Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Niroj Pokhrel

# Drone Obstacle Avoidance and Navigation using Artificial Intelligence

Master's Thesis
Espoo, April 20, 2018

Supervisor: Professor Alex Jung
Instructor: Enrique Ramirez M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

| | | | |
|---|---|---|---|
| **Author:** | Niroj Pokhrel | | |
| **Title:** Drone Obstacle Avoidance and Navigation using Artificial Intelligence | | | |
| **Date:** | April 20, 2018 | **Pages:** | xii + 102 |
| **Professorship:** | Embedded System | **Code:** | SCI3024 |
| **Supervisor:** | Professor Alex Jung | | |
| **Instructor:** | Enrique Ramirez M.Sc. (Tech.) | | |

This thesis presents an implementation and integration of a robust obstacle avoidance and navigation module with ardupilot. It explores the problems in the current solution of obstacle avoidance and tries to mitigate it with a new design. With the recent innovation in artificial intelligence, it also explores opportunities to enable and improve the functionalities of obstacle avoidance and navigation using AI techniques. Understanding different types of sensors for both navigation and obstacle avoidance is required for the implementation of the design and a study of the same is presented as a background. A research on an autonomous car is done for better understanding autonomy and learning how it is solving the problem of obstacle avoidance and navigation. The implementation part of the thesis is focused on the design of a robust obstacle avoidance module and is tested with obstacle avoidance sensors such as Garmin lidar and Realsense r200. Image segmentation is used to verify the possibility of using the convolutional neural network for better understanding the nature of obstacles. Similarly, the end to end control with a single camera input using a deep neural network is used for verifying the possibility of using AI for navigation. In the end, a robust obstacle avoidance library is developed and tested both in the simulator and real drone. Image segmentation is implemented, deployed and tested. A possibility of an end to end control is also verified by obtaining a proof of concept.

**Keywords:** artificial intelligence, drones, obstacle avoidance, autonomous navigation, computer vision, deep neural network, ardupilot

**Language:** English

# Acknowledgements

This thesis is developed in collaboration with Nokia Networks and Department of Machine Learning for Big Data Group of Aalto University of School of Science. I would like to thank Nokia for providing opportunity and support to undertake this project. I am especially grateful towards my supervisor Professor Alex Jung who incited an interest in Machine learning when I was taking his course on Basics in Machine Learning in Aalto University. His insight and feedback helped in structuring and developing the thesis. The motivation and constant support from my advisor Enrique Ramirez helped me overcome several hurdles.

Espoo, April 20, 2018

Niroj Pokhrel

# List of Figures

# List of Tables

# Abbreviations and Acronyms

| | |
|---|---|
| UAV | Unmanned Aerial Vehicle |
| AI | Artificial Intelligence |
| ML | Machine Learning |
| GPS | Global Positioning System |
| SLAM | Self-localizing and Mapping |
| GPU | Graphical Processing Units |
| IMU | Inertial Measurement Units |
| Lidar | Light Detection and Ranging |
| Radar | Radio Detection and Ranging System |
| Sonar | Sound navigation and ranging |
| UHF | Ultra-high frequency |
| ASIC | Application Specific Integrated Circuit |
| NED | North East Down |
| ECEF | Earth Centered Earth Fixed |
| VFH | Vertical Field Histogram |
| VCP | Vehicle Center Point |
| POD | Polar Obstacle Density |
| SAD | Sum of Absolute Difference |
| ReLU | Rectified Linear Unit |
| CNN | Convolutional Neural Network |
| MLP | Multi-Layer Perceptron |
| SVM | Support Vector Machine |
| ESC | Electronic Speed Controller |
| RC | Radio Controller |
| PWM | Pulse Width Modulation |
| GCS | Ground Control Station |
| MAV | Micro Aerial Vehicle |
| PID | Proportional Integration Differentiation |
| EKF | Extended Kalman Filter |
| SITL | Software In The Loop |
| RTL | Return To Launch |

| | |
|---|---|
| DIGITS | Deep learning GPU Training System |
| SGD | Stochastic Gradient Descent |
| DNN | Deep Neural Network |
| GIS | Geographic Information System |
| RGB | Red Green Blue |
| CIFAR-10 | Canadian Institute for Advanced Research-10 |
| BSD | Berkeley Software Distribution |
| LMDB | Lightning Memory-Mapped Database |
| HDF5 | Hierarchical Data Format 5 |
| FCNN | Fully Convolutional Neural Network |
| CMOS | Complementary Metal Oxide Semiconductor |
| IDA* | Iterative Deeping A* |
| CUDA | Compute Unified Device Architecture |
| CSI | Camera Serail Interface |
| USB | Universal Serail Bus |
| HDMI | High-Definition Multimedia Interface |
| RL | Reinforcement Learning |
| DRL | Deep Reinforcement Learning |
| CC | Companion Computer |
| VTOL | Vertical Take Off and Landing |
| GPL | General Public License |
| UML | Unified Modeling Language |
| SDK | Software Development Kit |
| API | Application Programming Interface |
| SGD | Stochastic Gradient Descent |

# Contents

# Chapter 1

# Introduction

Applications and uses of UAVs (Unmanned Aerial Vehicles), also colloquially known as drones, are drawing a lot of interest in the recent years. UAVs have potential to bring revolution in various fields like logistics and defense, to name a few. However, a number of research works are still needed to realize robust, smart, and truly autonomous drones. Some of these challenges are concerned with obstacle avoidance and autonomous navigation. Given the recent innovation and research outcomes in artificial intelligence (AI), this thesis explores opportunities to enable and improve functionalities in UAVs using AI techniques. In particular, the use of computer vision techniques is explored which can solve many current issues related to obstacle avoidance and autonomous navigation. Further, in order to enable rapid prototyping and research with various modules and techniques, this thesis also proposes new software architecture for drones with enabling components for obstacle avoidance and navigation.

## 1.1 Motivation

In the recent years, we have witnessed a rapid development in the field of sensors technology and the advent of high computation power. This has unleashed several possibilities which were previously deemed impossible. The enabling method for such possibilities has been AI. To put it simply, AI is defined as a search problem where an intelligent agent is trying to find the best possible solution in a given search space of possible solutions. The performance of AI-driven solution has even started to come on par with the solutions crafted by humans [21]. Current trends in AI sort of followed immediately after the development of high-speed internet alongside the massive and faster storage. Datasets containing billions of data and terabytes of memory can be easily found on the internet. These datasets are collected and used to train AI models. Besides the availability of the data, the increasing availability of the computational power also played a major role in the resurgence of AI. Computation power of a processor which looked like it was flattening out in mid-2000 due to excessive heat production saw over a new leaf with the development of parallel computing [7]. Parallel computing is facilitated by Graphical Processing Units (GPUs). They have massive parallel architecture but were only used predominantly for graphics processing in the past. Now they have proved invaluable for the highly parallelizable task such as AI.

AI has already shown its application in the highly diverse field ranging from agriculture, healthcare, finance, geography to robotics and self-driving cars. One such application area which can benefit hugely from the advancements in the AI is the UAV.

UAVs have been proliferating in their use in the recent years. Their advent followed closely with the development of autonomous cars. Unlike a car which runs on the road and has two axes of control, the drone uses air as a medium to fly and thus has to maintain the altitude as well and has three axes of control. When flying autonomously, drone needs to know where it is and how is it going to reach its destination. Such a process of mapping itself to its environment is known as localization. Global Positioning System (GPS) is used for localizing while flying outdoors. In the absence of GPS, other Self Localization and Mapping (SLAM) techniques are used. Navigation is nothing but the process of localizing and moving from one point to another. The process involves estimating position, velocity, and direction by fusing input from different sensors using probabilistic models. The input of sensor readings from GPS, Inertial Measurement Units (IMU), magnetometers and barometers are used for localization and estimation. The process of estimation itself is being done commonly using probabilistic models such as Extended Kalman Filter (EKF) and its variants. Quality of estimates depends on the quality of the sensor reading. As no sensor can provide 100% reliable measurements at all condition, it is common to rely on the readings from multiple sensors for navigation. For example, GPS is only good in outdoor scenarios and its quality drops significantly in the absence of a direct line of sights to satellites. Navigation, when the GPS data is questionable, can be aided with computer vision techniques. Such computer vision model can be developed by training end to end neural networks, as has been shown in the works like that of [9] and [57]. The use of end to end networks removes the need for handcrafting features which tend to be application-specific and not widely applicable.

Another avenue where the AI techniques could be of use in the UAV application is that of obstacle avoidance. The presence of obstacles in the environment can create additional difficulty in navigation for drones. The drone has to estimate the position of the obstacles in its surrounding and maneuver accordingly to prevent the crash. The process of avoiding obstacles can be considered to consist of two steps: detection and avoidance. The detection step is to realize the presence of obstacles in its planned path and stopping the drone from taking the collision course. Avoidance step involves planning an alternate path for avoiding obstacles. The use of range sensors can help in detecting the obstacles in the path of the drone. However, avoidance steps require additional information regarding the nature of the obstacles which will allow the drone to maneuver around the obstacles. The nature of obstacles can be identified by using the AI-driven detection or segmentation algorithms for identifying the regions and sizes of the obstacles from the images. With the use of Convolutional Neural Network (CNN), quite an accurate detection and segmentation model can be developed. Such model helps in accurately identifying the nature of an obstacle which will help in avoidance step of obstacle avoidance system.

As outlined above, AI techniques can be of a great resource for UAVs in the application for navigation and obstacle avoidance. Therefore it is of great interest to the research and development community to have a generic software framework for drones where they can easily experiment and test with different AI-driven navigation and obstacle avoidance modules. One of the most commonly used software frameworks is Ardupilot[60]. While Ardupilot has support for obstacle avoidance module, it still has some serious shortcomings. Ardupilot framework provides easy integration to flight controller at the expense of flexible code and algorithm modifications. Thus, it is difficult to tailor the Ardupilot according to the need of researchers and developers. Besides an exploration of the use of AI techniques to solve challenges of drones related to navigation and obstacle avoidance, this thesis also addresses the problem in software frameworks for drones like Ardupilot by building a modular software framework which can be easily extended to support different algorithms and sensors.

## 1.2 Objective and Scope

The first goal of this thesis is to build modular software architecture for UAVs with enabling components like obstacle avoidance. Such architecture should be easily extendable and simple but powerful enough to support any custom modules for obstacle avoidance and navigation. The modularity of the architecture should support both additions of new sensors and avoidance algorithms. One should be able to easily collect data, preprocess the data and analyze the data using an implementation of the said architecture. Such a processing pipeline for the input data is crucial for decision making for obstacle avoidance. After building such architecture, the test of the overall functionality of the system can be done with both simulation and field experiments with a real drone.

The second goal of the thesis is to explore the possibilities of using AI for obstacle avoidance. Such a scoping is possible due to our developed software architecture which provides support for varieties of new sensors and custom obstacle avoidance algorithm. Semantic segmentation techniques are to be explored in this thesis for the obstacle avoidance algorithm. The semantic segmentation is used as a proof of concept for verifying how AI can be used in obstacle avoidance. The developed semantic segmentation can be tested with aerial drone dataset [51] to verify the functionalities. The obstacle avoidance architecture discussed in the first goal should be able to support input from the developed segmentation model, as a further proof-point. The development of AI-driven capabilities in drone will help in better understanding the nature of obstacles and thus find better avoidance route.

The final goal of this thesis is to explore, implement, and test AI-agent driven techniques for the navigation of the drone. This can be tested using the forest trail dataset [57] using an end to end deep learning model for pose estimation and flying the drone across the forest trails.

In summary, the objective of this work is to develop a modular software architecture for drones which allows rapid prototyping and research on different components of the drones. With this, we explore and report different AI-driven obstacle avoidance and navigation algorithm for drones. An in-depth study of the application of artificial intelligence techniques for functionalities in drones is out of the scope of this thesis. Due to the limitation of time and complexity, many open problems and topics are not explored. However, this thesis will pave a path for anyone who wants to dive deeper into applications of AI in drones and use our software architecture to experiment easily and quickly with their research propositions.

## 1.3 Research Problems

The research problems for the thesis are formulated based on the three goals defined in section 1.2. Firstly, building a modular software architecture requires a thorough understanding of quadrotors. This is done through research on different sensors and algorithms used for the estimation and control of the drones. In addition to this, understanding flight kinematics and communication protocols used for controlling a drone also facilitate the development of such software architecture.

For the second goal, which explores possibilities of using AI for obstacle avoidance, understanding of AI, in general, is required as the first step. The research on current trends in deep learning such as the convolutional neural network which is indispensable for developing state of art computer vision technology is done next. Furthermore, research on autonomous cars is also an invaluable resource for finding out the popular trends for solving the problems of obstacle avoidance and navigations.

Finally, developing an AI agent for navigation requires further research on AI on top of the basic knowledge. For the scope of this thesis, research is conducted on end to end convolutional neural network and deep reinforcement learning for developing such intelligent agents.

In summary, there are several fields to explore, study and research. The sources of such information are research papers, online documentation, source codes, books, websites, and manuals.

## 1.4 Structure of the Thesis

Rest of this thesis explains what steps from knowledge gathering to implementation and testing were undertaken for attaining the predefined scopes and objectives discussed in section 1.2. Chapter 2, 3 and 4 summarizes research undertaken for understanding various components. Chapter 2 provides basic background where different topics needed to understand this thesis is presented with an introduction to navigational unit, obstacle avoidance unit and AI. Then, follows a chapter on a case study of autonomous cars to comprehend underlying technology currently used in autonomous cars. Understanding quadrotor and flying principle through discussion of hardware components, software components and controller is discussed in chapter 4. The design of the system, the methodology, and datasets used are discussed in chapter 5. The description of the implementation, experimentation, and results of different sensors integration to the obstacle avoidance library is discussed in chapter 6. It also includes discussion about the possibility of using image segmentation for obstacle avoidance and using end to end deep neural network for navigation. Chapter 7 provides a discussion of the achievements and outcomes of the thesis and finally, chapter 8 concludes the thesis.

# Chapter 2

# Background

Understanding UAV requires understanding different components associated with it. As discussed in section 1.2, different goals of this thesis are developing a software architecture, obstacle avoidance system and navigation system for quadrotor drone. This requires an understanding of different peripherals associated with it. On a functional level, three components listed below were explored in detail. This section introduces different sensors, algorithms, devices, and software associated with each of them.

1. Navigation Unit
2. Obstacle Avoidance Unit
3. Aritificial Intelligence

## 2.1 Navigation Unit

Navigation unit comprises of sensors and controllers which is required by the UAV for flying. It consists of fundamental parts required for navigating from one point to another. Fusing the input from sensors such as IMU, GPS, magnetometer, and barometers are useful for estimating the position, velocity, acceleration, and orientation of the vehicle. This information is used by the UAV to provide the corresponding input to the controller for navigating to its destination. The control command is converted to the thrust which is applied to the motors by controllers such as proportional-integral-derivative (PID). This section introduces sensors such as IMU, GPS, and optical flow. It also introduces simple probabilistic model such as Bayes algorithm that can provide a good estimation. The foundation built with Bayes algorithm will be useful later when discussing Kalman filter and extended Kalman filter which is more robust at estimation compared to Bayes algorithm and currently used in many flight controllers. The navigation unit is summarized in figure 2.1.

