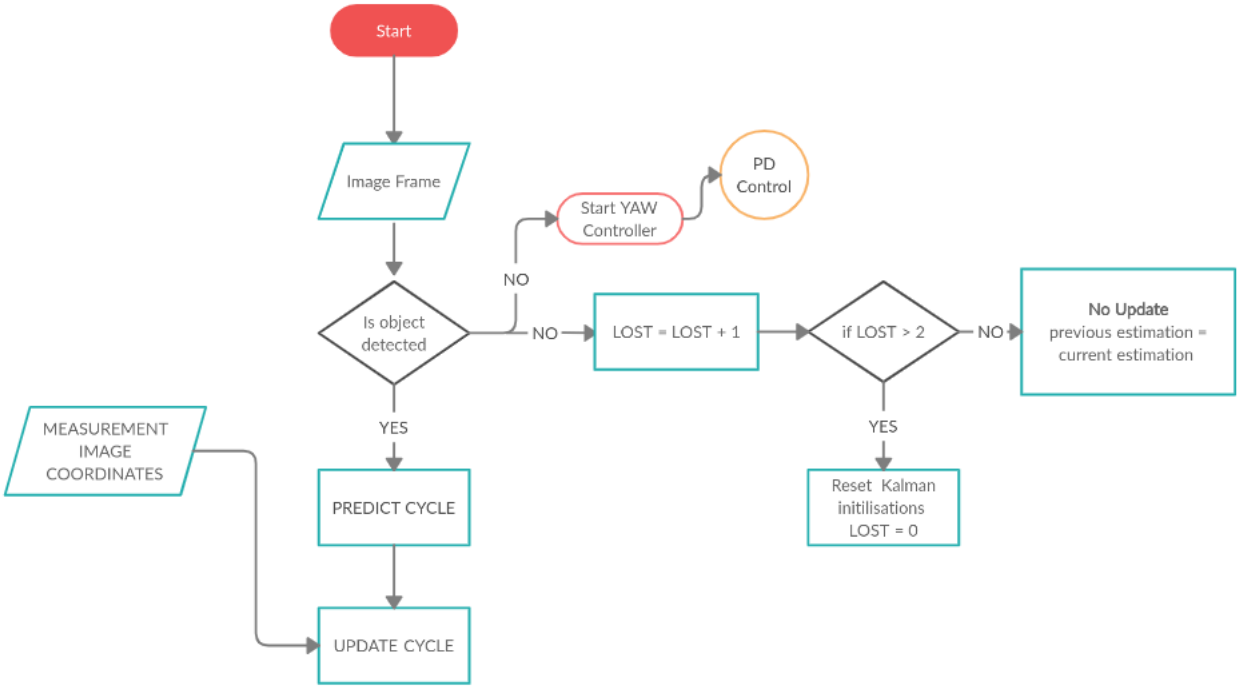

Figure 14: Kalman Filter Flowchart

Assuming a constant velocity model the Kalman filter was designed to track the vehicle in the continuous image sequence. The constant parameter sensor variance was set using object detection bounding box data collected over the experiment and calculate the variance and the rest of the parameters like process noise variance etc were tuned by trial and error. The Kalman filter cycle starts when the object is first detected and it starts to track the centre of the object. If there are no objects detected, and no objects have been detected for over 2 cycles of the filter, the values reset to the initial values prompting the restart of the Kalman filter node, otherwise, within the 2 lost cycles, the filter assumes the last known position as the good value but does not go to the update cycle since no sensor information is available.

4.3.4 Image coordinates to world Coordinates

Assuming that the drone is at the origin, the Kalman filter node publishes the predicted location of the object with respect to the frame of reference of the image. To run the position controller for the drone, we need to find the position of the object with respect to the drone in world coordinates so that appropriate control signal for pitch, roll and yaw angles could be generated. This is termed as an inverse projection problem.

We have the position of the drone (x image, y image) which needs to be inversely projected to its actual position (x world, y world, z world), inverse projection because the camera had performed a projection of the world coordinates to the image plane coordinates to form the image and now we need to do the opposite.

It is a difficult problem to solve due to the loss of depth information during perspective projection. This can be solved using a stereo camera or depth sensor, but since we have to rely on a monocular camera, another approach has to be implemented. From equation 7 and 8 discussed in the previous section, we know that the real-world position of the object x world and Z world are normalised to the depth y world. From the approach discussed by [19], this project assumes from experimentation that the diameter of a box enclosing the target vehicle should approximately be 6m in real-world terms. Hence the actual diameter of the bounding box can be calculated using the focal length of the camera obtained as given in the equation below where f is the focal length of the camera, d real is the approximate diameter of the bounding box in the real world and d image is the calculated diameter of the bounding box on the detection.

$$d = \frac{f * d\_real}{d\_image} \tag{16}$$

Now the depth can be used to calculate the X world and Z world position normalised in-depth Y world. This node subscribes to the Kalman filter node and publishes the position of the object in 3d world which is subscribed by the position control node.

4.3.5 Position control: PD controller

If the object is detected, the node subscribes to the real world position message with respect to the drone. Since the bebop's internal controller requires pitch, yaw and roll angles to be set to move it to the desired position, two PD controllers are designed to sit hierarchically on top of the internal controller. The strategy adopted here is that if no object is found a yaw signal is generated to rotate the drone in search for the desired object in world space, as soon as the object is found the two hierarchical PD controllers for controlling the roll and pitch are run assuming the drone's position as the origin and the object position as the distance to be covered in roll and pitch directions respectively, The altitude is fixed at takeoff, therefore, another controller for altitude was not implemented. The flow chart below explains the process. The PD parameters were set as discussed in the previous section.

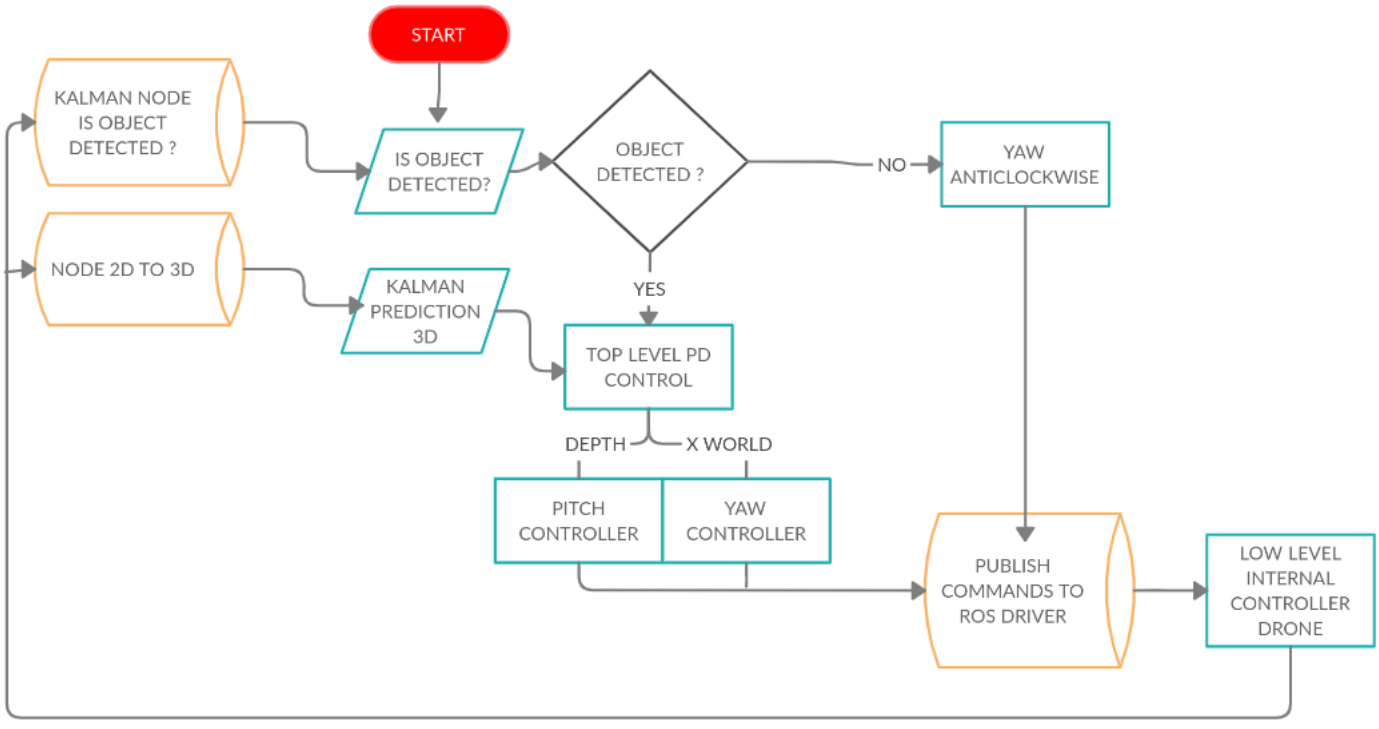

Figure 15: Position Control

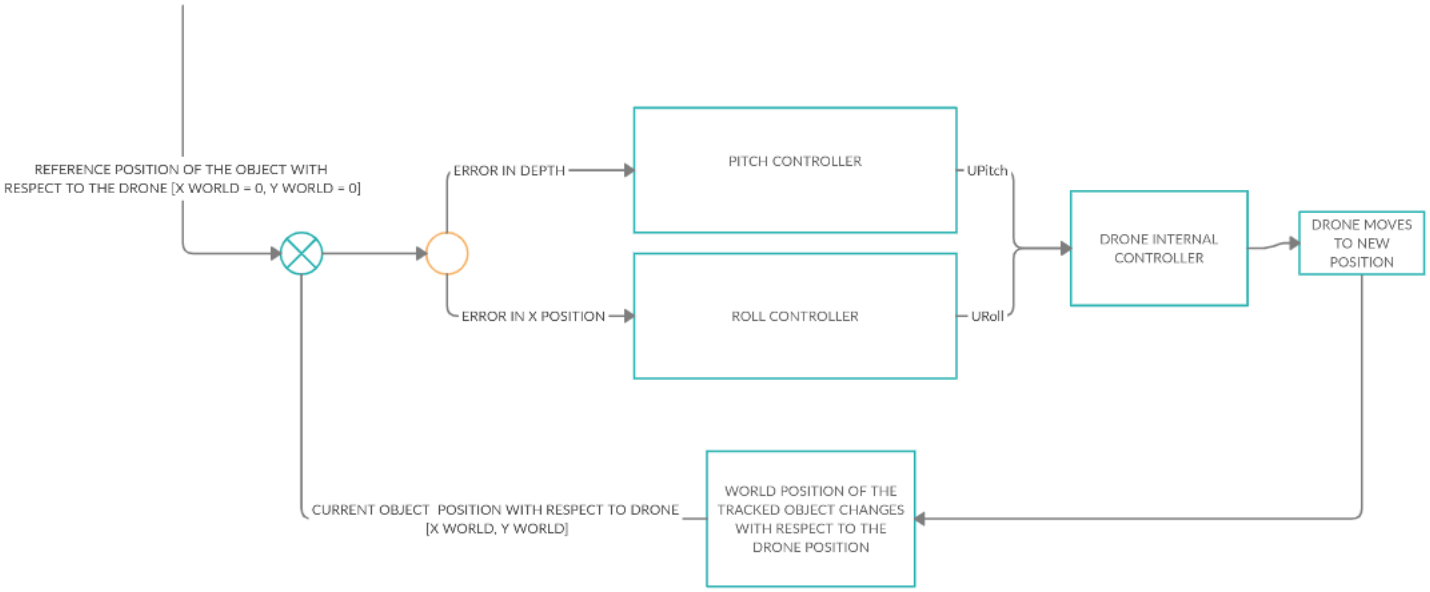

Figure 16: Control Diagram

5 Results

5.1 YOLOV3 Performance

The following section deals with the performance of the YOLOV3 object detection model on the synthetic data. The synthetic data was generated using simulation and 3 epochs of the training were run. Similarly, a testing dataset of 494 images was created. On testing the detector on the testing dataset the following precision-recall curve was obtained as shown in figure 17.

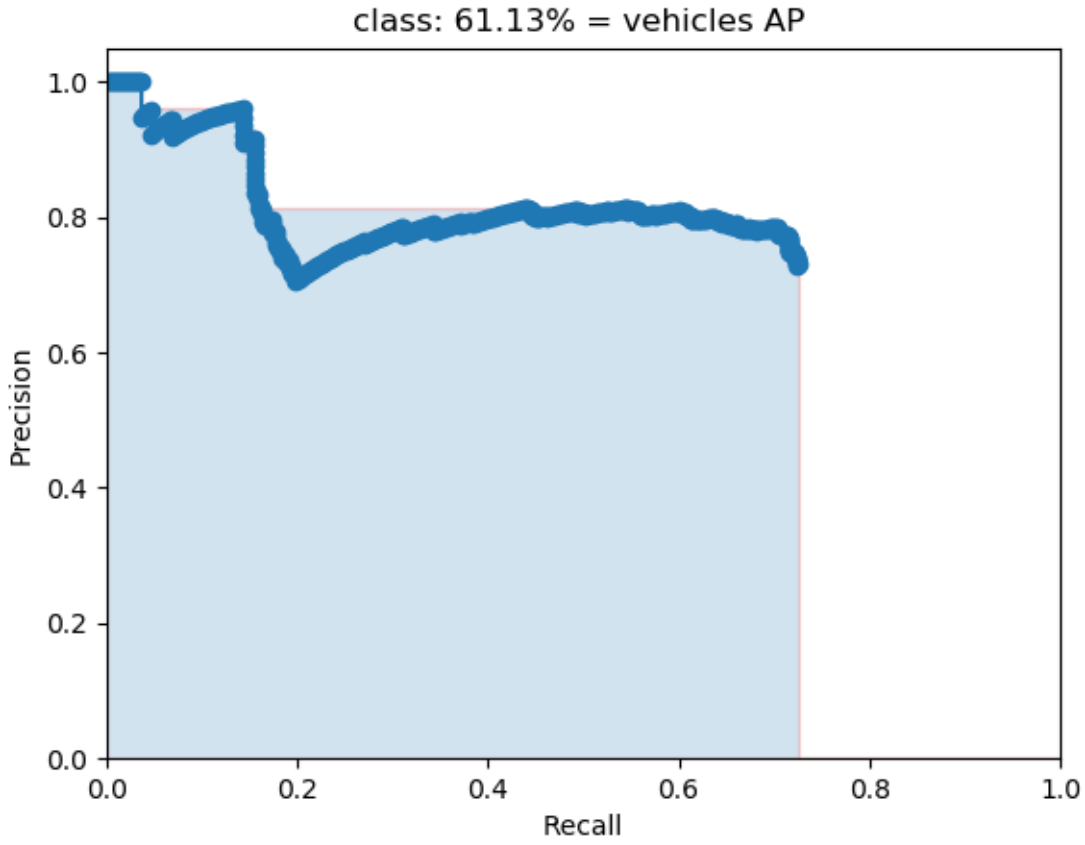

Figure 17: Precision-Recall Curve

The intersection of Union of IOU refers to the ratio of $\frac{area\_of\_overlap}{area\_of\_union}$ of the bounding box that is the ground truth and the bounding box that has been predicted. Usually, an IOU of > 0.5 and correct classification is considered true positive. Similarly, an IOU of < 0.5 will be considered false positive or if a wrong classification is made. A false negative is when there is no detection or IOU > 0.5 but wrong classification. Precision and Recall can be calculated using the true positive, false positive and false negative.

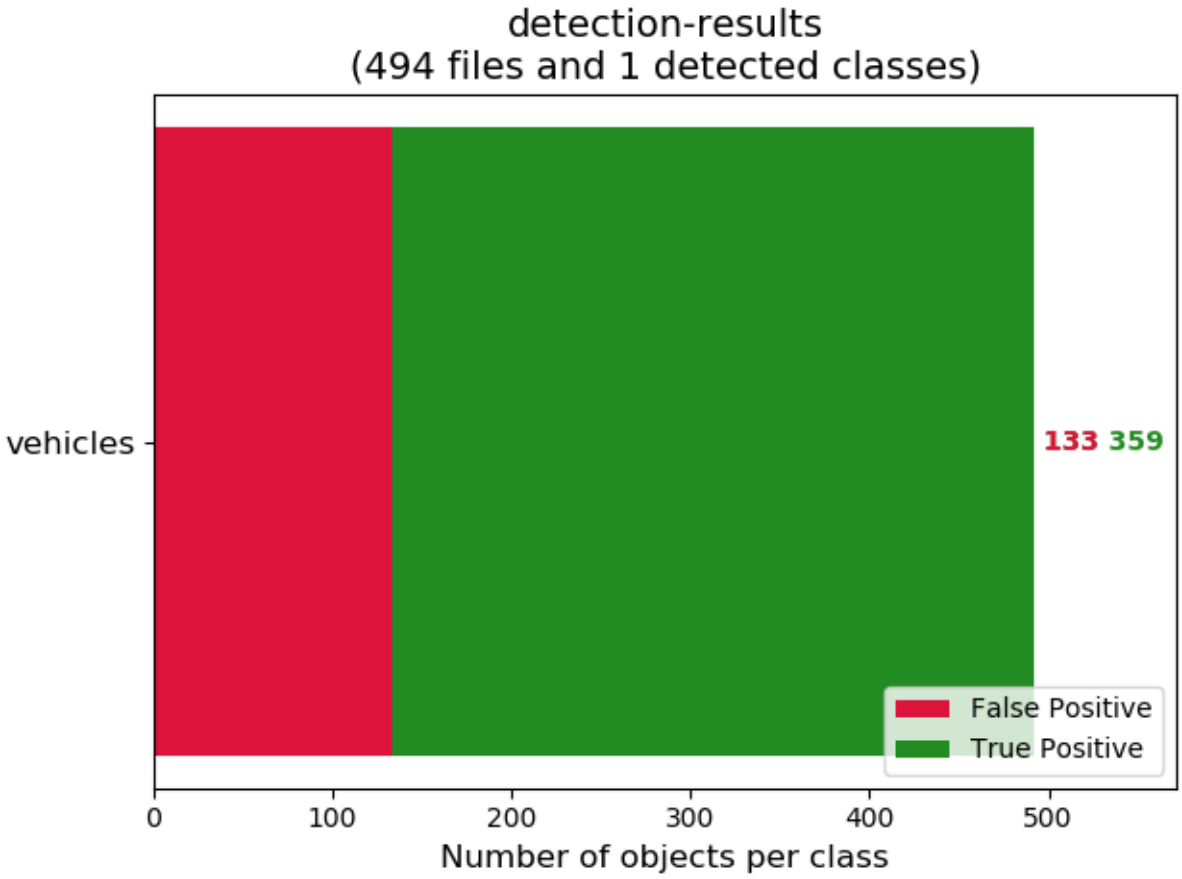

Figure 18: Detection Results

The precision-recall curve highlights the trade-off between precision and recall for different threshold values. A perfect object detection model will have the precision 1 and recall 0 for all thresholds and a very bad object detection model will have the recall of 1 and precision of 0 for all thresholds. The idea is to set a balance between precision and recall. From the Area under the Curve, we can see that the object detection model has fairly good precision and recall balance.