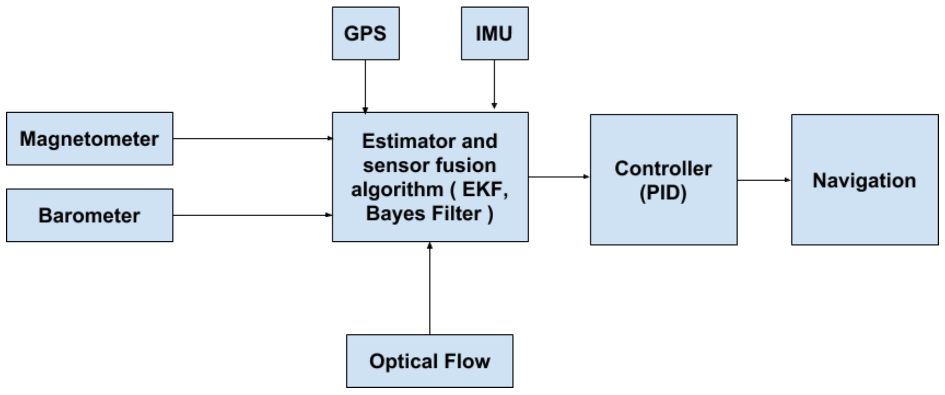


Figure 2.1: Navigation Unit

### 2.1.1 Inertial Measurement Units (IMU)

IMU is a device that helps in estimating relative position, velocity, and acceleration of moving vehicles using gyroscopes and accelerometers. IMU is of two types: gimballed and stripped down. Gimballed IMU keeps mass in the horizontal position and is free to move in any direction. Stripped down IMU has fixed system connection which calculates orientation. The gyroscope is used to measure changes in position and is built with technologies such as fiber optic, a ring laser, hemispherical resonator, and MEMS. Similarly, the accelerometer is used to measure external forces including gravity. The force is calculated based on the deflection of the mass. Despite being essential equipment for measuring motion, IMU inherits several problems such as random drift caused by measuring errors leading to short or long-term drift. Nevertheless, the problem of drift can be mitigated by fusing data from sensors like GPS using an appropriate filter such as Kalman filter [5]. It is commonly used in devices which requires estimating motions such as planes, cars, smartphones, robots, and drones.

### 2.1.2 Global Positioning System (GPS)

GPS is a satellite-based navigation system consisting of 24 satellites orbiting the earth. These satellites revolve around the earth every 12 hours transmitting signals containing a set of three values. Three values are a unique number assigned to each satellite, position of the satellite in space and time of transmission of the signal. GPS receiver on the earth calculates its distance with respect to these satellites based on the signals received. For estimating the position on the surface of the earth with triangulation, signals from three or more satellites are needed. The number of signals from satellites increases to four or more for calculating elevation as well. However, the downside of GPS is its stringent requirement for the receiver to maintain a line of sight with the satellites which makes it useless for working in indoor environment, forest, and city with high structures[18]. Despite some shortcomings, GPS is popular and useful in outdoor robotics and drones, GIS data collection, surveying and mapping.

### 2.1.3 Optical Flow

To overcome shortcomings of GPS which require a direct line of sight to the satellites, visual odometry sensors such as optical flow can be used to estimate velocity and position using camera and range sensor. Optical flow camera has an ultrasonic sensor, Complementary Metal Oxide Semiconductor (CMOS) image sensor, and gyroscope as shown in figure 2.2. The ultrasonic sensor is used for scaling the distances whereas gyroscope is used for angular rate compensation. The CMOS image sensors enable optical flow sensors to operate indoor and outdoor environments with low light conditions. Furthermore, the distance measured by the ultrasonic sensor is also used for scaling optical flow values for calculating velocity. The calculation of the flow can be done using various methods such as phase correlation, block based, differential and discrete optimization methods.

Optical flow calculates the flow or motion of different points of interest in consecutive frames of video or images. One of the popular block-based method commonly used for flow calculation is the sum of absolute differences (SAD) [34]. SAD algorithm serves as the first step and measures a similarity between different image blocks. For calculating flow, these different image blocks will be the reference block of pixels of the current and preceding frame. The position of the match between these reference frames is selected and used for calculating flow value. Furthermore, subpixel refinement can also be done for better accuracy using bilinear interpolation.

Another prominently used differential based algorithm is Lucas-Kanade [67]. It uses affine model and image patches for flow field. It is less sensitive to noise but is a local method. Once the flow is calculated from the images, it can be used to estimate translational velocity. Translational velocity is calculated from two-dimensional flow field after scaling it to current scene and compensating angular rate. Thus, using optical flow gives an accurate estimate of position and velocity. Optical flow is used for high estimation accuracy and in the absence of GPS signals.

Figure 2.2: Optical Flow sensor with CMOS image, gyroscope and ultrasonic[34]

.

### 2.1.4 Bayes Algorithm and Probabilistic Model

Drones cannot directly observe the real world; thus there is a need to estimate the motion so that they can localize themselves and move towards the destination. By getting feedback and reading from sensors, a drone can estimate its position against the known surroundings. State estimation is done based on position and velocity of the drone and map of the environment where the drone is operating. The estimation can be done in one of three ways which are control based, odometry based and velocity based. In control based estimation, robot estimates motion from the control commands issued to it. Similarly, odometry based are used for the systems with distance sensors such as wheel encoders, whereas, velocity based state estimation is used in the absence of wheel encoders. There are several algorithms and filters used for controlling and navigating the drones, and one of the basic filters is Bayes filter which is discussed in succeeding paragraphs.

Before jumping into the Bayes algorithm, it is required to understand how Bayes formulates sensor and motion model. Robot perceives its environment through its sensors and can be defined as in equation 2.1.

$$z = h(x) \tag{2.1}$$

where,
$z$ is sensor reading
$h$ is sensor model (observation function)
$x$ is world state
Equation 2.1 shows how we can get output based on the current world state

and sensor model. However, we can get the world state as well if we know the sensor reading by inverting the sensor model as shown in equation 2.2.

$$x = h^{-1}(z) \tag{2.2}$$

Similarly, based on the motion, state of the drone can be estimated. Belief state will be updated based on action or control command issued to the drone. It is defined in equation 2.3.

$$x^{'} = g(x, u) \tag{2.3}$$

where,
$x^{'}$ is current state
$g$ is motion modeling
$x$ is the previous state
$u$ is executed action
The motion model is vague and prone to errors with the increase of time. However, it has a higher frequency and can update up to 500-1000 Hz compared to sensor model which has much lower update rate at 10-100Hz. Thus, the motion model can calculate the belief state at a higher frequency which can be corrected with sensors model. The sensor model is also not completely accurate and is often noisy and incomplete which makes the model partially wrong. Such a problem can be mitigated with the prior knowledge about the environment where the robot is operating, thus improving the estimate. This information can be used to verify the belief and estimates calculated from sensor and motion model. Thus, the estimation is done more in the probabilistic sense rather than trying to find the exact position [65]. Probabilistic models is represented as shown in equations 2.4 and 2.5.

$$Probabilistic\ sensor\ models = p(z|x) \tag{2.4}$$

$$Probabilistic\ motion\ models = p(x^{'}|x, u) \tag{2.5}$$

As shown in equation 2.6, input from different sensors can be fused which is also known as multi-modal models and is used to improve the accuracy of the estimation.

$$p(x|z_{vision}, z_{ultrasound}, z_{IMU}) \tag{2.6}$$

Understanding Markov assumption is also necessary for understanding the probabilistic models and is defined by two statements.

1. Sensor observations depend only on current state

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.7}$$

2. Current state depends on current action and previous state

$$p(x_t|x_{0:t-1}, z_{1:t}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.8}$$

With Markov assumption, the probabilistic state estimation estimates state of the dynamic system given the sequence of observations and actions, sensor model, action model and prior probability of the system. Such an estimation of the state is also called a belief.

$$Bel(x_t) = p(x_t|u_1, z_1, ..., u_t, z_t) \tag{2.9}$$

In motion model, it tries to calculate the probabilistic region of interest based on the motion. However, measurements are inaccurate which weakens the estimation, and increases the probable region for localizing drone. Thus, input from sensor model is used to correct the readings from motion model. Bayes filter algorithm can be summarized in two steps [58] which is given below.

Repeat for each time step,

1. Apply motion model

$$Bel^{'}(x_t) = \sum_{x_{t-1}} P(x_t|x_{t-1}, u_t)Bel(x_{t-1}) \tag{2.10}$$

2. Apply sensor model

$$Bel(x_t) = (z_t|x_t)Bel^{'}(x_t) \tag{2.11}$$

## 2.2 Obstacle Avoidance Unit

Similar to the navigation unit discussed in 2.1, obstacle avoidance unit is an integral part of a UAV. Though it is not required for a drone to fly, it ensures the vehicle reaches the destination safely. UAV can sense and react to the environment both dynamic and static based on the input from this unit. While navigation unit concerns about reaching the destination through the shortest possible path, obstacle avoidance unit concerns about reaching the destination with short and safest path. It comprises range sensors such as lidar, radar, sonar and depth cameras. The input from these sensors is provided to obstacle avoidance algorithms such as thresholding or vertical field histograms. This section introduces some of the range sensors and vertical field histograms algorithm for obstacle avoidance. The obstacle avoidance unit is summarized in 2.3.

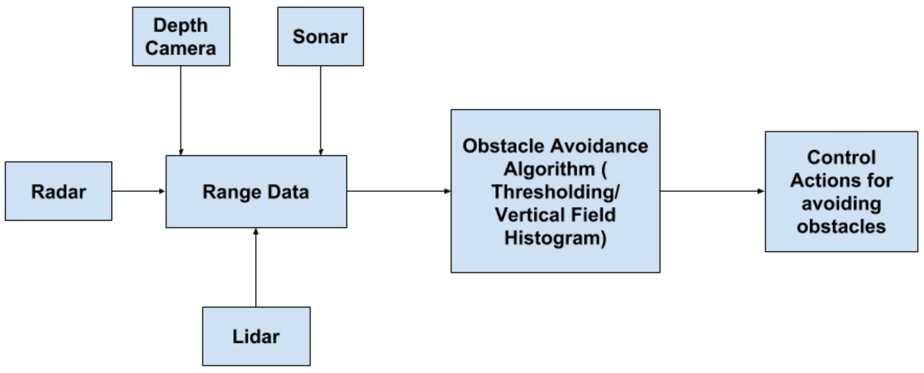

Depth Camera
Sonar
Radar
Range Data
Lidar
Obstacle Avoidance Algorithm ( Thresholding/ Vertical Field Histogram)
Control Actions for avoiding obstacles

Figure 2.3: Obstacle Avoidance Unit with corresponding components

## 2.2.1 Light Detection and Ranging (Lidar)

Light Detection and Ranging is a method for measuring ranges with a light in the form of pulsed laser. There are several lidar available commercially such as Garmin lidar lite [29], Velodyne lidar[25] and Rplidar[54]. For this thesis, Garmin lidar lite is used as obstacle avoidance sensor. The application of this sensor can be found around unmanned vehicles, robot, and drones for detecting range and proximity. The size of the device is compact with low power consumption which can be useful for autonomous vehicles. The communication with the sensors can be done either with I2C or PWM. Figure 2.4, 2.5 and 2.6 shows Garmin lidar lite, Velodyne lidar and Rplidar respectively.

Figure 2.4: Garmin Lite

Figure 2.5: Velodyne

Figure 2.6: Rplidar

### 2.2.2 Radio Detection and Ranging System (Radar)

Radar uses electromagnetic waves for finding the relative coordinate of the object in respect to its position. It works by radiating energy in UHF and microwave range, and monitoring the echo reflected back from the objects. The primary radar system consists of a transmitter which produces an electromagnetic signal radiated into space through an antenna. This electromagnetic signal is either reflected back or reradiated when it strikes on objects. The reflected signals which are received by Radar antenna are processed to determine the position of objects[59]. The distance is calculated by multiplying the speed of light with the time taken by signals to travel from the radar to the target.

### 2.2.3 Sound navigation and ranging (Sonar)

Unlike radar which is based on electromagnetic waves, sonar is based on sound waves. The detection of the object is based on the propagation of sound from target detector. There are two types of sonar, active and passive [24]. Active sonar system transmits waves which travel back to the receiver. However, in passive sonar system target is the source of energy propagating to the receiver. Distance to the object is calculated by the speed of sound in the medium multiplied by the time taken to traverse the distance.

### 2.2.4 Depth Camera

Depth camera provides an additional information of depth value in addition to common Red-Green-Blue (RGB) value for each pixel. Depth information gives drones or any other computer vision application capability to perceive three dimensions of its environment. The process of finding the depth itself is usually done with stereoscopic vision in which two cameras are used. The camera can be used for not only perceiving the obstacles in the environment where drones are operating but also for finding a safe path to navigate through the obstacles. Several depth cameras are available in the market such as Microsoft Kinect, Intel RealSense, and ZED stereo cameras. This thesis uses Realsense camera which is a depth camera developed by Intel and shown in the figure 2.7. The process of calculating depth is given by the equation 2.12. In the equation 2.12, the baseline is the separation between the two-identical infrared camera, the focal length is the focal length of the camera and disparity is the differences between the images obtained from two cameras.

$$Depth = (Baseline * FocalLength)/Disparity \tag{2.12}$$

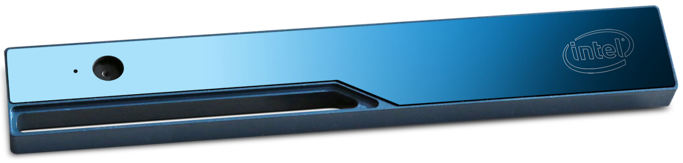


Figure 2.7: Intel Realsense R200 [53].

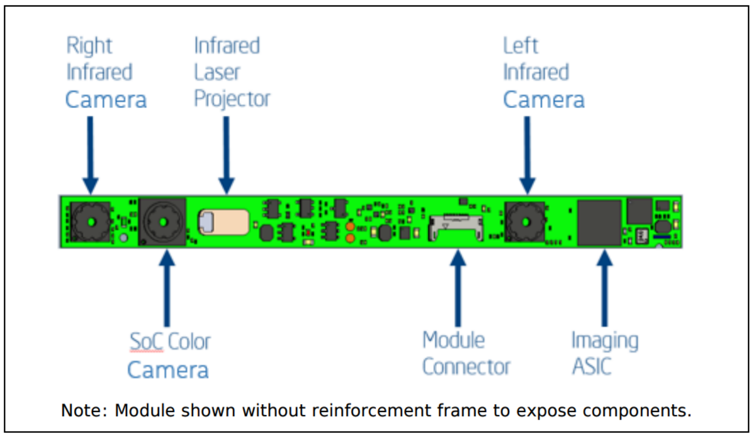


Figure 2.8: Intel Realsense R200 internals as provided in [53].