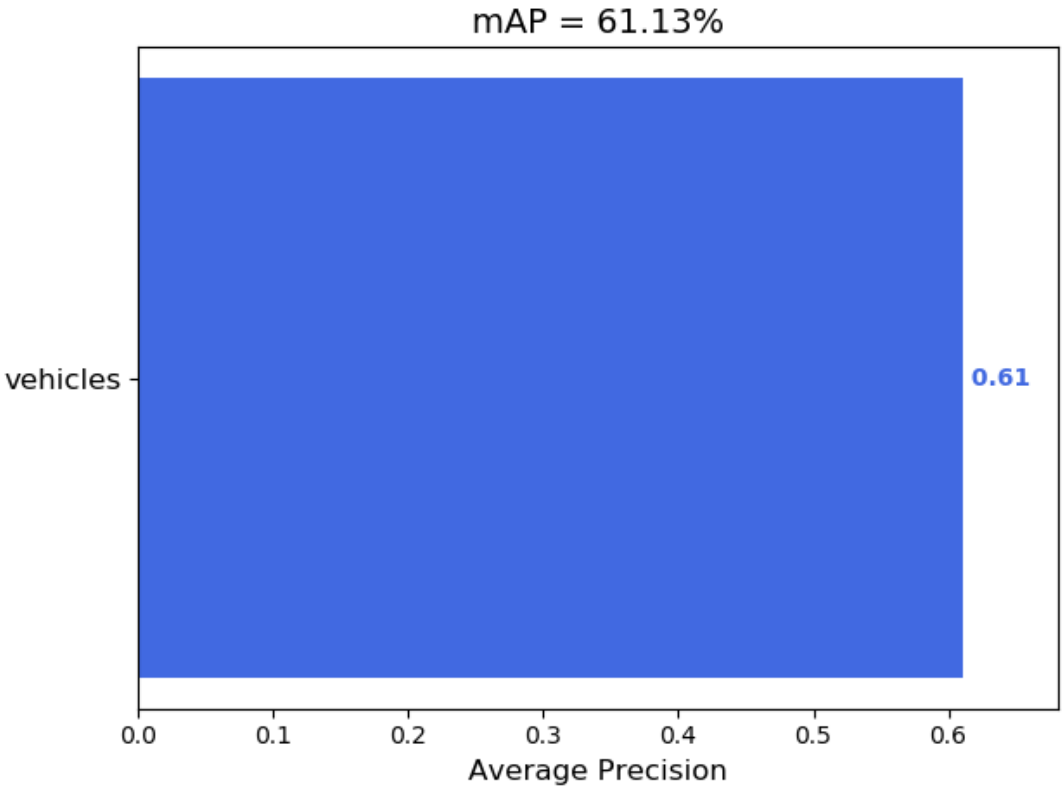

Figure 19:

From figure 18 we can see that the object detection model has a good rate of true positives in comparison to false positives. Average precision can be calculated using the area under the curve from the precision-recall curve. The mean of average precision for all classes is the mean average precision or MAP value. MAP is widely used to judge the performance of an object detection model. The plot below shows the MAP of the object detection model on the synthetic training data. Figure 19 shows the MAP of the model used.

5.2 Tracking Results

The following subsection discusses the results of the experiments conducted to verify the performance of the vision-based tracking algorithm implemented by this project.

5.2.1 Experiment 1

The following section shares the results of the performance of the tracking algorithm developed. In this experiment, the vehicle's path (ground truth) was along a straight section of the road. Figure 20 illustrates the performance of the Kalman filter for x coordinate and fig 21 for the y coordinate.

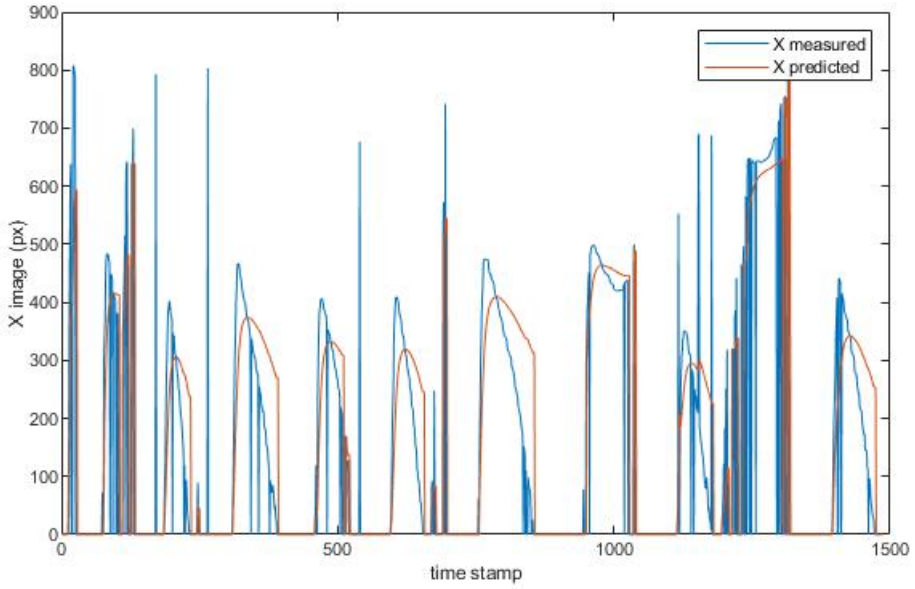

Figure 20: Kalman Filter prediction vs measure for the x coordinate

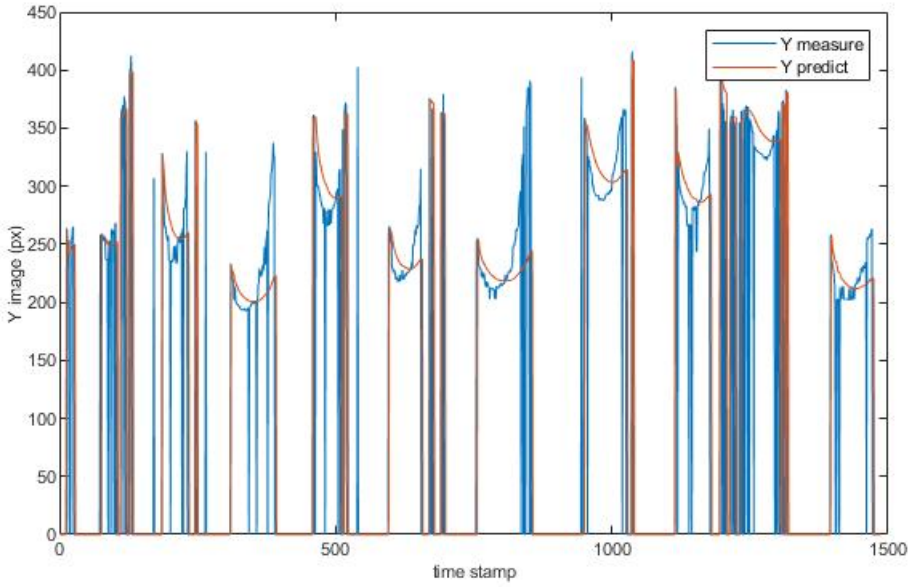

Figure 21: Kalman Filter prediction vs measure for the y coordinate

Figures 22 and 23 show the accuracy of tracking the position of the vehicle in the pitch and roll respectively.

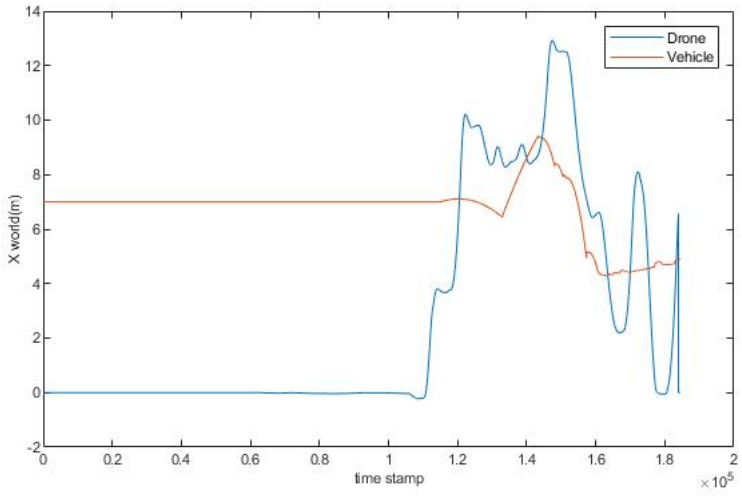

Figure 22: Drone position vs Vehicle position in x direction

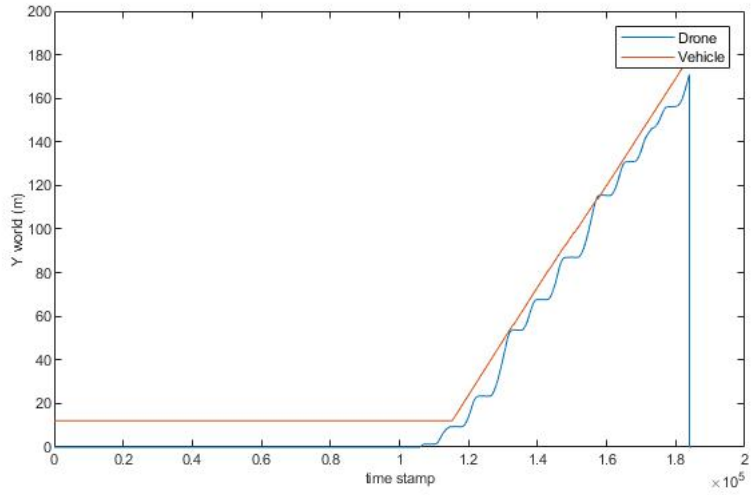

Figure 23: Drone position vs Vehicle position in y direction

Figure 24 is the representation of the paths taken by the vehicle and the Drone as captured by their respective GPS. The figure provides an insight into the performance of the tracking implemented. Since the actual experiment was carried out in a simulated environment, the map represented in the image below is a virtual map and does not correspond to a real location.