Figure 2.8 shows the interior of the camera which includes following components

1. Imaging ASIC onboard camera
2. Depth capture in VGA resolution
3. Class 1 Infrared laser projector system

The R200 camera provides several video streams such as color, depth and infrared. The difference between depth video streams and color video is based on what each pixel represents. The pixel in color video stream encodes RGB values whereas pixel in-depth video streams represent depth. The module consists of infrared laser projection system, two infrared cameras, and a full HD color imaging sensors. Per-pixel depth is calculated with stereo vision technology in assistance with the infrared laser projector and the two infrared imaging sensors [40].

### 2.2.5 Polar Histogram/Vertical Field Histogram

Vertical Field Histogram is a method for finding the obstacles present on the navigation path of drone based on the input from range sensors creating a polar obstacle density as shown in figure 2.9. The world is modeled as a two-dimensional histogram grid which is updated continuously with distance data obtained from range sensors. The process of creating a world model involves two stage of data reduction which in turn has three level of data representations [10]. The first level of data representation involves continuously updating cartesian histogram grid in real time with range data from sensors. The second level of data representation involves constructing one-dimensional polar histogram(H) around drone's momentary location. The third level of data representation is the command for navigating the drone.

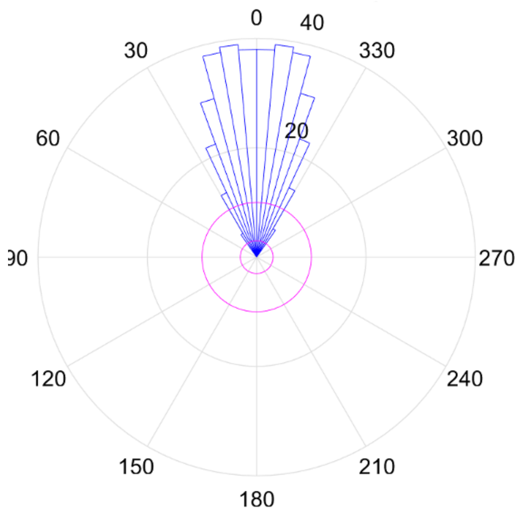


Figure 2.9: Polar Obstacle Density [20]

**Creation of Polar histogram** The polar histogram $H$ comprises of $n$ angular sectors each of width $\alpha$ as shown in figure 2.10. An active region $C^*$ which is a region drone currently sees is transformed such that each sector $k$ is holding a value $h_k$. This value represents polar obstacle density (POD) in the direction of the sector $k$. Active region window moves with the vehicle overlying a square region of $w_s * w_s$ cells in the histogram grid. Contents of active cells in the histogram grid are treated as obstacle vectors, the direction of which is determined by the angle between the cell and the Vehicle Center Point (VCP).

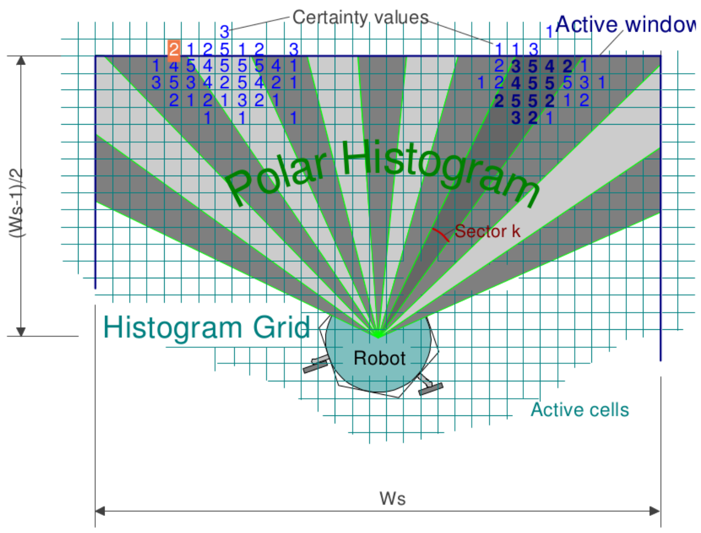


Figure 2.10: Vertical Field Histogram [10] showing the formation of polar obstacle densities in front of the vehicle.

**Steering controller** The next stage computes steering direction $\theta$. The smooth polar histogram has peaks and valleys representing sectors with high and low PODs respectively. Valley with POD below a certain threshold is called as a candidate valley which can be used for navigating the drone. There can be more than one candidate valley to choose from, and the selection of appropriate valley is based on minimum deviation from the direction of the target. Valley may be comprised of multiple sectors, thus after selection of the valley suitable sector within that valley has to be chosen.

## 2.3 Artificial Intelligence

This thesis explores how AI can be used to assist obstacle avoidance and navigation which requires understanding different components of AI and how it can be used in UAV. In this section, the general concept of AI, search problems, neural network, convolutional neural network, reinforcement learning and deep reinforcement learning is introduced. It also discusses a software tool caffe and digits (Nvidia Deep Learning GPU Training System) for implementing the convolutional neural network. Finally, it summarizes current research ongoing on AI for drones. The architecture for AI used in this thesis is summarized in 2.11.

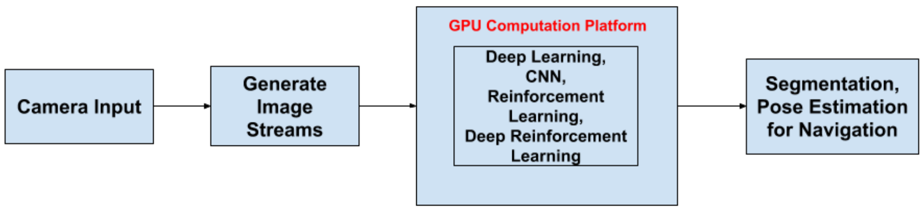

Camera Input
Generate Image Streams
GPU Computation Platform
Deep Learning, CNN, Reinforcement Learning, Deep Reinforcement Learning
Segmentation, Pose Estimation for Navigation

Figure 2.11: Aritificial Intelligence Unit showing different components involved for developing an intelligent system.

### 2.3.1 General Introduction to AI

Artificial intelligence has established itself as an integral part of robotics. The definition of AI can differ depending on its usability and field, but as defined in [50], it is designing of an intelligent agent which can interact with the environment and take action to maximize its success where an agent acts rationally to get the best outcome. AI can also be described as a search problem where an agent is trying to find the best possible solution out of several choices. Thus, local search problems such as hill climbing, adversarial search problems such as minimax and alpha-beta pruning and uninformed search such as A* or IDA* are part of the AI. Due to its vast nature, it has found applications in several fields such as speech recognition, handwriting recognition, machine translation, robotics, recommendation system, spam filtering, face detection, face recognition and autonomous driving. Some of the concepts of Artificial Intelligence is discussed in succeeding paragraphs.

**Intelligent Agents** An agent is capable of perceiving and taking action to attain some goals in an environment. AI tries to define a rational agent which can maximize its reward under different constraints such as limitations of computation power. Based on complexity, an agent can be of different types such as simple reflex agent, model-based reflex agent, goal-based agent and utility-based agent [56]. A simple reflex agent has no memory and selects an action based on current state only. This type of agent can work efficiently only in the fully observable environment. A model-based agent has some memory and is operable in the partially observable environment. Goal-based agent understands its goals and knowledge of its environment which it uses to attain that goal. Finally, the utility-based agent has a utility function for measuring the performance of the agent. All the above agents can be generalized as learning agents who have four components learning, performance, critic and problem generation.

**Environment** The environment is where an agent performs an action. It is of several types such as fully observable or partially observable, deterministic or stochastic, episodic or sequential, static or dynamic, discrete or continuous, single agent or multi-agent and known or unknown.

**Search Agents** AI can be generalized as a search problem so an agent in AI is basically a search agent. Such agent is goal oriented and tries to identify series of actions to attain the defined goal. Search problem can be defined with an initial state, state space, action space, transition model, a test of goal and cost of the path. The initial state is where search agents start its search. Set of states that can be reached from the initial state is state space. Set of actions available to the agent is known as action space. Transition model defines if an agent can go from one state to another and what actions are required for such transition. Test of goal checks whether the defined goal is reached or not. Cost of the path is the total cost incurred to reach the goal.
Search can be Uninformed search or Informed search based on whether search agent has knowledge of the domain it is searching. In uninformed search, an agent has no information regarding searching criteria, thus, must search entire state space in a brute force manner. Some examples of such searches are breadth-first search, depth-first search, depth-limited search, iterative deepening and uniform cost search. However, an agent can have a knowledge of the domain through a heuristic function. The heuristic function measures the closeness of state to its goal. Such search is known as informed search and greedy best-first search, A* search, and Iterative Deeping A* (IDA*) search are some examples. All the searches discussed earlier are trying to find the best path by applying optimization globally. However, search can be local and local search is useful for optimizing complex problems.

**Local Search** The real-world problems are usually much complex and are not suitable to apply the generic global search approaches discussed above, instead, a search can be done locally by iteratively improving the utility.

Local search tries to keep a single current state and tries to improve it without maintaining a search tree. Thus, it has less memory and performs better in large state spaces. Some examples of local searches are hill climbing, simulated annealing, local beam search and genetic algorithms.

**Adversarial Search** Adversarial Search is used in multiagent competitive environments such as games. In games, there usually is an adversary not under the control of the agent acting to minimize the utility. Two popular adversarial searches are minimax and alpha-beta pruning. In minimax, there are usually two players one of them is trying to maximize its utility whereas the other one who is adversary is trying to minimize it. Alpha-beta pruning is like minimax but has better performance as it keeps track of two bounds for pruning the search space. Alpha is the largest value for maximum across visited state spaces and beta is the lowest value for minimum across visited children.

### 2.3.2 GPU Computing Platform

Graphics Processing Unit (GPU) is used for processing graphics. Processing of videos and images are highly parallel in nature such as mean subtraction from image involves subtraction of mean value from entire pixels of the image. For that reason, GPU was created to be a highly parallel system with hundreds of cores running thousands of threads. This support for the highly parallel system is realized to be highly efficient for computing parallel task such as training Neural network. Thus, GPU saw a rise in its demand recently with increased uses of AI. This thesis is using Jetson TX2 as a GPU computing device which is shown in figure 2.12.

Jetson TX2 is one of the dominant embedded AI computing device developed by Nvidia which has 8 GB memory and 59.7 GB/s memory bandwidth. GPU architecture is Nvidia Pascal with 256 Compute Unified Device Architecture (CUDA) cores. In addition to that, it has quad core 64-bit arm, eight processors. For interfacing the video, it has Camera Serial Interface (CSI), Universal Serial Bus (USB), High-Definition Multimedia Interface (HDMI) and gigabit ethernet port. It can support processing of up to 6 HD videos. Furthermore, it includes latest technology for deep learning and computer vision which makes it an ideal candidate for embedded AI computing. In addition to reliable hardware resources, it comes with a platform that enables smooth implementation of Artificial intelligence. The difficulty to start developing AI is very low with Jetson hardware and software resources. The SDK includes deep learning (tensorRT, cuDNN, Nvidia Digits workflow), computer vision (Nvidia visionworks, OpenCV), GPU compute (Nvidia Cuda, Cuda libraries) and multimedia[52].

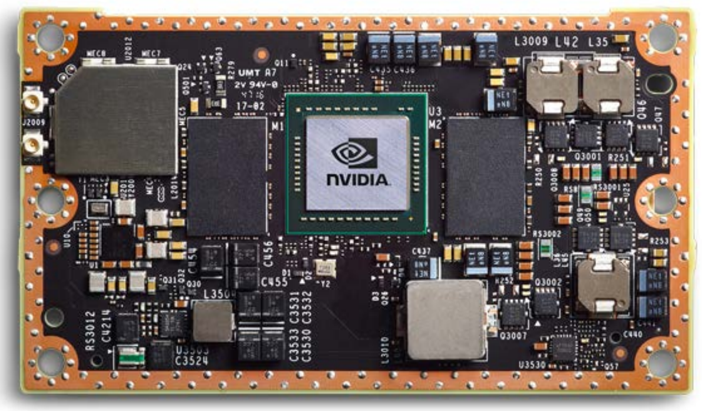


Figure 2.12: Nvidia Jetson TX2 model [52]

### 2.3.3 Neural Network

The neural network is a network of neurons arranged in several layers and dimensions. The fundamental part of any neural network is a neuron. Each neuron has a set of inputs, weights, biases and activation functions. The structure of the neuron is defined and shown in figure 2.13.

**Input** For the first layer, an input of a neuron will be data, but for hidden layers, it will be the output of the neuron in preceding layer.

**Weights** After receiving the inputs, neuron computes the weighted sum by assigning a parameter known as weight for multiplying with each input variable. Weights are an important parameter because the activation of a neuron depends on its value. Learning in the neural network means finding proper value for these weights parameters. During the first step of training, the weights should be initialized with some small random value but not zero.

**Bias** The weights are used for computing linear weighted sum of input; however, there might be a need for thresholding this value depending on the applications and data. Bias is a constant and is used to threshold the output of weighted sum. It can be positive or negative depending on which direction thresholding is needed. It is also another learnable parameter and is learned over the course of training. The initialization of bias can be zero during training. Each neuron possesses a single bias to shift its weighted sum up or down.

**Activation Function** The summation of the weighted sum and bias models a linear system. However, most of the real-life systems are nonlinear. In such a scenario, activation function provides a mechanism for representing a nonlinear system. The output of summation is fed into activation function which does the nonlinear transformation.

**Output of Neuron** It is the output of the activation function. For the hidden layers, this output is fed as input to succeeding layers. However, in the output layer, output calculates the weighted sum and provides a classification or regression data as per the application [38].

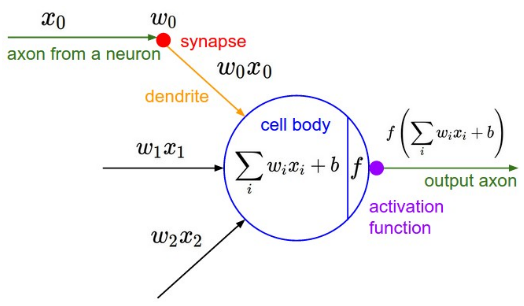


Figure 2.13: Single Neuron[38]

### Activation Function

As discussed above, one of the fundamental parts of a neuron is its activation function, in the absence of which neural network cannot represent anything more than a linear system[37]. The performance of neural network and training duration can be largely influenced by choice of the activation function[38]. Some of the popular activation functions are discussed below.

**Sigmoid** It is also known as logistic function and is represented by equation 2.13. Parameter $a$ which is a slope is important. When it approaches infinity, logistic sigmoid approaches threshold function, and when it approaches zero, the function has a large linear region between the threshold. It is continuous and continuously differentiable.

$$f(v) = 1/(1 + e^{-av}) \tag{2.13}$$

The problem with sigmoid function is that it saturates and kills the gradients. Furthermore, the output from the sigmoid functions are not zero-centered. Output value ranges from [0, 1]. Sigmoid function is shown in figure 2.14.