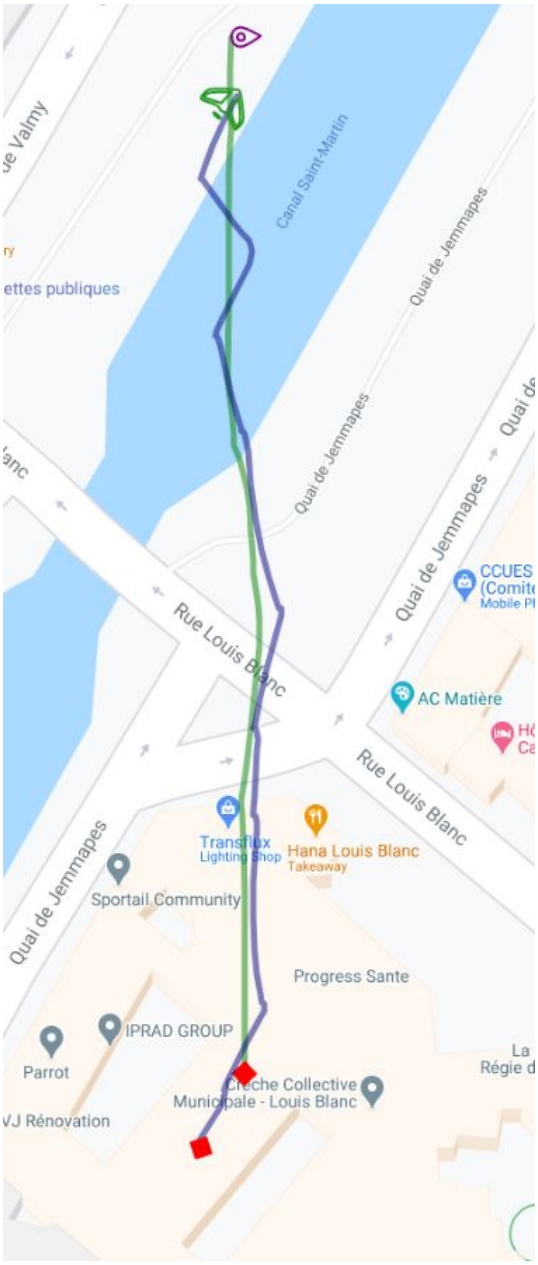

Figure 24: GPS position of Drone vs Vehicle

5.2.2 Experiment 2

In this experiment, the vehicle path is completing a square track (ground truth). Figure 25 and 26 illustrate the Kalman filter predictions.

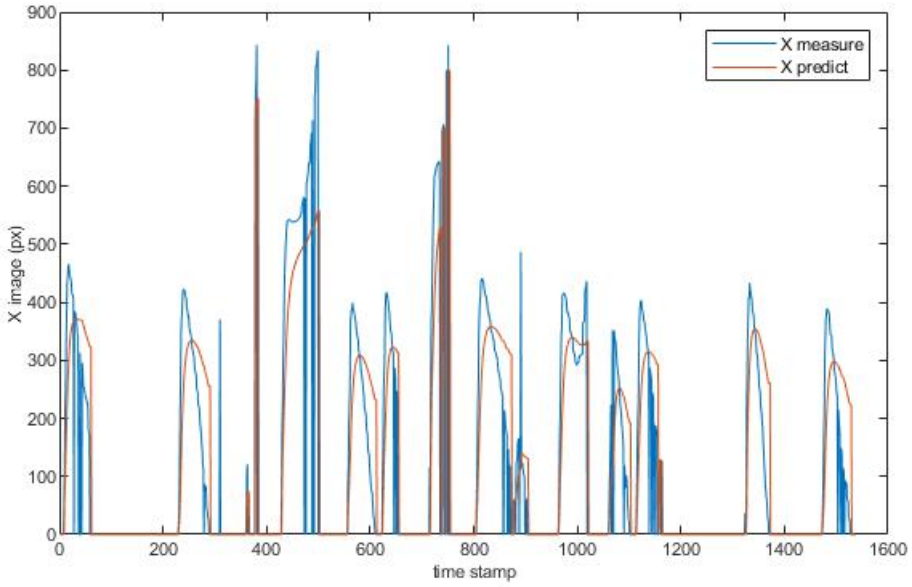

Figure 25: Kalman Filter prediction vs measure for the x coordinate

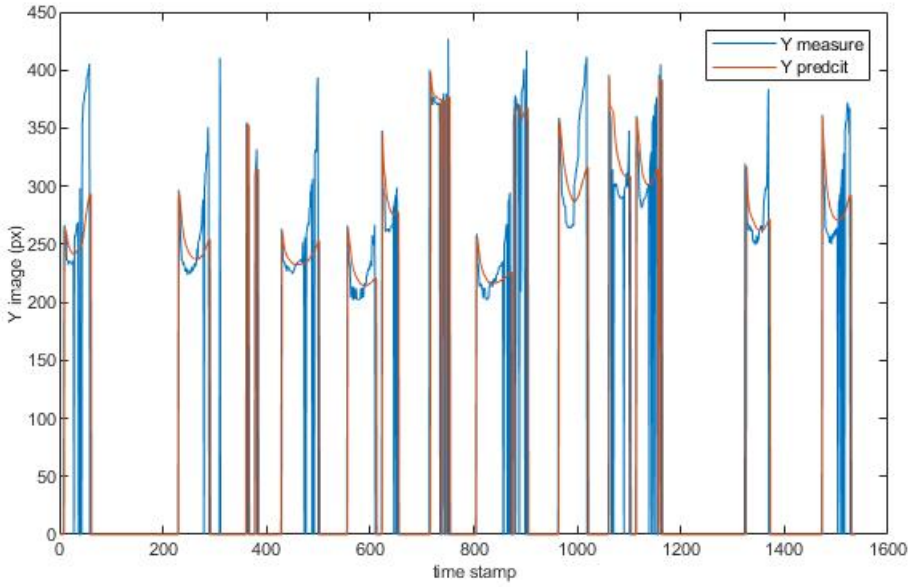

Figure 26: Kalman Filter prediction vs measure for the y coordinate

Figures 27 and 28 show the accuracy of tracking the vehicle's position in the pitch and roll respectively.

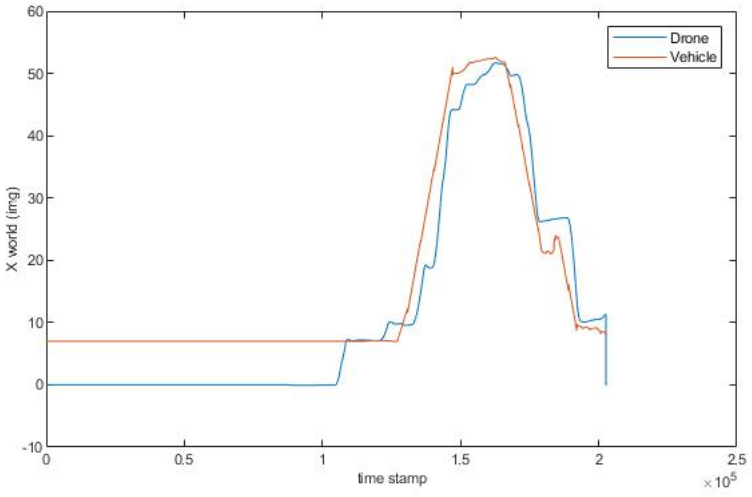

Figure 27: Drone position vs Vehicle position in x direction

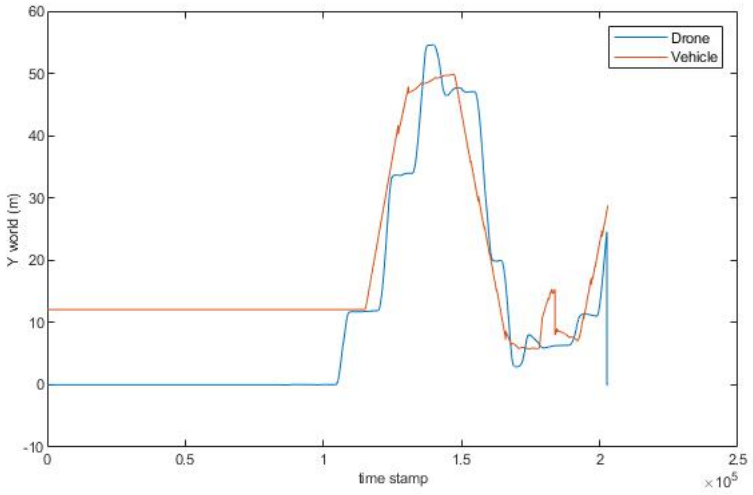

Figure 28: Drone position vs Vehicle position in y direction

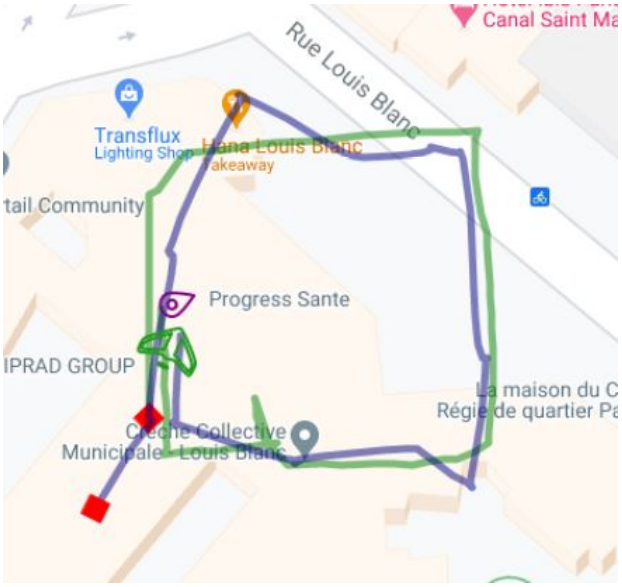

Figure 29: GPS position of Drone vs Vehicle

6 Analysis

From the results, it can be concluded that the tracking method implemented can be used to successfully track the moving vehicle. However, the method still requires a lot of improvement to be able to adapt to real-world tasks.

One of the limitations of the implementation noted is that it does not take into account the velocity of the vehicle being tracked. Since the Kalman filter model used is a constant velocity model, therefore the varying velocity of the vehicle is not taken into consideration which is very likely to hinder its performance in real-life scenarios. In the simulation, it was observed that when the distance between the drone and vehicle is large, the drone speeds up and exceeds the position of the target, and then has to wait for the target to catch up to the drone's position and once again come under drones field of view. This issue can be solved by using the constant acceleration model.

From the results, it is noted that the accuracy of the roll controller is not up to the mark. From figure 22 it can be observed that the error margin between the drones position and the vehicle position is higher than what could be tolerable in real-life applications. This error difference has implications which can be seen in the tracking performance; the implication is that the object keeps on going out of the view of the vehicle which causes the Kalman filter to reset since the Kalman filter takes some cycles to converge to the correct predictions, the controller receives wrong position targets.

Another issue highlighted by the results is that the control signal is possibly oscillating. This is a common issue with the PD controller if it is not very finely tuned. Also, the implementation discussed in this thesis does not separately implement a yaw controller as well as an altitude controller. It is assumed that yaw needs to be corrected in case the object is not found. A hard set mechanism is implemented to rotate the drone anticlockwise until the vehicle is found and then fix the yaw position. This can be improved by adding another yaw controller which can correct the yaw either based on centring the found object to the image frame or in cases when the object is not found and a local search has to be made the controller can be implemented to remember the last known good position and predict the most probable current position and set the Euler angle for yaw accordingly.

As far as the altitude controller goes, since external sensors were not to be relied on, a solution for estimating the current altitude from visual data needs to researched and implemented. This will also help with the issue of losing the object and yawing to find the object instead of raising the altitude and expand the field of view of the camera. This issue can be observed in figure 23 where it can be seen that the vehicle's position in y is straight slope whereas the drones increase in y position step like this is due to the drone losing the vehicle and waiting at a point to search for the target.

Another issue with the implementation was caused by the simulation environment. Due to the simulation running on a regular laptop and the simulation application relying on single thread CPU implementation, the performance of the simulation had considerably slowed down. The real-time factor of the simulation was less than 0.34, where a rtf of 1 means that the sim time and real-time is the same and a rtf of zero means that the simulation is not running/paused or crashed. This caused considerable lag. It also caused issues with running the vehicle actor on time and the correct path. Due to lack of resources for the simulation, considerable lags caused the vehicle actor to miss way-points and move in random directions, therefore requiring manual intervention by rotating/translating the vehicle to keep it on track. The issue also caused lowered frame rates of the input video to the object detection model, therefore, causing unexpected behaviour by the drone.

The last issue observed is that there is no consideration for obstacle avoidance. Flying autonomously for some applications may require that the UAV flies at low heights, where obstacles such as trees, tall buildings, lamp posts etc may be a hindrance to the drone and may lead to crashes. Obstacle avoidance can also be implemented using vision-based solutions.

7 Conclusions and future work

The objective of this thesis was to present a vision-based system to detect and track a moving object using a monocular camera. UAV needed to rely on reactive navigation, that reacts to the information obtained from the surrounding. Several methods to do this were discussed; it was found that active sensors like radar, GPS or passive sensors like Kinect and cameras can be used for UAV navigation. Since active sensors are disadvantaged on account of their power consumption and weight and also the UAV needs to perform in areas where GPS is not likely to work, a solution was proposed to use monocular cameras as sensors and process the input using machine vision techniques to implement a robust vision-based tracking mechanism aided with Kalman filter and PID controllers.

Several object detection methods were discussed including SIFT, SURF etc. It was found that the convolutional neural networks worked the best in terms of speed and accuracy over more traditional methods of object detection like Haar-cascade etc. Moreover, single-step detectors like YOLO and SSD were found to work faster than two-step detectors like fast-RCNN. YOLOV3 was chosen as the CNN for detecting the object for this project. A synthetic dataset from the simulation model was generated to train the model for detection. The object detection model gave a mean average precision of 61.13

The process described herein is that an object detection model is trained to detect the class for the object of interest which in this case is a vehicle. The project was implemented in a simulation environment with the help of ROS driver bebop autonomy and the parrot sphinx simulation environment based on the gazebo. The ROS drive allowed the communication of the control node to control the drone in the sphinx environment. The position of the detected object is then passed as a ROS message to the Kalman filter node, where the Kalman filter tracks the object by predicting its real position which may not be the same as the predicted position due to various factors such as occlusion, false detections etc. The predicted position is given to the node that converts the position of the object in the image to its real-world position with respect to the drone. This is done by assuming a pinhole camera model and using the real-world measurements, focal length and camera intrinsic matrix to find out the depth and thereafter the x position of the target. Once the position of the target is known, a simple PD controller is implemented to reach the position of the target.

The results indicated that the method works but several limitations were also observed. They may not be entirely due to an inherent weakness of the method but rather due to the limitation of the implementation in a simulated environment. Several limitations were found, such as the PD controller needs to be tuned a lot better to remove oscillations. The implementation needs extensive testing in real-world scenarios; a controller for yaw and altitude adjustment needs to be implemented.

One of the objectives that have not been implemented due to issues with the simulation was to implement a got to GPS location search method. The idea was to obtain the GPS location of the target for a brief period so that the drone can move to the general area of the coordinate (global search) and then searches locally for the object starting the tracking process once the object is found. This remains to be implemented in the future. It is suggested that in going forward the Kalman filter be designed considering constant acceleration model instead of the constant velocity model so that the drone takes into consideration the speed of the target and change its velocity accordingly, and the values of the Kalman parameters be tuned base on real data collected. Object detection model needs to train on more classes and the training data needs to be collected with more diverse conditions and obstacle avoidance needs to be incorporated. Further, SSD object detection models can be used to replace the YOLOV3 because it is much faster and accurate than YOLO. It is also suggested that fuzzy logic be used as a controller replacing the PD controller.

References

[1] P. Corke, 'Mobile Robot vehicles: Flying Robots', in *Robotics, Vision and Control*, 2011, pp. 114–118.

[2] M. Simon, *What is a Robot?*, 2017. [Online]. Available: https://www.wired.com/story/what-is-a-robot/ (visited on 18/08/2020).

[3] J. Dadwal and B. Sharma, 'Use of Robotic System for Creator's Solution in Completing the Assigned Task and Dealing with the Physics of the Environment Around It', *SSRN Electronic Journal*, 2018, ISSN: 1556-5068. DOI: 10.2139/ssrn.2927206.

[4] A. Avitan, *The difference between UAV, UAS and Autonomous drones*, 2019. [Online]. Available: https://percepto.co/what-are-the-differences-between-uav-uas-and-autonomous-drones/ (visited on 18/08/2020).

[5] G. M. Hoffmann, H. Huang, S. L. Waslander and C. J. Tomlin, 'Quadrotor helicopter flight dynamics and control: Theory and experiment', in *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2007*, 2007, ISBN: 1563479044. DOI: 10.2514/6.2007-6461.

[6] *Drones need to be encouraged, and people protected*, 2019. [Online]. Available: outline.com/https://www.economist.com/leaders/2019/01/26/drones-need-to-be-encouraged-and-people-protected (visited on 18/08/2020).

[7] *ARDroneSDK3 API Reference*. [Online]. Available: https://developer.parrot.com/docs/SDK3/ (visited on 18/08/2020).

[8] Hunini, *Parrot bebop 2*. [Online]. Available: https://commons.wikimedia.org/wiki/File:JGSDF_Parrot_Bebop_2_Power_Drone_at_Camp_Akeno_November_4,_2018.jpg (visited on 18/08/2020).

[9] A. Ollero and L. Merino, 'Control and perception techniques for aerial robotics', *Annual Reviews in Control*, 2004, ISSN: 13675788. DOI: 10.1016/j.arcontrol.2004.05.003.

[10] E. Feron and E. Johnson, 'Aerial robotics', in. Jan. 2008, pp. 1009–1029. DOI: 10.1007/978-3-540-30301-5_45.

[11] Y. Lu, Z. Xue, G.-S. Xia and L. Zhang, 'A survey on vision-based uav navigation', *Geo-spatial Information Science*, vol. 21, no. 1, pp. 21–32, 2018. DOI: 10.1080/10095020.2017.1420509.

[12] B. P. Tice, 'Unmanned Aerial Vehicles-The Force Mulitplier of the 1990's', *Airpower Journal*, 1991.

[13] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. 2012, p. 226, ISBN: 9780691149219.

[14] G. R. Bradski and A. Kaehler, *Learning OpenCV - computer vision with the OpenCV library: software that sees*. 2008, p. 2, ISBN: 978-0-596-51613-0.

[15] S. Li and D. Y. Yeung, 'Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models', in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 2017.

[16] M. Sanfourche, J. Delaune, G. Le Besnerais, H. de Plinval, J. Israel, P. Cornic, A. Treil, Y. Watanabe and A. Plyer, 'Perception for UAV : Vision-Based Navigation and Environment Modeling', *Aerospace Lab*, 2012.

[17] J. Engel, J. Sturm and D. Cremers, 'Camera-based navigation of a low-cost quadcopter', in *IEEE International Conference on Intelligent Robots and Systems*, 2012, ISBN: 9781467317375. DOI: 10.1109/IROS.2012.6385458.

[18] K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller and M. Suppa, 'Stereo vision based indoor/outdoor navigation for flying robots', in *IEEE International Conference on Intelligent Robots and Systems*, 2013, ISBN: 9781467363587. DOI: 10.1109/IROS.2013.6696922.