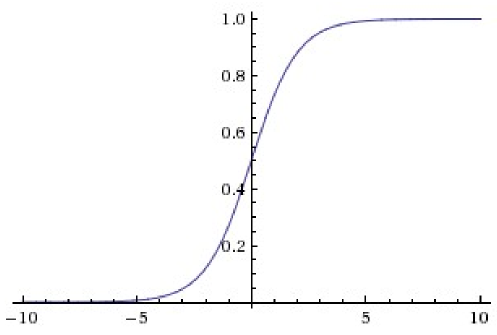


Figure 2.14: Sigmoid Function

**Tanh** The symmetric version of sigmoid function is tanh function which is represented by equation 2.14.

$$tanh(x) = (e^x - e^{-x})/(e^x + e^{-x}) \tag{2.14}$$

This activation function overcomes the problem of sigmoid of not being zero centered. However, the problem of saturation persists. The value of such a function range from [-1, 1]. Tanh function is shown in figure 2.15.

**ReLU** It is getting more popular with deep neural network and is defined by equation 2.15.

$$f(x) = max(0, x) \tag{2.15}$$

ReLU helps in accelerating the convergence of stochastic gradient descent when compared with sigmoid and tanh. Implementation is quite straightforward which is done by simply thresholding a matrix of activation at zero. However, it can be fragile during training and die. ReLU function is shown in figure 2.16.

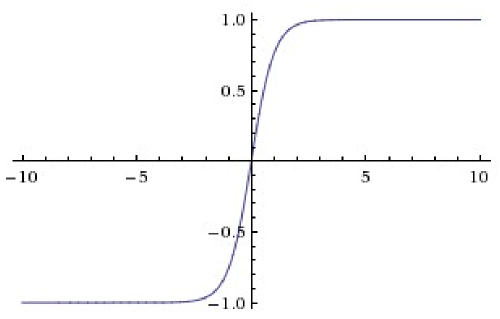


Figure 2.15: Tanh

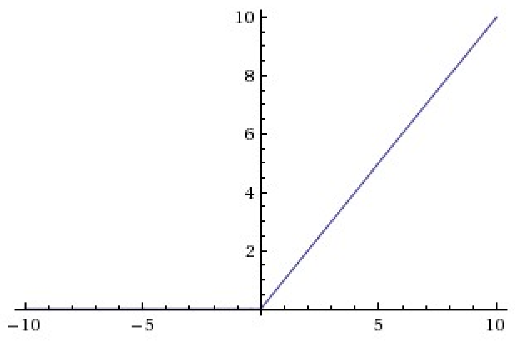


Figure 2.16: ReLU

### Multilayer Perceptron

A single neuron can be used by itself for modeling some simple system. By stacking such neurons into multiple layers, a complex system can be represented, and such a network of neurons is known as multilayer perceptron. The first layer is called input layer, and no computation is done in this layer. The final layer is output layer which only does linear combinations of input from an earlier layer and has no nonlinear component. All the layers in between are the hidden layer. Each neuron in the hidden layer has the architecture like the one shown in figure 2.17.

### Backpropagation

With no training, all the parameters such as weights and biases have random values or zeroes. Such a system does not represent anything. For making a neural network useful, it needs to undergo training so that all the parameters

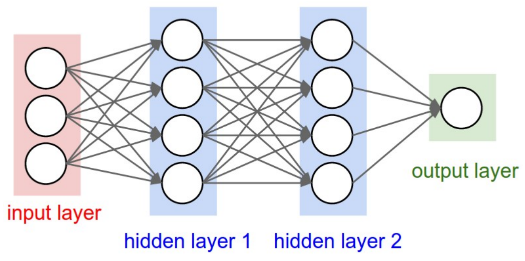


Figure 2.17: Multilayer Perceptron [38]

such as weights and biases are properly updated to model the system. The first step of training is to initialize these parameters. The network will be incapable of representing much if all the weights in the network are initialized to zero. With zero initialization, the weighted summation of input will be zero, and all the neurons in each hidden layer will be learning the same thing [49]. Also, the choice of initialization affects the performance and time of convergence of training of a neural network.

During an initial phase of training, excitation of neurons is random generating random output. For the supervised learning where labeled data is available, the difference between what is expected and what is generated can be observed. This difference between the expected output and real output is a loss. The losses over all the training examples are averaged by defining a cost function which is used to update the parameters of the network. In a neural network, least square cost function is popular, where the sum of squared difference between the expected output and real output is calculated. Once a cost is calculated, weight is adjusted such that it will start favoring expected outcome. The process of updating weights and biases starts from output layer with stochastic gradient descent (SGD) method [37]. The process does not stop in the outermost layer but proceeds to the earlier layer until it reaches the first hidden layer. The name backpropagation is thus coined as the algorithm starts from the last layer and propagates backward to the first layer.

### 2.3.4 Convolutional Neural Network

Convolutional neural networks are used for image classification [43][42]. The way we store the image in the memory leads to loss of spatial information. Spatial information is lost because images are flattened and represented as an array of pixels in one-dimensional memory. When this representation is fed to the machine, it just sees a bunch of pixels. As a result, all the spatial information present in the image is lost. CNN tries to preserve this spatial information by using filters which convolve with multiple pixels in a small window of 3x3, 5x5 or 7x7 sizes. Convolution is used for searching spatial features in images such as contours, lines, circles, honeycomb or any subtle features [19].

### Comparison with Multilayer Perceptron (MLP)

Convolutional Neural Network is a variant of the multilayer perceptron. However, unlike generic MLP, CNN has sparse connectivity and shared weights [22]. MLP has fully connected layers which means output from a layer is connected to all neurons in the subsequent layer, but CNN only has spatially contiguous connections. The learning parameters are weights in MLP, but CNN has two-dimensional filters. Each of these filters is used to convolve images creating a two-dimensional output. The number of filters can be more than one generating three-dimensional output. Same weights of the filter are used to convolve entire image in contrast to MLP which has different weights for different input.

Like MLP, CNN has repetitive blocks of neurons creating 3D volumes of neurons. As shown in figure 2.18, a convnet has 3 dimensions. Different operations are happening in various stages of CNN which can be repeated over. Different operations of the convolutional neural network are described in the succeeding paragraphs.

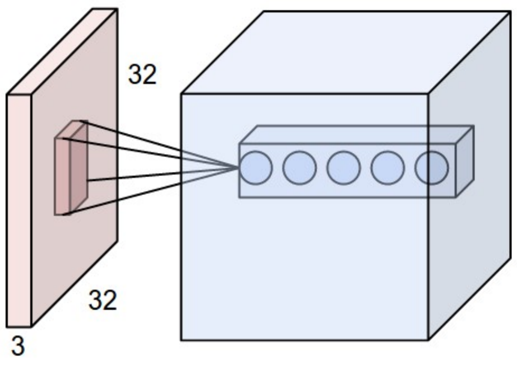


Figure 2.18: Convolutional Neural Network Internal [38]

**Input** When dealing with images, the input will be Red-Green-Blue (RGB) pixels. Thus, the total size of input will be $height * width * 3$. Taking an example of Canadian Institute for Advanced Research-10 (CIFAR-10) datasets which has $32 * 32$ size images, the total number of input in a single image will be $32 * 32 * 3$.

**Conv** Conv layer is a convolutional layer where spatial features are extracted by convolving a part of an image with a filter. The weights of such filters are learnable parameters. However, the number of filters each conv layer has can vary with each filter trying to learn distinctive characteristics of an image. The output of conv layer will be three dimensions $height * width * (numOfFilters)$. Padding can be done for keeping the same dimension along height and width. Taking an example of CIFAR-10, if we have 12 filters with proper padding in first conv layer receiving CIFAR-10 input, then the output will be $32 * 32 * 12$.

**Relu** Relu is an activation function popular in deep learning which is already discussed in section 2.3.3. The function is applied for each element from the output of CONV layer. There is no change in the size of output in this layer.

**Pool** If the same size of input is continued, the number of parameters will increase by many folds. The mechanism used to decrease the parameters for a deeper network is called pooling which helps in reducing the size of the input. Pooling also helps in improving the performance of the layer and decreasing the training time. It is due to this layer all the subtle features which are identified in initial stages gets dropped when size is decreased resulting in more specific features of an image getting identified in later stages. The common way of pooling is max pooling where the maximum value is selected from a window of fixed sizes such as $2 * 2$ or $3 * 3$. If $2 * 2$ is used in CIFAR-10 datasets, the size of the image will decrease to $16 * 16 * 12$. This layer, however, doesn't decrease the depth.

**Fully connected Layer** CNN extracts features from the images and uses them to classify. The process of features extraction and representation happens before the fully connected layer which is used for classification. It is the last layer of CNN and uses Softmax or SVM for doing actual classifications [48].

### 2.3.5 Deep Learning with Nvidia DIGITS

Nvidia's Deep Learning GPU Training System (Digits) [17] is an interactive platform for training deep neural network. It provides the facility of visual-

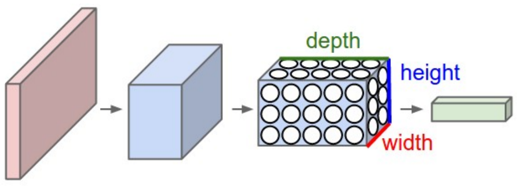


Figure 2.19: Convolutional Neural Network [38]

izing each step of deep learning. The platform is used for processing data, configuring the deep neural network, monitoring the training progress, visualizing the trained sets, monitoring the performance of the network in real time and managing multi GPU training. Performance of deep neural network can be monitored in real time.

**Dataset Creation** Dataset can be created and uploaded using Digits. Additionally, it provides information about the dataset such as how many training samples are available and what kind of data they are.

**Network Configuration** It provides facility to select standard networks such as LeNet, Alexnet or GoogleNet. These default networks can be used for training or can be modified according to the need. Options for setting solver parameters are available such as training epochs, the interval of snapshots, interval for validation run, batch size, type of solvers such as Stochastic Gradient Descent (SGD), ADAGRAD or NAG and learning policy and rate.

**Training Results** It provides real-time monitoring of training performance and accuracy. The model can be discarded or modified if performance is poor but can be retained if we are getting satisfactory results.

**Monitor Overfitting and Underfitting** It provides the graph of training and validation loss for each epoch. If the loss continues to decrease for training set while increasing for validation set, then the network has overfitted. However, if the loss is not increasing, but accuracy is also not improving, then the network is underfitting.

**Deployment** It provides an easy method to download the network and transfer to the deployment device which can be mobile, server or laptops [17]. In this thesis, deployment device used is Nvidia Jetson. Figure 2.20 summarizes the process of training and deploying using Nvidia DIGITS.

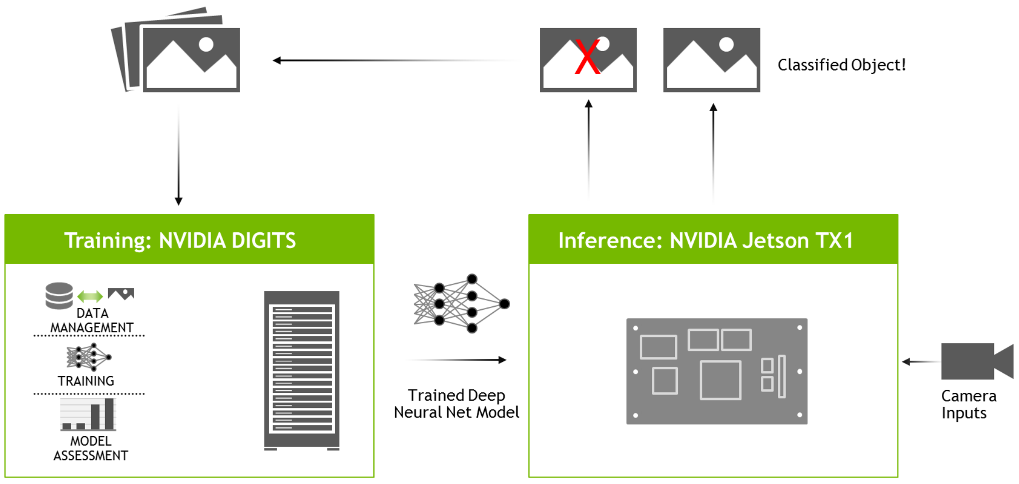


Figure 2.20: Training and deploying deep learning with DIGITS [17]

### 2.3.6 Caffe

Different frameworks are available for training and deploying deep neural networks such as Theano, Tensorflow, Torch, and Caffe. Each framework has their share of strengths and weaknesses, and it is out of the scope of this thesis to analyze each of them. For this thesis, Caffe is used as deep learning framework due to its low learning curve, wide acceptability, and permissive Berkeley Software Distribution (BSD) license [35]. It is developed in Berkeley Vision and Learning Center and is written in pure C++ with CUDA support which has the command line, Python and Matlab interfaces. In addition to providing deep learning framework, it also has an implementation for pre-processing and deployment. There are reference models, and examples as a part of the framework and plethora of trained models are available in model zoo[12]. Such models ease in developing the application through early prototyping, easy training and easy deploying.

For providing input, Caffe has support for LevelDB or Lightning Memory-Mapped Database (LMDB) database, Hierarchical Data Format 5 (HDF5) and raw image files. It also supports in-memory computation for python and C++ language. Data preprocessing such as creating LevelDB/LMDB from raw images, shuffling data to generate training and validation sets and generating mean-image can also be done with Caffe. It also includes data transformations tools such as image cropping, resizing, scaling, mirroring and mean subtraction.

The model definition which includes defining network architecture and training parameters in Caffe is done with the protobuf model format. The format is developed by Google which is strongly typed and human readable. Several loss functions are available for both classifications such as softmax and hinge loss, linear regression such as Euclidean loss and attributes/multi-classification such as sigmoid cross entropy loss. The available layer types are convolutional, pooling and normalization. Similarly, many activation functions such as ReLU, Sigmoid and Tanh are available.

After training, Caffe generates .caffemodel which can be quickly deployed. Furthermore, it provides support for detection with regional CNN and segmentation with fully connected CNN. The platform is easily extendable for python and C++ to add a new layer or loss function.

### 2.3.7 Reinforcement Learning (RL)

Reinforcement learning is another variant of learning where drone tries achieving some task by trial. Unlike supervised learning, reinforcement learning does not have any labeled data. In the absence of labeled data and cost functions, reinforcement learning uses reward function. The reward is a quantitative measurement of the outcome of an action performed. As shown in figure 2.21, reinforcement learning has several components such as agent, environment, action, reward, and observation. An agent is always trying to learn to achieve some goals. An environment is where such action is performed and can be observed by the agent. An action is a task performed by an agent to attain its goal. A reward is a scalar feedback signal indicating the performance level by the agent at any time step t. Observation is a state of the environment an agent can observe [6]. The agent always tries to get the best outcome by maximizing its reward function. The reward is cumulative, and an agent tries to maximize its cumulated reward. At each time step, there is an interaction between the agent and the environment. The agent executes an action, receives observation and scalar reward, whereas, the environment gets action, emits observations and rewards. RL agent has no information about the environment, but it learns about it through a continuous process of actions, observations, and rewards. The goal of an agent is to find the policy that maximizes the utility function. The policy is a series of actions and utility function governs the rewards function. Agent continually interacts with the environment which is unknown initially to learn more about it. For doing so, an agent must balance and perform two sets of opposing steps which are exploration and exploitation.