[19] D. A. Mercado-Ravell, P. Castillo and R. Lozano, 'Visual detection and tracking with UAVs, following a mobile object', *Advanced Robotics*, 2019, ISSN: 15685535. DOI: 10.1080/01691864.2019.1596834.

[20] T. Luukkonen, 'Modelling and Ccontrol of Quadcopter', *Journal of the American Society for Mass Spectrometry*, 2011, ISSN: 1879-1123. DOI: 10.1007/s13361-011-0148-2.

[21] J. Thomas, J. Welde, G. Loianno, K. Daniilidis and V. Kumar, 'Autonomous Flight for Detection, Localization, and Tracking of Moving Targets with a Small Quadrotor', *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1762–1769, 2017, ISSN: 23773766. DOI: 10.1109/LRA.2017.2702198.

[22] A. Olsen, *4.1 Arranging Cameras for Stereo Photography*. [Online]. Available: https://aaronolsen.github.io/tutorials/stereomorph/arranging%7B%5C_%7Dcameras%7B%5C_%7Dphotography.html (visited on 20/08/2020).

[23] S. Al Habsi, M. Shehada, M. Abdoon, A. Mashood and H. Noura, 'Integration of a Vicon camera system for indoor flight of a Parrot AR Drone', *ISMA 2015 - 10th International Symposium on Mechatronics and its Applications*, pp. 1–6, 2016. DOI: 10.1109/ISMA.2015.7373476.

[24] J. Anitha and S. Deepa, 'Tracking and Recognition of Objects using SURF Descriptor and Harris Corner Detection', *International Journal of Current Engineering and Technology*, vol. 4, no. 2, pp. 775–778, 2014.

[25] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan and J. Walsh, 'Deep Learning vs. Traditional Computer Vision', in *Advances in Intelligent Systems and Computing*, 2020, ISBN: 9783030177942. DOI: 10.1007/978-3-030-17795-9_10. arXiv: 1910.13796.

[26] I. Ozhiganov, *Convolutional Neural Networks vs. Cascade Classifiers for Object Detection*, 2017. [Online]. Available: https://dzone.com/articles/cnn-vs-cascade-classifiers-for-object-detection (visited on 21/08/2020).

[27] P. Useche Murillo, R. Moreno and J. Pinzón Arenas, 'Comparison between cnn and haar classifiers for surgical instrumentation classification', *Contemporary Engineering Sciences*, vol. 10, pp. 1351–1363, Dec. 2017. DOI: 10.12988/ces.2017.711157.

[28] Y. Bouafia and L. Guezouli, 'An overview of deep learning-based object detection methods', Mar. 2019.

[29] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, 'You only look once: Unified, real-time object detection', in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.91.

[30] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. arXiv: 1804.02767 [cs.CV].

[31] A. Sonawane, *YOLOv3: A Huge Improvement*. [Online]. Available: https://medium.com/@anand%7B%5C_%7Dsonawane/yolo3-a-huge-improvement-2bc4e6fc44c5 (visited on 21/08/2020).

[32] H. Thakore, 'Moving Object Tracking Using Kalman Filter', *International Journal of Computer Science and Mobile Computing*, vol. 2, no. 4, pp. 326–332, 2013.

[33] A. Alahi, *Lecture 5 Visual Tracking q*, 2015. [Online]. Available: http://vision.stanford.edu/teaching/cs231b%7B%5C_%7Dspring1415/slides/lectureTracking.pdf (visited on 21/08/2020).

[34] R. E. Kalman, 'A new approach to linear filtering and prediction problems', *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[35] M. Marrón, J. C. García, M. A. Sotelo, M. Cabello, D. Pizarro, F. Huerta and J. Cerro, 'Comparing a Kalman filter and a particle filter in a multiple objects tracking application', in *2007 IEEE International Symposium on Intelligent Signal Processing, WISP*, 2007, ISBN: 142440830X. DOI: 10.1109/WISP.2007.4447520.

[36] K. M. Thu and A. I. Gavrilov, 'Designing and Modeling of Quadcopter Control System Using L1 Adaptive Control', in *Procedia Computer Science*, 2017. DOI: 10.1016/j.procs.2017.01.046.

[37] P. Castillo, R. Lozano and A. E. Dzul, *Modelling and control of mini-flying machines*, 2006. DOI: 10.1109/MCS.2006.1636317.

[38] K. Hata and S. Savarese, *CS231A Course Notes 1: Camera Models*. [Online]. Available: https://web.stanford.edu/class/cs231a/course%7B%5C_%7Dnotes/01-camera-models.pdf (visited on 21/08/2020).

[39] *Camera Calibration and 3D Reconstruction*. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera%7B%5C_%7Dcalibration%7B%5C_%7Dand%7B%5C_%7D3d%7B%5C_%7Dreconstruction.html (visited on 21/08/2020).

[40] J. Swiezewski and M. Maj, *YOLO Algorithm and YOLO Object Detection: An Introduction*, 2020. [Online]. Available: https://appsilon.com/object-detection-yolo-algorithm/ (visited on 20/08/2020).

[41] H. H. Aghdam and E. J. Heravi, *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. 2017, ISBN: 978-3-319-57550-6. DOI: 10.1007/978-3-319-57550-6.

[42] E.R.Davies, *Computer Vision*, 5th ed. Elseiver, 2018, ISBN: 9780128092842.

[43] *How a Kalman filter works, in pictures*, 2015. [Online]. Available: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ (visited on 20/08/2020).

[44] A. Urquizo, *PID Controller Overview*. [Online]. Available: http://commons.wikimedia.org/wiki/File:PID.svg (visited on 09/01/2020).