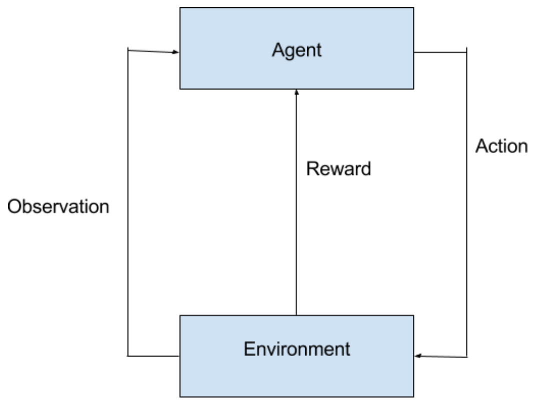


Figure 2.21: Reinforcement Learning

### 2.3.8 Deep Reinforcement Learning (DRL)

The deep reinforcement learning tries to leverage the concept of a deep neural network with reinforcement learning. By representing the utility function as a deep neural network, deep reinforcement learning tries to solve complex problems. Much research is ongoing such as [55] and [39] for using deep reinforcement learning to fly drones autonomously. It is different from supervised learning in that we do not have labeled datasets for training so that we can find errors and then back propagate them. Thus, the concept of policy gradient comes along. The rewards can be (0, +1, -1) which will modulate our gradient to 3 possible values. For zero reward, our weights remain the same.

### 2.3.9 AI for drone navigation

There is much ongoing research on using AI and auto navigation in drones. Few of the recent articles focused in the thesis are [55], [39], [30], [13] and [28]. The authors in [55] and [39] have used imitation learning for controlling the drones and avoiding the obstacles. Similarly, authors in [28], [13] and [30] have used feed-forward deep neural network for controlling the drones and avoiding the obstacles. The authors in [28] discuss creating a dataset from 11,500 crashes which are then used to train standard deep network. The network outputs binary classification and provides the information about how to prevent a crash. It is a simple self-supervised model which is also effective in an extremely cluttered environment. The authors in [13] provide a method of using a single forward facing camera which is fed into the trained CNN model for depth estimation. The trained model was tested in both real and simulated environment with satisfactory results. Similarly, authors in [30] trained the drone to perceive forest trails with a single monocular camera using the deep neural network. Furthermore, the dataset used to train the drone is freely available for download, and it is the same dataset used in this thesis for experimentation. Through imitation learning and recurrent neural network, authors in [39] are trying to train the UAV to fly across the simulated room in a cluttered environment. Authors in [55] also used a similar approach with forward-looking camera and imitation learning to navigate in a forest and controlled indoor environment.

## 2.4 Summary

The chapter introduced the fundamental background for this thesis. It presented the basic components, sensors and algorithms explored during the course of this thesis. The navigation unit is required for the drone to navigate from one point to another. Similarly, obstacle avoidance unit is required for safely navigating in the cluttered environments. The thesis will later provide integration of some obstacle avoidance sensors such as Garmin lidar lite and Realsense camera r200. A short introduction to AI is also provided and current research on the field on a drone is also summarized. AI will be used later in this thesis for better understanding the nature of an obstacle through image segmentation and navigation. Understanding of current research and solutions in autonomous cars can help us understand autonomy in drones as well. Autonomous cars have already solved many complex problems of navigation and obstacle avoidance. Next chapter will explore in more details how autonomy works in cars.

# Chapter 3

# Case Study: Autonomous Cars

Both automobile and technology companies such as Tesla, Google, Uber, Nissan, BMW, Ford, and Mercedes are all developing autonomous cars [47]. Many of these autonomous cars today have level 4 autonomy where an autonomous vehicle has complete control. These autonomous cars can be a huge source of inspiration and knowledge for applying similar technology for drones. Thus, the thorough understanding of architecture and applications for autonomous cars is one of the research focus of this thesis.

As defined by National Highway Traffic Safety Administration of US, autonomous driving has five levels of autonomy [23]. The levels are differentiated based on the degree of control and autonomy the vehicle has. The first level, also known as level 0, is a level where the car has no control on its own and is entirely controlled by the driver. The degree of control by an autonomous system increases from level 1 to level 4, and the autonomous system finally takes complete control in level 4. Such a vehicle operating in level 4 has the responsibility of all activities related to driving and executing functionalities such as safety-critical functions, parking, starting and stopping. The ability to perceive the environment in real time based on the input from multiple sensors enables vehicle of such autonomy. Some popular sensors used in autonomous cars are lidars, GPS, IMU, cameras, sonars, and radars. These sensors are used in localizing the vehicle and making real-time decisions for navigation. The task of autonomous driving, however, requires a high computation power to process a massive volume of sensor data.

As provided in [45], any autonomous system can be considered to have three components as shown in figure 3.1.

1. Sensing
2. Perception

3. Decision Making

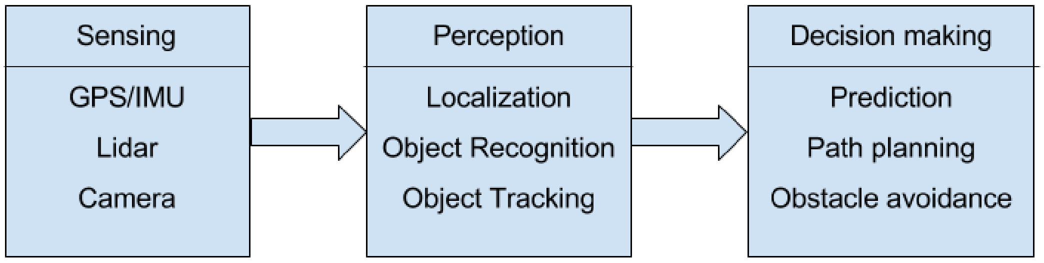


Figure 3.1: Autonomous Car architecture [45]

Sensing is merely the collection of data. Making meaning out of those collected data is perception. Finally, making a decision on what actions to take based on the semantics obtained from perception is decision making.

## 3.1 Sensing

Sensing is the collection of data from the sensors. As discussed earlier, an autonomous car is equipped with several sensors such as GPS, IMU, Lidar, Camera, Radar, and Sonar. However, each of the sensors has their strengths and shortcomings. GPS has good accuracy but has a lower sampling frequency whereas, IMU has a much higher frequency but its accuracy degrades with time. These shortcomings can be mitigated by fusing two together so that they can complement each other to provide accurate and real-time update required for autonomous driving. The accuracy of such system can be within the range of one meter [8]. This only applies to the scenario where the vehicle has a direct line of sight to the satellite. GPS signals get noisy in cities and forests, and no signals will be received in tunnels. This shortcoming can be overcome by using LIDAR or camera. Sensing in an autonomous vehicle is of two types: internal sensing and external sensing. Internal sensing is measuring vehicle's internal states, and external sensing refers to sensing the environment in which vehicle is navigating.

## 3.2 Perception

Sensing state accumulates data from the sensors, but collected data has no meaning until it is processed and semantic is drawn out of it. Understanding meaning of the data happens in the perception where vehicle processes the data from different sensors and try to perceive its environment. The first task of an autonomous car is to understand its surrounding and figure out where it is and where it must go. Such a process of mapping its location with respect to the environment is known as localization. It is achieved by data fusion from different sensors such as GPS, IMU, and Lidar. The particle filter is used to create a map out of Lidar data which can give real-time localization with 10 cm accuracy [45] even in an environment with many obstacles. Another important task autonomous vehicle must perceive is detecting the objects. Computer vision based technologies such as CNN are used for detecting objects near the vicinities of the vehicle. Furthermore, both dynamic and static objects should be tracked once detected. The vehicle should keep track of moving objects such as pedestrians, other vehicles or any other objects in the environment for safe driving. For such purposes, approaches using deep learnings are more effective than conventional computer vision approach for object tracking[45]. Tracking is done through series of temporal readings. Sensor fusion can also be used in this process to corroborate that the data from various sensors conclude to the same decision either about detection, localization or tracking[8].

## 3.3 Decision Making

After the vehicle is localized, objects are detected and tracked then an autonomous system can make informed decisions. Usually, the process of making a decision is based on Markov chains and probabilistic models. An autonomous vehicle must make some important decision regarding motion prediction, path planning, and obstacle avoidance. While driving on the highway, autonomous vehicle not only has to predict its movement but also of other vehicles. The difficulties of prediction increases in multi-lane highways and intersection of the roads. After predicting the motion of other vehicles, the vehicle has to plan the path for itself for full maneuverability [8]. The process of path planning involves generating a set of trajectories which are obstacle free and is not on a collision course with any other vehicles. The optimal path from the generated path is selected for steering. The problem of path search can be done either using brute force to search all possible paths or use probabilistic models. The task of planning the path also has to incorporate avoiding the obstacles in the vicinity of the car. There are two types of obstacle avoidance based on when the process of avoiding takes place. They are proactive and reactive. The first level of avoiding takes place with the proactive system which is based on prediction and metrics considered are time to collide and the minimum distance to obstacles. In this level of obstacle avoidance, obstacles are avoided by replanning the path. The second level of obstacle avoidance is reactive which use sensors such as sonar and radar. The data from such sensors are directly fed into the main control unit without going through the entire processing pipeline. The action taken at this level is more severe actions such as braking and moving sideways. It overrides current control for driving the cars to avoid obstacles.

## 3.4 Lidar vs Vision-based system

The state of the art solution for an autonomous vehicle is based on one of the two approaches: Lidar-based autonomous driving and camera-based autonomous driving [45]. Considering performance lidar based system is better in which lidar creates millions of data points in a second within a range of 200m. The lidar streams laser beam, measure reflection time from the surfaces and determines vehicle's distance from various objects in the environment. However, the cost of good Lidar system is prohibitively high costing up to several tens of thousands of dollars. Furthermore, raindrops and dust will create a lot of noises which will be inevitable while driving. An alternative to such a system is the vision-based autonomous system which is more economical. Instead of relying on particle filter like the one used in Lidar, it uses visual odometry which is based on feature description. To extract depth information, it uses stereo images. From the extracted features, key features are used to compare with the known map to localize itself in the world.

## 3.5 Lateral and Longitudinal Driving

Lateral and longitudinal driving is the way autonomous vehicle operates while driving. The lateral guidance keeps the car in current lane whereas longitudinal guidance tries to keep a safe distance to other cars in the front.
The crucial factors to keep in mind for autonomy are lateral and longitudinal control, functional distributions, conflict resolutions, fault propagation and isolation of system failures and cognitive complexity.

Main features of autonomous architecture should be service-oriented communication, a dynamic operating system which is real-time with fault tolerance with required safety and security[31]. Such design also provides a continuous connection to the backend to collect the data and use it for learning various mistakes and making decisions.

## 3.6 Fully convolutional neural network (FCNN) based control

Autonomous cars can be formulated as a single problem of driving with the fully convolutional neural network where FCNN issues steering command based on input from the front camera mounted on the car. As given in the PilotNet[9], Nvidia has trained a car to mimic a behavior of the driver. The training of PilotNet is done with images generated from three cameras mounted in front of the car and facing left, center and right. Input also contains corresponding steering commands issued by a human driver to drive the car. In contrast to an earlier approach where problems such as lane detection, pedestrian detection, and traffic signal detection are done in parallel and are trained explicitly to do so, PilotNet learns essential features embedded in the image based on the action taken by the driver. The research has shown that the module learns features such as lane markings, edges of the roads and bushes lining implicitly. End to end learning will eliminate the need for hand coding rules for an autonomous vehicle. The figure 3.2 shows the block diagram used by [9] for training. They have used three cameras and recorded steering wheel angle as input. Left camera and right camera are used to get center shifts so that vehicle can recover from drifts.

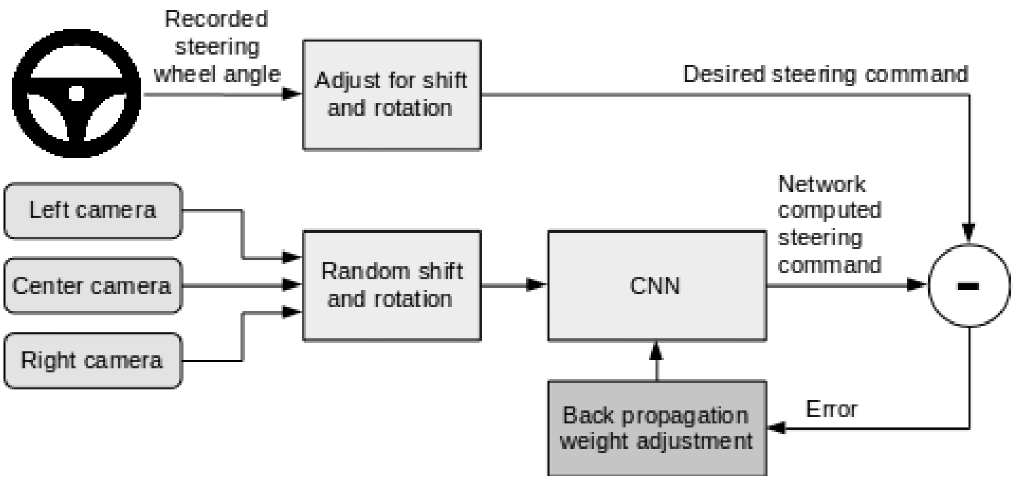

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error

Figure 3.2: Training End to End FCNN for autonomous driving [9]

Once training is done, network is deployed with a FCNN controlled autonomous car with a single center camera mounted on it. The deployed model issues steering wheel control command to keep the car on lane and to drive towards its destination. The process of deploying is shown in figure 3.3.

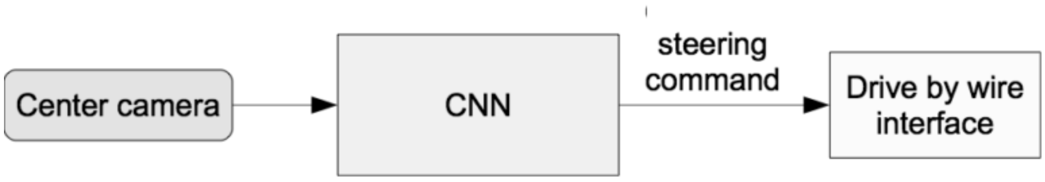


Figure 3.3: Deploying End to End FCNN for autonomous driving [9]

## 3.7 Summary

Sensing, perception and decision making are three most important steps of autonomous cars. Sensing collects sensor data, perception analyze those data and decision-making step makes an informed decision based on the analysis. Lidar-based system and Vision-based system are currently the most popular approaches for autonomy. Current trends and research are going in the direction of an end to end control of cars with the fully convolutional neural network. In such a system, the task of driving is considered to be a single problem of estimating steering commands instead of dividing the task of driving into subtasks such as lane detection and pedestrian detection. After understanding basics of different sensors and knowledge of autonomous cars, next chapter will introduce autonomy in quadrotors and different components associated with it.