[45] Rohan Dwivedi, 'Project Proposal', Tech. Rep., 2020.

```python
# importing OpenCV
import sys
sys.path.remove('/opt/ros/kinetic/lib/python2.7/dist-packages')
import cv2
sys.path.append('/opt/ros/kinetic/lib/python2.7/dist-packages')

# importing ros packages
import rospy
from sensor_msgs.msg import Image
from rospy_tutorials.msg import Floats
from rospy.numpy_msg import numpy_msg
from cv_bridge import CvBridge, CvBridgeError

# importing packages from TrainYourOwnYOLO Package #
sys.path.append('../../TrainYourOwnYOLO/2_Training/src/')
from keras_yolo3.yolo import YOLO, detect_video
sys.path.append('../../TrainYourOwnYOLO/Utils/')
from utils import load_extractor_model, load_features, parse_input, detect_object

# importing misc packages
#from PIL import Image
import numpy as np
import pandas as pd

#import weights and classes for the model
model_weights = "../../TrainYourOwnYOLO/Data/Model_Weights/trained_weights_final.h5"
model_classes = "../../TrainYourOwnYOLO/Data/Model_Weights/data_classes.txt"
anchors_path  = "../../TrainYourOwnYOLO/2_Training/src/keras_yolo3/model_data/yolo_anchors.txt"

def img_callback(img_msg):
 '''
 ->subscribe to the bebop/img_raw topic
 ->publish img with bounding box draw around the detect objects(s)
 ->publish bouding box coordinate data
 '''
 rospy.loginfo(img_msg.header)
 try:
  cv_image = bridge.imgmsg_to_cv2(img_msg, "bgr8") # convert image in ros msg format to opencv format
 except CvBridgeError as e:
  rospy.logerr("CvBridge Error: {0}".format(e))

 rospy.loginfo("Hey!! Node Received Image, starting detection !")
 pred = detector(cv_image) # call detector function
 pred_image = drawBox(cv_image,pred)
 pub_image.publish(bridge.cv2_to_imgmsg(pred_image, encoding="passthrough"))
 data = np.array(pred,dtype = np.float32) # numpy.array.flatten() converts 2d array to 1d array since numpy_msg format doesnt support 2d arrays.
 pub_data.publish(data.flatten())

def drawBox (cv_img,pred):
    '''
    ->draws bounding box over the predictions
    '''
 # pred coordinates format xmin 0 ,ymin 1,xmax 2,ymax 3
 # bounding box start point top left corner, end point bottom right corner

    detections = len(pred)
    for i in range (0, detections):
        cv_img = cv2.rectangle(cv_img,(pred [i][0],pred[i][3]),(pred[i][2],pred[i][1]),(0,0,255),2)
    return cv_img

def detector(img):
    '''
    ->takes input image from the callback
    ->YOLOV3 detects the object in the image
    ->returns the coordinates of the prediction
    '''
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.medianBlur(img,5) # remove noise from the image to reduce false detections
    pred, img = detect_object(yolo, img, save_img = False) # call the YOLOV3 detector
    rospy.loginfo('Prediction bounding box coordinates below: ')
    rospy.loginfo(pred)
    return pred

if __name__ == '__main__':

# define YOLO detector
 yolo = YOLO(
```

```
    **{
    "model_path": model_weights,
    "anchors_path": anchors_path,
    "classes_path": model_classes,
    "score": 0.25,
    "gpu_num": 1,
    "model_image_size": (416, 416),
        }
)

rospy.init_node("object_detection")
rospy.loginfo("Detection Node started")
bridge = CvBridge()
sub = rospy.Subscriber("/bebop/image_raw", Image, img_callback, queue_size=1, buff_size=2**32) # dont mess with buffer size, causes drop in fps
pub_image = rospy.Publisher("/obj_image", Image, queue_size =1)
pub_data = rospy.Publisher("/obj_cdnts", numpy_msg(Floats), queue_size = 100)
rospy.spin()
```

```
#!/usr/bin/env python2.7

import rospy
from rospy_tutorials.msg import Floats
from rospy.numpy_msg import numpy_msg
import os


from visual_nav.msg import pos_data
from visual_nav.msg import pos_vector
import pandas as pd

# Do not runs this file from rosrun.
# This code is intended to obtain the position data
# and generate a csv file to calculate variance
# in obtaining position data of the object
# in order to tune the covaraince matrix R
# for Kalman filter tuning


def find_center(x_min,y_min,x_max,y_max):

 x_center = x_min + ((x_max - x_min)/2)
 y_center = y_min + ((y_max - y_min)/2)
 return [x_center, y_center]


def pos_callback(msg):

        global data_measure

 if( msg.data != []):
  x_min = msg.data[0]
         y_min = msg.data[1]
         x_max = msg.data[2]
         y_max = msg.data[3]
  x_center, y_center = find_center(x_min,y_min,x_max,y_max)
         data_measure.append([x_center, y_center])
 else:
  data_measure.append([0, 0])



def kalman_callback(msg):

        global data_predict

 x_pred = msg.posVector[0].x_pos
        y_pred = msg.posVector[0].y_pos
        data_predict.append([x_pred, y_pred])



if __name__ == '__main__':

 cwd = os.getcwd()
        file_path = os.path.dirname(cwd) + '/Desktop/catkin_ws/src/visual_nav/DataLogs/data/log.csv'
        data_measure = []
        data_predict = []

 rospy.init_node("log_data")
 rospy.loginfo("Logging pos data")

 sub = rospy.Subscriber("/obj_cdnts", numpy_msg(Floats), pos_callback, queue_size = 1)
 sub2 = rospy.Subscriber("/Kalman_pred", pos_vector, kalman_callback, queue_size = 1)
```

```
66          rospy.spin()
67
68      print("writing data")
69      df1 = pd.DataFrame(data_measure, columns = ['x_measure','y_measure'], dtype = float)
70        df2 = pd.DataFrame(data_predict, columns = ['x_predict','y_predict'], dtype = float)
71          df = df1.join(df2)
72      df.to_csv(file_path)
```

```cpp
/* Kalman Filter Node */

#include "ros/ros.h"
#include "rospy_tutorials/Floats.h"                  // custom ROS message type Float32 defined in rospy_tutorials package
#include <vector>
#include <iostream>
#include "/usr/local/include/eigen3/Eigen/Dense"       // Newer version of the eigen library 3.37, support for orthogonal decomposition, installed from github  https://gitlab.com/libeigen/eigen.git
#include <cmath>
#include "visual_nav/pos_data.h"
#include "visual_nav/pos_vector.h"
#include "std_msgs/Bool.h"

using namespace Eigen;

// Global Declarations

float dt = 0.1;               // time step T(k) - T(k-1)                    :TODO: Tune the value
float lost = 0;
float q_var = 0.0;            // variance caused due to external disturbances :TODO: Tune the value

float sensor_var_x = 0.19;                          // values obtained by experimentation, see visual_nav/data_logs/log.csv
float sensor_var_y = 0.62;               // variance in the sensor values

Matrix < float, 4, 1 > X;      // [pos_x; pos_y; vel_x; vel_y]
Matrix < float, 4, 4 > P;     // covariance Matrix
Matrix < float, 4, 4 > F;      // Prediction Matrix
Matrix < float, 2, 4 > H;      // Sensor Model Matrix
Matrix < float, 2, 2 > R;      // Covariance Matrix of sensor uncertainity (sensor --> object detection prediction)
Matrix < float, 4, 4 > Q;      // Covariance Matrix of world uncertainity ( Noise added due to slippage, wind or other disturbances)
Matrix < float, 4, 1 > x_predict;    // values obtained for [pos_x; pos_y; vel_x; vel_y] at the end of the predict cycle
Matrix < float, 4, 4 > p_predict;    // values obtained for P at the end of predict cycle
Matrix < float, 2, 1 > inn;      // innovation in sensor meseurement
Matrix < float, 4, 1 > x_update;    // values obtained for [pos_x; pos_y; vel_x; vel_y] at the end of the update cycle
Matrix < float, 4, 4 > p_update;                          // values obtained for P at the end of update cycle
Matrix < float, 4, 4 > I;                        // eye(rows,cols)



std_msgs::Bool is_obj_det;

std::vector<float> kalman_filter(std::vector<float> obj_center,  Matrix < float, 4, 1 > &X,
        Matrix < float, 4, 4 > &P,
        Matrix < float, 4, 4 > &F,
        Matrix < float, 2, 4 > &H,
        Matrix < float, 2, 2 > &R,
        Matrix < float, 4, 4 > &Q,
        Matrix < float, 4, 4 > &I )

{
   /* + This function updates 1 pass of the kalman filter algorithm
      + tracks the center of the object (point)
      + Predicts the path of the point over time
      + returns the prediction to be published for the UAV controller */



MatrixXf sensor_val = Map<Matrix<float, 2, 1> >(obj_center.data());                 // changing std::vector to Eigen::vector
//std::cout<<"Sensor value: " << sensor_val << "\n";

if (!sensor_val.isZero(0))                                                  // if object is detected then correct the measurement
{
  lost = 0;
// --PREDICT CYCLE--

x_predict = F * X;
//std::cout << " x_predict: " << x_predict << "\n";

p_predict = F * P * F.transpose() + Q;
//std::cout << " p_predict: " << p_predict << "\n";


// --UPDATE CYCLE--

CompleteOrthogonalDecomposition <MatrixXf> cqr(H * P * H.transpose() + R);          // pseudo inverse using orthgonal decomposition; pseudo inverse incase matrix has zero determinant
MatrixXf pinv = cqr.pseudoInverse();

Matrix < float, 4, 2> kalman_gain = P * H.transpose() * pinv;                       // calculate Kalman Gain
//std::cout<<"Kalman Gain: " << kalman_gain << "\n";

inn = sensor_val - (H * X);                                                 // calculate innovation/residual
//std::cout<<"Innovation: " << inn << "\n";

x_update = x_predict + (kalman_gain * inn);
//std::cout << " x_update: " << x_update << "\n";

p_update = (I - (kalman_gain * H)) * P;
//std::cout << " p_update: " << p_update << "\n";

X = x_update;
P = p_update;
std::cout<< "P: "<< "\n" << P <<"\n";

}

else                                                                        // if object is not detected then rely on predicted value
{

std::cout<<"No object detected"<< "\n";

if (lost > 2)
{
```

```
{
   lost = 0;
   is_obj_det.data = false;

   X.fill(0);
   P <<   100, 0, 0, 0,              //:TODO: Tune the value
          0,100, 0, 0,
          0, 0, 0, 0,
          0, 0, 0, 0;
   x_predict << 0,
   0,
   0,
              0;

   p_predict << 0,0,0,0,
                0,0,0,0,
                0,0,0,0,
                0,0,0,0;
}

else
{
   lost = lost + 1;
   X = X;
   P = P;               //:TODO: Tune the value
}

}

std::vector <float> pred_obj_center (x_predict.data(), x_predict.data() + x_predict.rows() * x_predict.cols());
// convert eigen matrix to std::vector // std::vector <float> pred_obj_center (X.data(), X.data() + X.rows() * X.cols());

return pred_obj_center;                                          // return tracked position,
}


std::vector<float> find_center(float xmin, float ymin, float xmax, float ymax)
{ /* This function takes the coordinates of the bouding box of the detected object and finds its center
  returns a vector of x and y coordinates of the center of the bounding box */

 float x_center = xmin + ((xmax - xmin)/2);
 float y_center = ymin + ((ymax - ymin)/2);
 std::vector<float> obj_center(2);
 obj_center[0] = x_center;
 obj_center[1] = y_center;
 return obj_center;
}


class SubPub