# Chapter 4

# Quadrotor and Flying Principle

Chapter 2 discussed the foundation of different sensors for navigation and obstacle avoidance and chapter 3 discussed how state of art autonomous cars use autonomy. This chapter dives into understanding different components of quadrotors and how state of art flight stack available in the market handle the task of navigation. The chapter starts by introducing physics and kinematics required for flying a quadrotor. Then chapter presents brief introduction about hardware components and various software components of quadrotor for ardupilot flight stack. The drones are programmable due to the availability of different software stacks as well as properly defined mavlink communication between them. Mavlink is the protocol used by component developed for this thesis to communicate with flight controller. Kalman filter which is a state estimation and sensor fusion algorithm and used in state of the art solution is also discussed in this chapter.

## 4.1 Flying Principle

This section introduces the aerodynamics of quadrotor drones. By understanding these principles, the concept of how force and thrust interplay to make a drone fly and change directions is understood. Flying is moving against the gravity which has to be compensated. Quadrotor has four wings which are powered by motors and are capable of rotating in clockwise or counterclockwise and producing enough thrust for quadrotor to fly. Motors and actuators on the drone produce forces and torques which induce linear and angular acceleration respectively. Force is a vector quantity which can be easily calculated using vector arithmetic. Gravity, thrust, and friction are some of the few forces which need to be considered for quadrotors. The application of these force on a body causes it to accelerate. Force on a lever

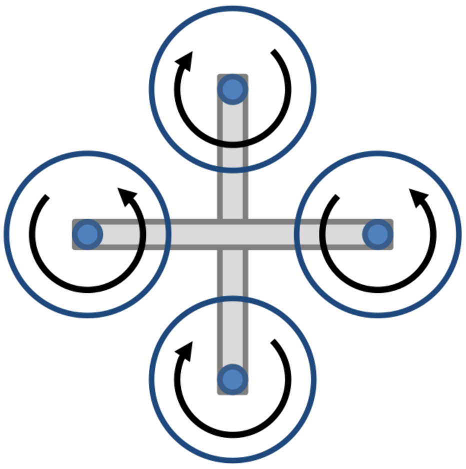


Figure 4.1: Four rotors in Quadrotor [58]

induces a torque. Torque results in angular acceleration $\alpha$ given by equation 4.1 and 4.2.

$$J\alpha = \Gamma \tag{4.1}$$

$$\Gamma = rF \tag{4.2}$$

Each of the propellers induces force and torque. Total thrust towards vertical direction is sum of all the forces applied in each propeller which is given by equation 4.3 and shown in figure 4.2.

$$F_{thrust} = F_1 + F_2 + F_3 + F_4 \tag{4.3}$$

Attitude changes when opposite motors generate unequal thrust. This induces torque and which is given in equation 4.4 and induced angular acceleration is given by equation 4.5.

$$\Gamma = (F_1 - F_3) * r \tag{4.4}$$

$$\alpha = J^{-1}\Gamma \tag{4.5}$$

Each propeller induces torque due to rotation and the interaction with the air and are additive in nature. The resulted torque is given in equation 4.6.

$$\Gamma = \Gamma_1 - \Gamma_2 - \Gamma_3 - \Gamma_4 \tag{4.6}$$

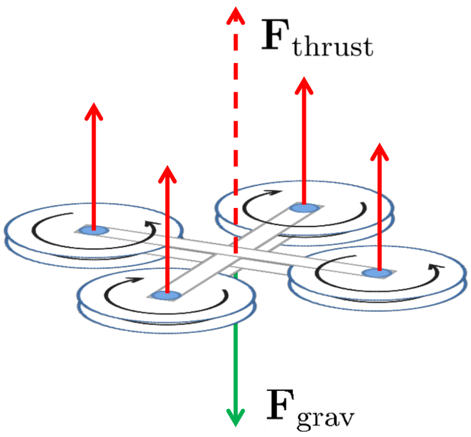


Figure 4.2: Thrust on four rotors [58]

In summary, thrust on the rotors compensates for gravity. When enough thrust is provided by actuators to compensate the gravity, quadrotor can hover. When the force applied to each actuator is not equal, it induces torque in the drone. For loitering on a fixed position in the air, thrust should compensate the gravity, and total torques on all four rotors should sum to zero.
However, when thrust in the drone increase and is more than that of gravity, then drone can gain altitude and rise as shown in figure 4.3. In the same way, when thrust is decreased to be less than gravity, the drone can descend. However, if the different amount of thrust is provided to the four arms of the drone, the drone will rotate due to induction of torque. So, to prevent that from happening while ascending and descending, it is better to apply equal thrust in all the arms of the drone.
The generation of torque by the applications of unequal amount of thrust in different arms of the drone can be used for moving drone sideways, forward and backward. When the thrust of the motor is increased in the front and back rotor while maintaining or decreasing thrust on the left and right, the drone will turn left. Similarly, when the thrust on the motor is increased on the left and right while maintaining or decreasing the thrust on the front and back, the drone will turn right. This process is illustrated by figure 4.4.
For the drone to move forward, thrust on the motor on the back must be the highest, thrust on the motor on the sides has to increase while the thrust on the motor in front is decreased. Similarly, for the drone to move backward,

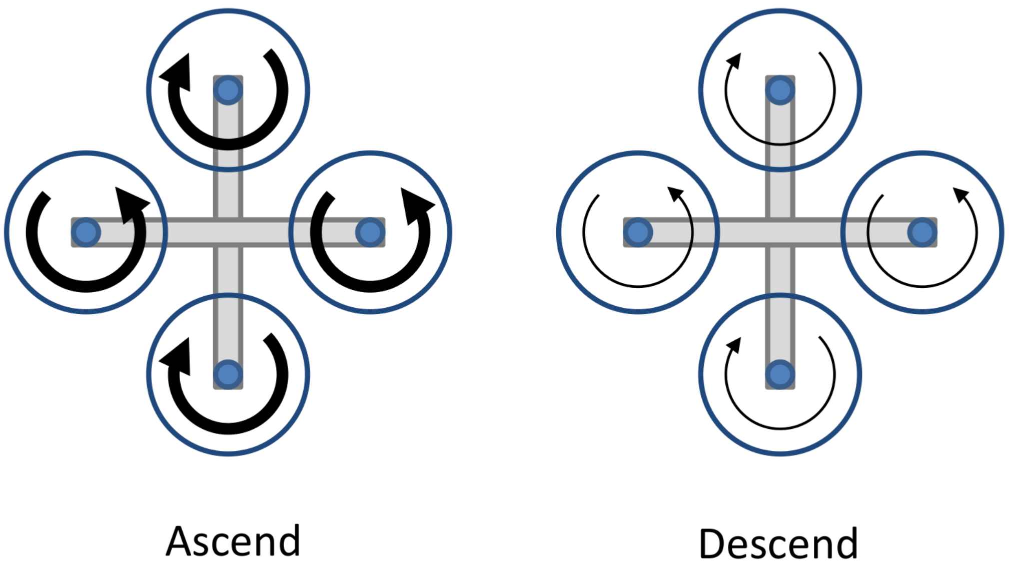


Figure 4.3: Ascend and Descend [58]

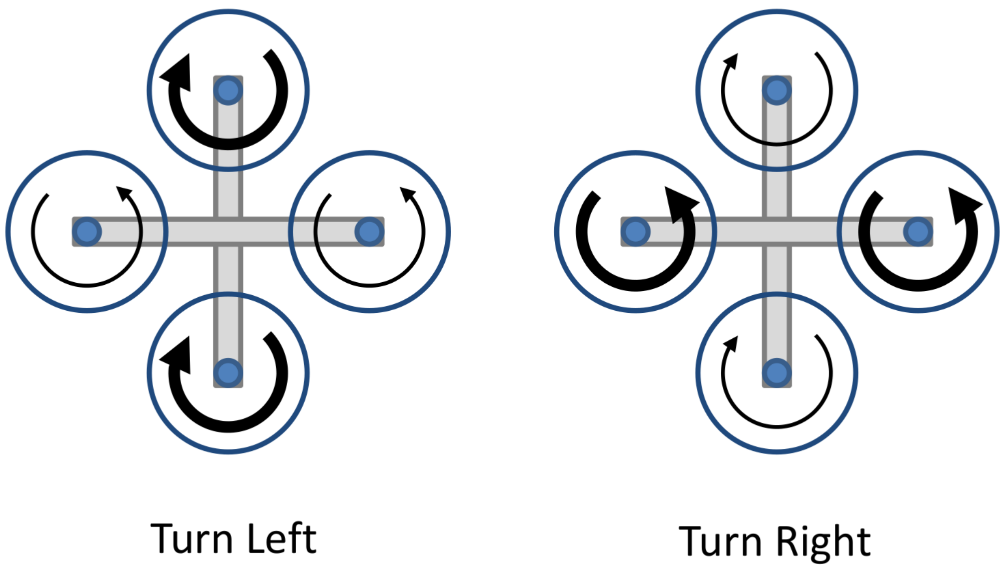


Figure 4.4: Turning Left and Turning Right [58]

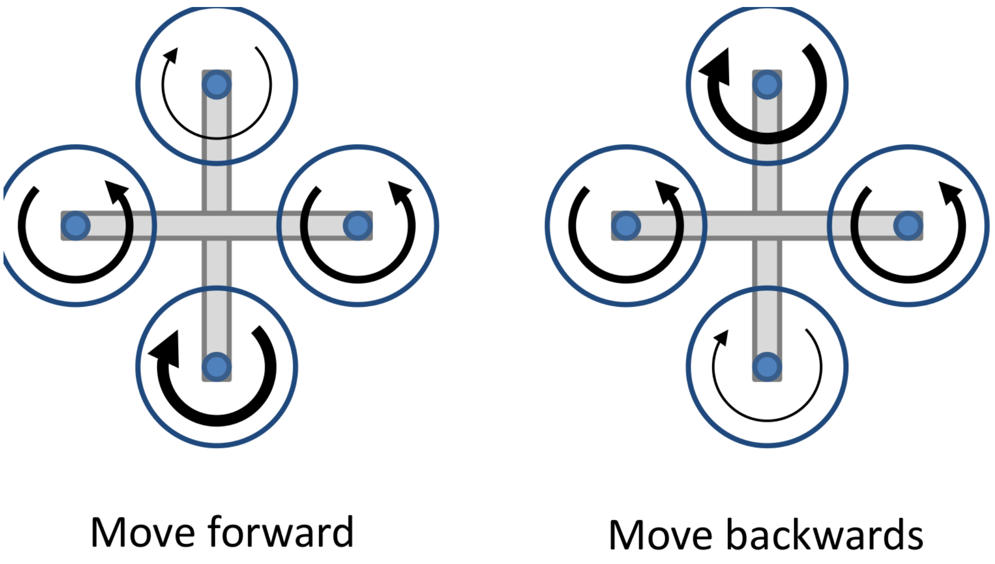


Figure 4.5: Move Forward and Move Backward [58]

thrust on the motor in front must be the highest, thrust on the rotor on the sides has to increase while thrust on the rotor in the back is decreased. Figure 4.5 summarizes this process. As shown in figure 4.6, for the drone to move left thrust on the motor on the right must be the highest, thrust on motor in the front and back have to increase while the thrust on the motor in the left is decreased. Similarly, for the drone to move right, thrust on the motor in left must be the highest, thrust on the rotor in the front and back have to increase while thrust on the rotor in the right is decreased.

## 4.2 Hardware Components

Hardware components comprise of all the mechanical and electronic components needed to build a quadrotor which may vary and differ from one manufacturer to another. However, there are few fundamental components needed by any drones for flying. Some of these are frames, motors, electronic speed controller (ESC), propellers, battery, flight controller, Radio Controller (RC) Transmitter/Receiver and other optional parts. The frame is a structure that holds all other components together. It is designed to be strong, rigid but lightweight. Motors are used to rotate propellers and can be brushless or brushed dc motors. ESC is used to control the speed of the motor. Propellers are the wings of quadrotors, and when they rotate,

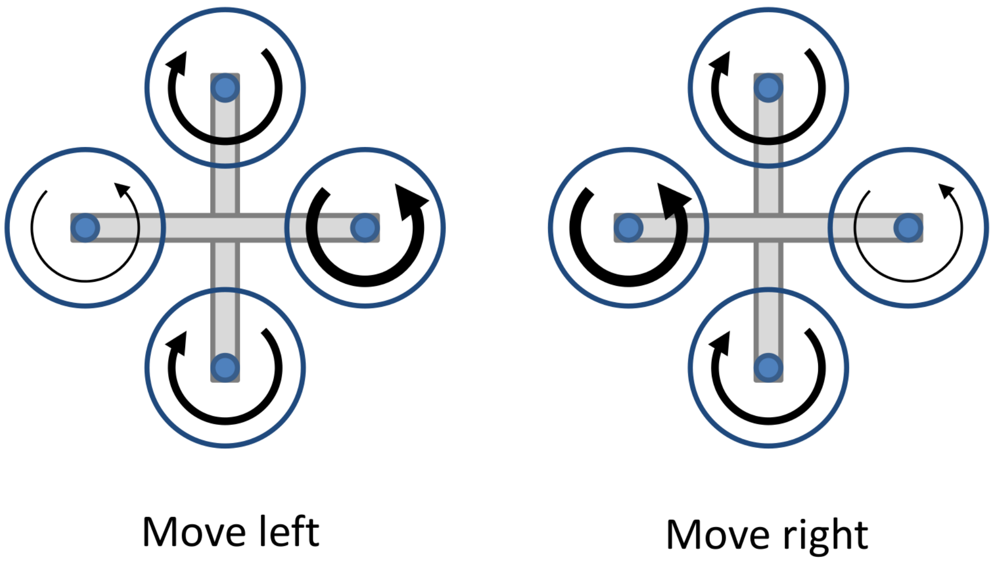


Figure 4.6: Move Left and Move Right [58]

they generate enough thrust for lifting the vehicle[44]. Quadrotor can have additional components such as camera, sonar, radar, and lidar depending on the use cases.