{

public:

   SubPub()
 {
  pub = n.advertise<visual_nav::pos_vector>("/Kalman_pred", 1000);                  // Declare a Publisher for kalman Prediction
              pub2 = n.advertise<std_msgs::Bool>("/is_obj_detected", 1000);                    // publish true if object has been detected
              sub = n.subscribe("/obj_cdnts", 1, &SubPub::get_obj_cdnts_callback, this);        // Subscribe to the /obj_cdnts topic
 }

   void get_obj_cdnts_callback(const rospy_tutorials::Floats msg)
 {

              /* This function runs callback to subscribe to obj_cdnts topic
               + Acts like a sensor value update
               + Pass the sensor value to the Kalman Filter to update 1 pass of the Algorithm
               + Publish kalman prediction */

               //ROS_INFO_STREAM("object coordinates:" << msg);
                 ROS_INFO_ONCE("Kalman Filter Node started");

                 float xmin = 0; float ymin = 0; float xmax = 0; float ymax = 0;

                 if(!msg.data.empty())
                   {
                     xmin = msg.data[0]; ymin = msg.data[1]; xmax = msg.data[2]; ymax = msg.data[3];
                     is_obj_det.data = true;
                   }  // Avoid segmentation fault in case of empty message (nothing detected)

                 std::vector<float> obj_center = find_center(xmin, ymin, xmax, ymax);
                 std::cout << "Sensor value:- " << "X Center: " << obj_center[0] << " " << "Y center: " << obj_center[1] << "\n";

                 std::vector <float> pred_obj_center = kalman_filter(obj_center, X, P, F, H, R, Q, I);  // Call 1 pass of Kalman Filter
                 std::cout << "kalman Prediction:- " << "X Center: " << pred_obj_center[0] << " " << "Y center: " << pred_obj_center[1] << "\n";
                 std::cout << "-------------------------------------------------" << "\n";

                 visual_nav::pos_data data;
                 visual_nav::pos_vector pos_msg;
                 data.x_pos = pred_obj_center[0];
                 data.y_pos = pred_obj_center[1];
                 pos_msg.posVector.push_back(data);
                 pub.publish(pos_msg);  // Publish Kalman Predictions to topic /kalman_pred
                 pub2.publish(is_obj_det);
          }

private:

ros::NodeHandle n;       // Declare node handle object

ros::Publisher pub;
ros::Publisher pub2;
```

```
ros::Subscriber sub;

};




int main (int argc, char **argv)
{

X.fill(0);

P <<   100,   0, 0, 0,                  //:TODO: Tune the value
       0,  100, 0, 0,
       0,   0, 0, 0,
       0,   0, 0, 0;

F << 1, 0, dt,  0,
     0, 1,  0, dt,
     0, 0,  1,  0,
     0, 0,  0,  1;

I << 1, 0, 0, 0,
     0, 1, 0, 0,
     0, 0, 1, 0,
     0, 0, 0, 1;

H << 1, 0, 0, 0,
     0, 1, 0, 0;

R << sensor_var_x, 0,                   //:TODO: Tune the value
     0, sensor_var_y;

Q << q_var, 0,     0, 0,                //:TODO: Tune the value
       0, 0, q_var, 0,
       0, 0,     0, 0,
       0, 0,     0, 0;

ros::init(argc, argv, "kalman_filter_tracking");                        // Initialise the kalman filter node
SubPub Kalman_filter;                                                   // Declare Class PubSub Object
ros::spin();                                                            // Lets the callback run in loop

return 0;

}
```

```
/* Convert 2d coordinates of the object location to world coordinates in 3d */

#include "ros/ros.h"
#include <vector>
#include <iostream>
#include "visual_nav/pos_data.h"
#include "visual_nav/pos_vector.h"
#include "/usr/local/include/eigen3/Eigen/Dense"  // Newer version of the eigen library 3.37, support for orthogonal decomposition, installed from github https://gitlab.com/libeigen/eigen.git
#include <cmath>
#include "rospy_tutorials/Floats.h"

using namespace Eigen;

// Global Declarations
float d_real = 6.0;         // Length of the diagonal of the bounding box enclosing the vehicle  as seen from the rear end ~ 6.0 mm
float d_img = 0;            // Length of the diagonal of the bouding box obtained from the object detection node
float depth = 0;            // Y world, depth or distance of the object from the camera
float x_world = 0;          // X world, distance from the camera in x direction. (-) object on the left side of the drone, (+) object on the right side of the drone
float y_world = 0;
float focal_length = 385.5;  // Focal length of the camera in pixels. From datasheet; F in mm = 1.8 mm, Horizontal FOV = 80, Image width = 640. F in pixels =  (640 * 0.5)/ (tan 80 * 0.5 * (PI/180))
std::vector <float> img_cdnts;
std::vector <float> pos_3d = {0,0};

// Intrinsic parameters of the camera obtained from calibration datasheet
float Fx = 537.29; // focal length in x direction; F * Sx
float Cx = 427.33; // focal length in y direction; F * Sy
float Fy = 527.00; // camera principal points, center of the image X coordinate
float Cy = 240.22;  // camera principal points, center of the image y coordinate


std::vector <float> conv_2dto3d ( float x_img, float y_img, float d_img )
{
  /* convert image coordinates to world coordinates */
  if (d_img !=0)
  {
 std::cout<<"d image: " << d_img << "\n";
  depth   = ((d_real * focal_length)/d_img);

   std::cout<<"depth: " << depth << "\n";
   x_world = (((x_img - Cx) * depth)/Fy);
       y_world = depth;

   std::cout<<"x_world: " << x_world << "\n";
   std::cout<<"y_world: " << y_world << "\n";

 pos_3d[0] = x_world;
   pos_3d[1] = y_world;
 return pos_3d;
  }

  else                                    // resolve divide by zero error when kalman pred is zero in case no object is detected
  {
 pos_3d[0] = 0; //0                       // prev values
       pos_3d[1] = 0; //0
       return pos_3d;
  }
}


class SubPub
{

public:

   SubPub()

 {
 pub = n.advertise<visual_nav::pos_vector>("/kalman_pred_3d", 10000);                  // Declare a Publisher for kalman Prediction
 sub2 = n.subscribe("/obj_cdnts", 1, &SubPub::get_obj_cdnts_callback, this);           // subscribe to /obj_cdnts to get diagonal of the bounding box
              sub1 = n.subscribe("/Kalman_pred", 1, &SubPub::get_kalman_pred_callback, this);        // Subscribe to the /'K'alman_pred topic

 }

   void get_kalman_pred_callback(const visual_nav::pos_vector::ConstPtr& msg)
 {
       // convert kalman_pred 2d coordinates to world coordinates

             ROS_INFO_ONCE(" Converting 2d position to 3d ");

             const visual_nav::pos_data &data = msg->posVector[0];

             std::vector <float> pos_3d = conv_2dto3d(data.x_pos, data.y_pos, d_img);

             std::cout<<"pos_3d: " << pos_3d[0] << " " << pos_3d[1] << "\n";

             visual_nav::pos_data data_;
             visual_nav::pos_vector pos_msg_3d;
             data_.x_pos = pos_3d[0];
             data_.y_pos = pos_3d[1];
             pos_msg_3d.posVector.push_back(data_);
             pub.publish(pos_msg_3d);
       }

       void get_obj_cdnts_callback(const rospy_tutorials::Floats msg)
 {
       // calculate the approximate diagonal of the bouding box

             float xmin = 0; float ymin = 0; float xmax = 0; float ymax = 0;
             if(!msg.data.empty()) {xmin = msg.data[0]; ymin = msg.data[1]; xmax = msg.data[2]; ymax = msg.data[3];}  // Avoid segmentation fault in case of empty message (nothing detected)
             d_img  = std::sqrt(std::pow(xmax-xmin,2) + std::pow(ymax-ymin,2));
       }
```

```
private:

ros::NodeHandle   n;   // Declare node handle object
ros::Publisher  pub;
ros::Subscriber sub1;
ros::Subscriber sub2;
};



int main (int argc, char **argv)
{

ros::init(argc, argv, "conv_2dto3d");
SubPub conv_2dto3d;
ros::spin();

return 0;
}
```

```
/* Controller to trace the path of the detected object as closely as possible */

#include "ros/ros.h"
#include <vector>
#include <iostream>
#include "visual_nav/pos_data.h"
#include "visual_nav/pos_vector.h"
#include <cmath>
#include "/usr/local/include/eigen3/Eigen/Dense"  // Newer version of the eigen library 3.37, support for orthogonal decomposition, installed from github https://gitlab.com/libeigen/eigen.git
#include "geometry_msgs/Twist.h"
#include "std_msgs/Bool.h"


class SubPub
{

public:

  SubPub()
    {

   sub = n.subscribe("/kalman_pred_3d", 1, &SubPub::get_kalman_pred_3d_callback, this);
   sub2 = n.subscribe("/is_obj_detected", 1, &SubPub::is_obj_detected_callback, this);
        pub = n.advertise<geometry_msgs::Twist>("/bebop/cmd_vel",1000);
    }


   void correct_yaw()
    {

  std::cout << " correcting yaw  \n";
  msg.linear.x = 0; msg.linear.y = 0; msg.linear.z = 0;
        msg.angular.x = 0; msg.angular.y = 0; msg.angular.z = 0;
  pub.publish(msg);
  msg.angular.z = 1;
        pub.publish(msg);
    }

   void pdControl()
    {

  if (!is_obj_detected)
  {

  Ux = 0;
  Uy = 0;

  error << 0,
    0;
  prev_error << 0,
  0;

        msg.linear.x = 0; msg.linear.y = 0; msg.linear.z = 0;
        msg.angular.x = 0; msg.angular.y = 0; msg.angular.z = 0;
  pub.publish(msg);

  return;
   }

  else
  {

        std::cout << " moving to position \n ";
  std::cout << "x world: " << x_world << " " << "y_world: " << y_world << " \n" ;


  desPos << 0,
    0;

  curPos << x_world,
    y_world;

  error = desPos - curPos;
  Ux = Kp_x * error.coeff(0,0) + Kd_x * (prev_error.coeff(0,0) - error.coeff(0,0));
  Uy = Kp_y * error.coeff(1,0); + Kd_y * (prev_error.coeff(1,0) - error.coeff(1,0));

  msg.linear.x = -(Uy); msg.linear.y = Ux; msg.linear.z = 0;
        msg.angular.x = 0; msg.angular.y = 0; msg.angular.z = -0.1;

  pub.publish(msg);

        }
    }

   void get_kalman_pred_3d_callback (const visual_nav::pos_vector::ConstPtr& world_msg)
    {
        const visual_nav::pos_data &data = world_msg->posVector[0];
        x_world = data.x_pos;
        y_world = data.y_pos;
    }

   void is_obj_detected_callback(const std_msgs::Bool& is_obj_det)
    {
       is_obj_detected = is_obj_det.data;
  (!is_obj_detected) ? correct_yaw() : pdControl();
    }

private:
```

```
98
99  ros::NodeHandle   n;
100 ros::Subscriber sub;
101 ros::Subscriber sub2;
102 ros::Publisher  pub;
103 geometry_msgs::Twist msg;
104 bool is_obj_detected = false;
105
106 float Kp_x = 0.1;
107 float Kp_y = 0.1;
108 float Kd_x = 0.1;
109 float Kd_y = 0.1;
110 float x_world = 0;
111 float y_world = 0;
112
113 Eigen::MatrixXf error = Eigen::MatrixXf::Zero(2, 1);
114 Eigen::MatrixXf prev_error = Eigen::MatrixXf::Zero(2, 1);
115 Eigen::Matrix < float, 2, 1 > curPos;
116 Eigen::Matrix < float, 2, 1 > desPos;
117
118 float Ux = 0;
119 float Uy = 0;
120 };
121
122
123
124 int main (int argc, char **argv)
125 {
126
127 ros::init(argc, argv, "controller" );
128 SubPub controller;
129 ros::AsyncSpinner spinner(1);
130 spinner.start();
131 ros::waitForShutdown();
132 return 0;
133
134 }
```