## 4.3 Software Components

Software components programmatically define a controller for flying a quadrotor. For designing a controller for the drone, it is of vital importance to understand fundamental software components of the drone. Various operations such as braking, changing the route, and bringing the drone back to base can be performed in a system with obstacle avoidance and autonomous navigation. Thus, a thorough understanding of software stack is needed for implementing such system. The basic architecture of different software components in high level is shown in figure 4.7. The main software components of automated drones are flight controller, software on companion computer and a remote client in the form of ground control station. The flight controller is both a hardware and software component which has a real-time operating system and is used mainly for navigations. Companion computer complements flight controller by performing accessory activities such as collision avoidance, computer visions, and payload maintenance. Similarly, the

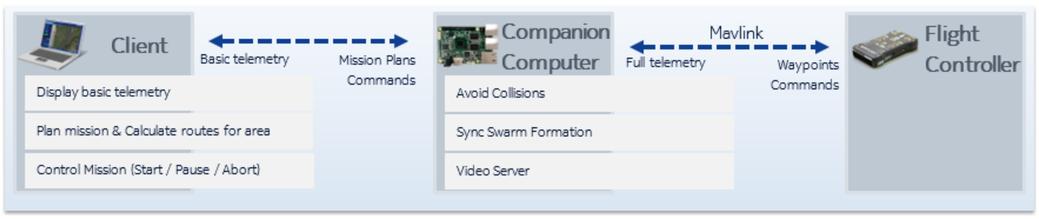


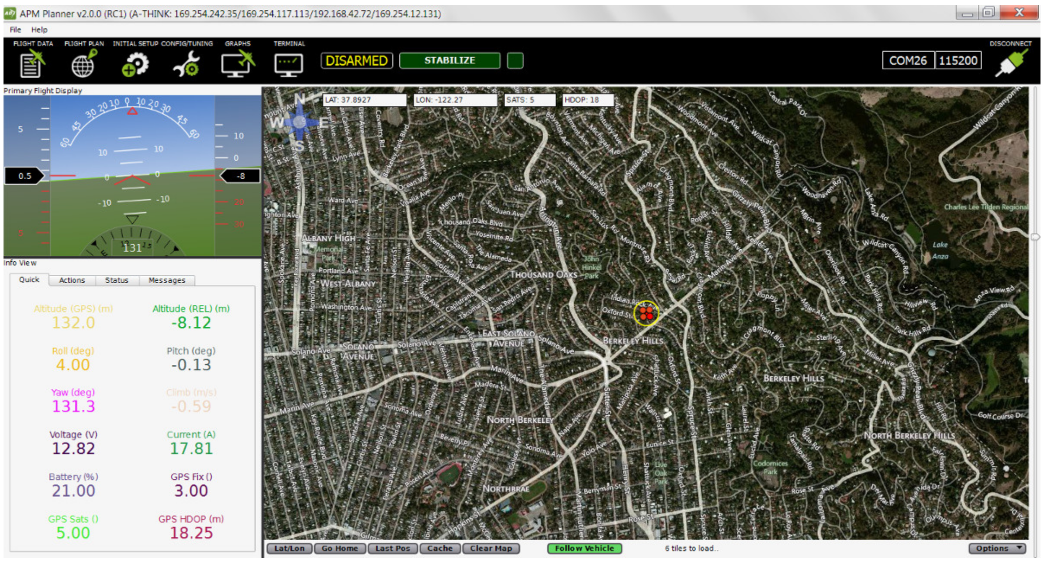
Figure 4.7: Different Software Components

remote client can be radio controller or ground control station for remotely controlling the drone.

**Ground Control Station (GCS)** Ground control is a remote station for programmatically controlling the drone through the built-in user interface. Few functionalities of a ground control station are controlling drones in flight, uploading and downloading missions, sending commands and updates, receiving telemetry, displaying the map and monitoring live streams. There are many open sources ground control stations such as apmplanner2, mavproxy, QGroundControl and UgCS. Apmplanner2 was a preferred GCS for this thesis as it has better performance in Linux environment where the thesis is done. Besides, it is an open source GCS for any mavlink based autopilots such as PX4/pixhawk and APM[61]. The figure 4.8 and 4.9 are screenshots from apmplanner2.

**Companion Computer (CC)** There are various choices of companion computers available in the market. The choice of companion computer should be based on the performance required for the system. The companion computer which has less computing load such as rerouting the sensors data, sending few commands to the drones, basic computations, and basic computer vision can be powered by less powerful hardware. But if intensive computation and near real-time calculation is needed then much powerful companion computers should be used. Raspberry Pi, ODroid, Intel Edison, Upboard, BeaglePilot, and Turnkey can be one of the solutions for companion computer. For this project, the choice of companion computer was Raspberry Pi, Upboard and Nvidia Tx2 for development and it progressed chronologically as the requirement for the computation increased.

**FlightController** Flight Controller as discussed above contains both hardware and software stacks for running real-time operating system. Pixhawk



APM Planner v2.0.0 (RC1) (A-THINK: 169.254.242.35/169.254.117.113/192.168.42.72/169.254.12.131)

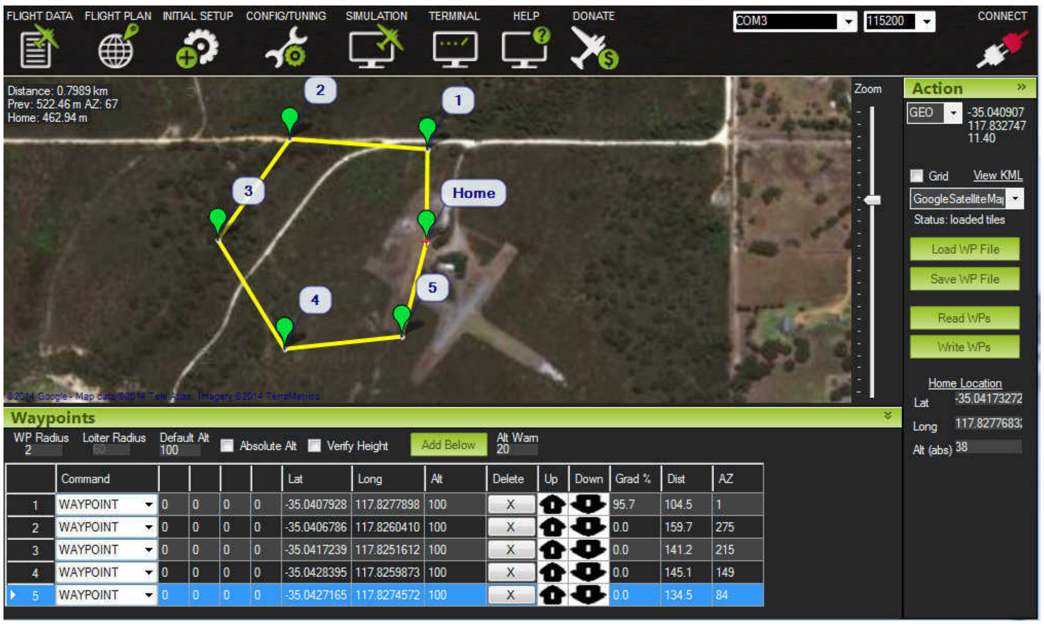
DISARMED STABILIZE COM26 115200

LAT: 37.8927 LON: -122.27 SATS: 5 HDOP: 18

6 tiles to load...

| | |
|---|---|
| Altitude (GPS) (m) 132.0 | Altitude (REL) (m) -8.12 |
| Roll (deg) 4.00 | Pitch (deg) -0.13 |
| Yaw (deg) 131.3 | Climb (m/s) -0.59 |
| Voltage (V) 12.82 | Current (A) 17.81 |
| Battery (%) 21.00 | GPS Fix () 3.00 |
| GPS Sats () 5.00 | GPS HDOP (m) 18.25 |

Figure 4.8: Apm planner 2 home screen [61]



COM3 115200

Distance: 0.7989 km
Prev: 522.46 m AZ: 67
Home: 462.94 m

GEO -35.040907 117.832747 11.40

Grid GoogleSatelliteMap
Status: loaded tiles

Home Location
Lat -35.04173272
Long 117.8277683
Alt (abs) 38

WP Radius 2 Loiter Radius Default Alt 100 Absolute Alt Verify Height Alt Warn 20

| | Command | | | | | Lat | Long | Alt | Delete | Up | Down | Grad % | Dist | AZ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | WAYPOINT | 0 | 0 | 0 | 0 | -35.0407928 | 117.8277898 | 100 | X | ⬆ | ⬇ | 95.7 | 104.5 | 1 |
| 2 | WAYPOINT | 0 | 0 | 0 | 0 | -35.0406786 | 117.8260410 | 100 | X | ⬆ | ⬇ | 0.0 | 159.7 | 275 |
| 3 | WAYPOINT | 0 | 0 | 0 | 0 | -35.0417239 | 117.8251612 | 100 | X | ⬆ | ⬇ | 0.0 | 141.2 | 215 |
| 4 | WAYPOINT | 0 | 0 | 0 | 0 | -35.0428395 | 117.8259873 | 100 | X | ⬆ | ⬇ | 0.0 | 145.1 | 149 |
| 5 | WAYPOINT | 0 | 0 | 0 | 0 | -35.0427165 | 117.8274572 | 100 | X | ⬆ | ⬇ | 0.0 | 134.5 | 84 |

Figure 4.9: Apm planner 2 creating mission interfaces [61]

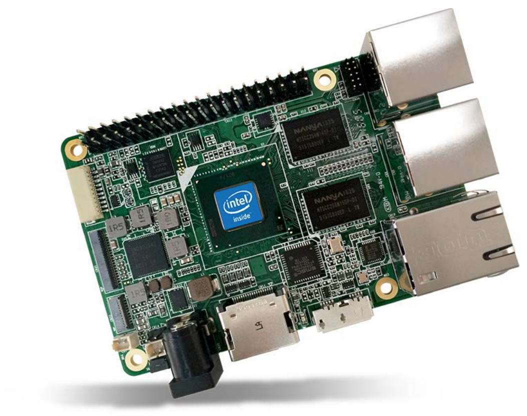


Figure 4.10: UpBoard [4]

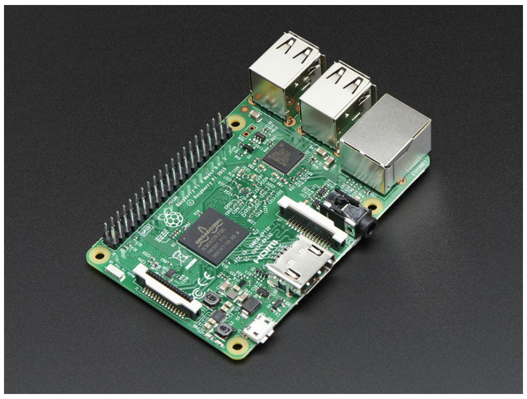


Figure 4.11: Raspberry Pi [3]

from 3DR robotics was the choice of the hardware. It can mount different software stacks for controlling drone. The most popular ones such as pixhawk flight stack and APM's ardupilot were considered for this thesis. APM's ardupilot was chosen for this thesis as it is more flexible, popular, safe and supports different hardware[66]. Ardupilot has support for wide va-

Figure 4.12: Pixhawk [2]

riety of hardware such as Pixhawk, Parrot Bebop, Beagle Bone, Intel Aero, Erle-Brain, PixRacer and Qualcomm SnapDragon. It is capable of controlling autonomous vehicles of different types such as helicopters, multirotor drones, fixed-wing Vertical Take Off and Landing (VTOL) model aircraft, rovers, boats, submarines and antenna tracker[60].

## 4.4 Mavlink

Mavlink is the communication protocol between different components of a drone. It might be between payloads and flight controller, between companion computer and flight controller or between the ground control station and flight controller. It is specifically designed for Mav(Micro Aerial Vehicle) and is very lightweight with header only message library. It packs C-structs and transmits through serial channels to different endpoints. It allows to check the message content, detect lost messages while only needing six bytes overhead for each packet. The protocol was designed for transmission speed and safety. Understanding and implementing Mavlink is important for developing a control system for the drones. The commands and messages are sent and received based on mavlink protocol. Basic frame of mavlink is shown in figure 4.13 and each fields is described in table 4.1.

There are several mavlink messages. Understanding them is vital for sending proper commands to the mav.

| **Byte Index** | **Content** | **Value** | **Description** |
|---|---|---|---|
| 0 | Start byte of packet | V1.0:0xFE, V0.9:0x55 | This value signifies the start of mavlink message. |
| 1 | Length of payload | 0-255 | It gives the length of payload. Since it is one byte, payload length can only be 0-255 bytes. |
| 2 | Packet Sequence | 0-255 | Each component keep counting of mavlink messages sent. It is useful for detecting packet loss. |
| 3 | System ID | 1-255 | Each drone in the network and GCS is uniquely identified by a system id. Since it is one byte, the number of drones in one network is theoretically limited to 255. |
| 4 | Component ID | 0-255 | Same system such as drone can have several components attached to it. Each of this component can send mavlink messages and are uniquely identified within a system with a component ID. |
| 5 | Message-ID | 0-255 | Mavlink has different message type. Based on the type of message, the payload is decoded accordingly. Message ID is an identifier for those messages. |
| 6 to n+6 | Data | 0-255 bytes | Actual mavlink message. All the component before this are header |
| n+7 to n+8 | Checksum ( low byte, high byte) | | This is for checking the correctness of the mavlink packet. |

Table 4.1: Mavlink Packet Field Description [1]

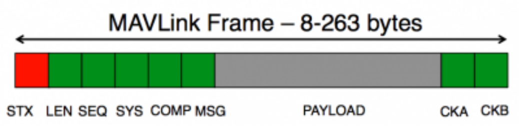


Figure 4.13: Mavlink Packets [1]

## Mavlink messages

1. MAVLINK_MSG_ID_REQUEST_DATA_STREAM This message is used to request the stream of data from the autopilot. Requested data can be sensors, RC channels, GPS position, status or the combination of them.

2. MAVLINK_MSG_ID_COMMAND_LONG This message is used to give commands to the autopilot. Several commands are supported. Types of commands used in the mission are navigation commands, DO commands and condition commands.

3. SET_MODE It sets the different mode of operations for the drone. Few supported modes for ArduCopter are

   - STABILIZE The mode is manual airframe angle with a manual throttle.
   - ACRO The mode is manual body-frame angular rate with the manual throttle.
   - ALT_HOLD The mode is manual airframe angle, but the throttle is automatic.
   - AUTO The mode is fully automatic waypoint control using mission commands.
   - GUIDED The mode is fully automatic fly to coordinate or fly at velocity/direction using GCS direct commands.
   - LOITER The automatic horizontal acceleration with the automatic throttle.
   - RTL The state automatically returns to launch.
   - CIRCLE This mode automatically circles a position with the automatic throttle.
   - LAND
   - DRIFT

- SPORT
- FLIP
- AUTOTUNE
- POSHOLD This mode automatically a position with manual override with an automatic throttle.
- BRAKE Full brake using inertial/GPS system with no user input.
- THROW
- AVOID_ADSB
- GUIDED_NOGPS

There are many other mavlink messages which were implemented for properly communicating with the drone.

## 4.5 Kalman Filter for Navigation

Kalman Filter is an optimal estimator which estimates states even from inaccurate and uncertain observations. The process is recursive where processing is done after every arrival of new measurements. The mean square error of estimated parameter is minimized for Gaussian noises. With only mean and standard deviation of the noise available, it can only estimate linear system. The popularity of Kalman filter lies in its optimality characteristics, online real-time processing, easy formulation and inversion free measurement equations. In summary, Kalman filter can optimally estimate only linear dynamic system with additive white noise. As discussed in section 2.1.4, Kalman filter can have motion and sensor model. The only difference is the system and noises are modeled to be linear Gaussian. Without undergoing the thorough derivation of Kalman filter which can be found in [64], Kalman filter can be summarized as having two steps. At each time step, we should apply

1. Motion model which is also known as prediction step
2. Sensor model which is also known as correction step

**Extended Kalman Filter (EKF)** For nonlinear systems which cannot be estimated by basic Kalman filter, the algorithm can be extended so that Kalman filter can be applied in such system as well. It is done by approximately linearizing the non-linear system with Taylor expansion. Taylor expansion can help to linearize the nonlinear equations approximately. After expanding and approximately linearizing the equations, Kalman filter can be applied, and such a system is called extended Kalman filter. Extended Kalman Filter is used by quadcopter for estimating position, velocity and angular orientation of the vehicle based on input from the compass, accelerometers, gyroscope, GPS, barometric pressure, and airspeed. EKF is advantageous over other filter algorithms because while fusing all available measurements it can reject measurements with significant errors [41].

## 4.6 Obstacle Avoidance in ardupilot

Obstacle avoidance is one of the most critical aspects of any autonomous vehicles. When flying in the real world, drone faces many obstacles in the form of buildings, walls, trees, poles, wires, and other static and dynamic objects. Without the means to locate these objects, a drone will crash into them destroying itself and likely cause danger for others. Typical sensors for identifying such objects are lidars, radars, sonars, and cameras. Each of these sensors has distinct characteristics and is suitable for a different scenario. Better results can be obtained by fusing different sensors and exploiting benefits from each one of them.
Ardupilot, the flight controller used for this thesis, has some basic support for avoiding the obstacles as described in [62]. It requires installing obstacle avoidance sensors and passing the data as mavlink distance sensor message to the flight controller. The avoidance is only supported in loiter mode, and changing the algorithm to suit custom needs is difficult. Similarly, Intel has developed an obstacle avoidance library [14] for its aero board using the Realsense camera. However, it also limits the use of any other sensors or algorithms for avoiding the obstacles. In addition, several private drone companies have implemented obstacle avoidance system tailored to their specific needs. [28] and [36] have implemented obstacle avoidance using reinforcement learning, but most of them find correct path and avoid obstacle only in the environment where they have been trained.

## 4.7 Summary

This chapter introduced how quadrotor can fly. Ardupilot is used for this thesis which is one of the advance flight stack available in the market today. There are several software components in the drone which makes the drone programmable. Kalman filter is used as an estimator and sensor fusion algorithm in ardupilot. Ardupilot also has a basic implementation of obstacle avoidance. Now, next chapter will discuss in detail about the shortcomings of current avoidance system in ardupilot and how a better design can improve such shortcomings. Also, the chapter will discuss the methodology and data used in this thesis.

# Chapter 5

# Methodology and System Design

This chapter discusses the shortcomings of current obstacle avoidance system available in ardupilot. Then it provides a high-level architecture design showing how a better design can mitigate such shortcomings. It also summarizes the methodology used in developing such system.

## 5.1 Current solution

The current solution of obstacle avoidance system available in ardupilot discussed in 4.6 has limited functionalities. Following limitations were identified in the system

**Works only in loiter mode** Current obstacle avoidance system provided by ardupilot only works in loiter mode. However, we needed a system to operate in any mode of operation.

**Support for limited types of sensor** The ardupilot obstacle avoidance supports only the range based sensors but not HD or depth camera.

**Extension to new sensors** It is not easy to extend the current library to new sensors and algorithm. The entire code needs to be recompiled and new binary has to be generated.

**Licensing of the library is GPL** Due to General Public License (GPL) licensing, modifying the code has legal implications if there is any intention of using it commercially.

Thus, we identified a need for building a versatile obstacle avoidance library which can be easily extended to support different sensors and algorithms as per the need. Developing a software framework for such library was the first step in building a robust obstacle avoidance system. Such framework requires only a few changes for addition or deletion of new sensors, hardware or algorithms.

## 5.2 High level architecture

**Choice of programming language for building library** Initial proof of concept was built with python. Despite being easy to use and quick to build, it had some performance issues. Obstacle avoidance is a time sensitive task which requires drone to take a quick action so we decided to use C++ as the main language to build the library. The exact benchmark with performance benefit can also be found online at [27]. After selecting the language of choice for programming, the design of the framework was done next.

The high-level design of the system is summarized in figure 5.1. Such a system has an interface for connecting to obstacle sensors of types range or image and is able to read data from such sensors. The system should have a mechanism to process the collected data to decide whether a valid obstacle is present in the vicinity of the drone. Finally, the system should also communicate with flight controller if it needs to send a command when an obstacle is detected. Based on the above basic needs, obstacle avoidance system has three modules interacting with one another. The three modules as shown in the figure 5.2 are the mavlink communication module, obstacle sensor module, and sensor fusion module. These three modules interact with one another for making a smart decision regarding the presence or absence of obstacles.

### 5.2.1 Sensor Module

Obstacle Avoidance sensors are connected to companion computer which is the main computing resource for detecting and avoiding obstacles. The sensors are connected to the companion computer through I2C, serial, PCI, USB or CAN bus. The sensor module provides the software interface for connecting with the obstacle avoidance sensors and accumulating data. This module is similar to sensing discussed in section 3.1 for autonomous cars. The module creates an abstraction for connecting to rest of the library and hides unnecessary details such as types of sensors or hardware connectivity. Such abstraction provides uniformity in the design of the interface for different

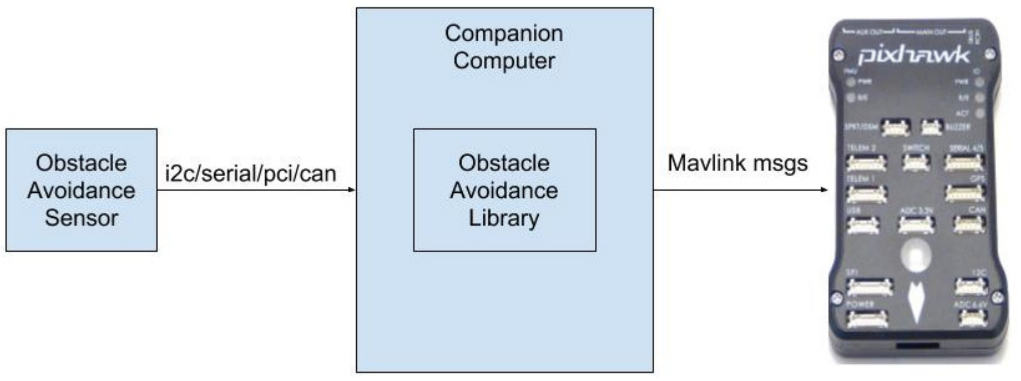


Figure 5.1: High Level Architecture of Obstacle Avoidance System

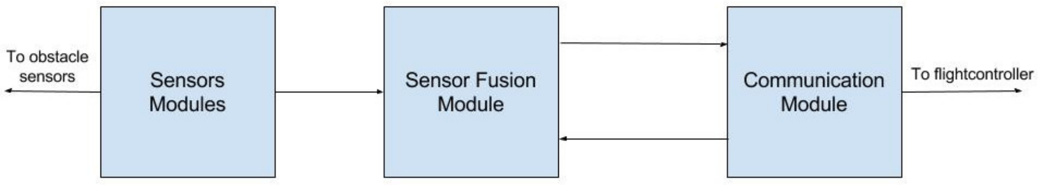
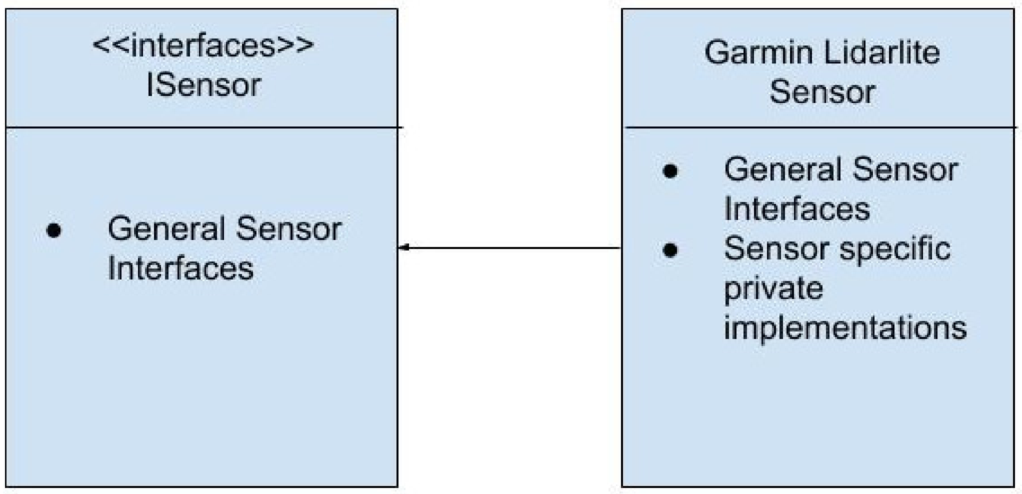


Figure 5.2: Main Modules of Obstacle Avoidance Library

sensors. Each sensor implements such abstractions as a sensor interfaces in the code. Sensor fusion module talks to sensor modules only through these interfaces. Each sensor module is a separate thread which collects the data from respective sensors and updates sensor fusion module with the new data. The format of this data should be clear to sensor fusion module, and thus, the sensor module includes metadata in the header which provides the type of data getting updated. In addition to accumulating the data, the module filters collected data with sensor-specific information. For example, different sensors may have diverse ways to represent an absence of an obstacle or minimum distance for detection, so it converts such reading to the generic absence or minimum distance as understood by sensor fusion module. It also has a responsibility of converting data into correct scale such as centimeter or meter as required by sensor fusion module. The basic Unified Modeling Language (UML) diagram for implementing a sensor is shown in figure 5.3.



<<interfaces>>
ISensor
General Sensor Interfaces
Garmin Lidarlite Sensor
General Sensor Interfaces
Sensor specific private implementations

Figure 5.3: Sensor Interfaces and Implementation for Garmin lidar lite

### 5.2.2 Mavlink Communication module

This module serves to communicate with flight controller. Since flight controller only communicates with mavlink protocol, this module performs decoding and encoding of mavlink messages. Rest of the modules in the obstacle avoidance library are agnostic about mavlink protocols. Beside encoding and decoding mavlink messages, it establishes software connection to flight controller either through serial or UDP interfaces. Different information about the drone such as current position in North East Down(NED) format or GPS data in the Geodetic coordinate system, the current pose of the drones such as roll, pitch, and yaw angles are provided by this module. Other useful information such as velocities of the drone with respect to the ground or air and mode of operation of the drone is also available. This information serves as a basis for making a decision regarding presence or absence of obstacles and finding the evasive maneuver. It also takes command from sensor fusion library and converts the command into a mavlink message and send it to the flight controller.

### 5.2.3 Sensor Fusion Module

Sensor fusion module is the center or brain of the entire obstacle avoidance library. The module decides whether to stop the drone, bring it back or move the drone around the obstacles. As shown in figure 5.2, it connects to both sensor module and a mavlink communication module. It implements a factory method for creating sensor class. Sensor classes are generated based on the configuration files. The sensor fusion itself will not access the sensor implementation directly but through the sensor interfaces. For receiving sensor data, it implements a listener method and subscribes to sensor data from each sensor instantiated from the factory class. Once subscribed, it keeps on collecting data from sensor class as long as it is not unsubscribed. On the other hand, it gets telemetry data including its position, velocity, angular acceleration and linear acceleration data from mavlink communication modules. The pose of the drone including roll, pitch, and yaw provides information about which direction the obstacle avoidance sensor is pointing at. Finally, the available data from different sensors and flight controller are fused with the sensor fusion module. The fusion algorithm can be chosen according to the need for the implementation and can be easily extended. The input from different sensors can be fused using different filters such as Kalman filter as described in section 4.5 or Bayes filter algorithm described in section 2.1.4. The complexity of such fusion algorithm can vary from simple algorithms to the complex ones. The fusion algorithm can be easily extended by overriding it if the available implementation does not support the complexity of the system.

## 5.3 Methodology

Unlike cars which can carry heavy payloads and sensors, quadrotors have restrictions on size and weight of the sensors it can fly with. So, the first step was selecting the sensors for the obstacle avoidance. There are many models and types of obstacle avoidance sensors available in the market. For the purpose of this thesis, following sensors were selected

1. Garmin Lidar Lite
2. Realsense R200 camera
3. USB HD camera

Garmin Lidar lite is compact and lightweight and is an ideal for a quadrotor. Despite having a low angle of resolutions, it has simple interfaces and is easy for quick development. Similarly, the concept of avoidance using depth camera was verified with R200 camera. It is one of the compact depth camera available in the market and has a low weight which comes with a Software Development Kit (SDK) for obstacle avoidance sensor development. Similarly, an HD camera was selected so as to test computer vision task of image segmentation. After selecting the component, hardware interfaces were designed and connected to the companion computer. Understanding the protocol to communicate and acquire data from each of the sensors is necessary. Once this is done, set up is ready for development. Each of the sensor development involved two steps

1. Development in a simulator
2. Testing in real environment or drone

The cost and time required for development are hugely reduced with appropriate simulation environment for testing. Deploying the software without testing the functionalities can be highly risky leading to the damage of the equipment and loss of time to debug the issues. Simulators used for this thesis are ardupilot Software in the Loop (SITL) and gazebo which are summarized in succeeding paragraphs.

**Ardupilot SITL[63]** It is a software in the loop simulator which can simulate quadrotor without any hardware. Ardupilot sitl is easy to use and is compatible with the mavlink message.

**Gazebo[26]** It is a tool for simulating robots. It has support for a plethora of vehicles, drones, and robots. In addition, it is easily extendable to support new bots and sensors.

## 5.4 Datasets Used

Due to the limitation of time, datasets used in this thesis are open dataset available to be downloaded from the internet. It needs a substantially large amount of time for collecting, processing and augmenting data if it is collected personally. Also, size of the datasets should be substantially large to ensure that it does not overfit the model. The succeeding paragraphs introduce the datasets used in this thesis.

**Aerial Drone Datasets** The aerial drone dataset is open dataset provided by [51]. This dataset is used for separating ground terrain from the sky. The dataset represents the First-Person View emulating vantage point of a drone in flight. It has two classes sky and terrain. The sky is represented with blue color and terrain is represented with green.

**Forest Trail Datasets** The datasets used for testing navigation in the forest trail with an end to end neural network is Forest Trail datasets. Like the autonomous cars discussed in [9], training data was collected with three cameras facing center, left and right along the trail of the forest. Cameras are mounted on a pole of length 1m with certain lateral offset. Data collected from such setup is used for detecting the rotation and lateral offset of the drone. The data used in this project was from [30] and it contains video of 30 minutes from 3 cameras which are used for training.

## 5.5 Summary

Obstacle avoidance library available with the ardupilot is not robust and has several shortcomings. To mitigate those shortcomings, a system is designed for obstacle avoidance. The designed framework was implemented next and integrated with obstacle avoidance sensors. The selection of obstacle avoidance sensors for testing was not easy either. With the restrictions on weight and size, we decided to test the system with Garmin Lidar lite, Realsense r200 camera, and USB HD camera. The developed framework was first tested and verified in the simulator before deploying it to the real environment. Finally, for experimenting with artificial intelligence, open datasets are used which were introduced in this chapter. Next chapter will be discussing an implementation of the design discussed in this chapter and interfacing with sensors for obstacle avoidance and navigation.

# Chapter 6

# Implementation

This section discusses implementation and outcome of various experiments done during the course of this thesis. The initial task was the implementation of the design discussed in chapter 5. The core parts of the mavlink communication module and sensor fusion module were developed first and then integrated with obstacle avoidance sensors. This chapter presents the experimentation done with different obstacles avoidance sensors such as Garmin lidar lite and Realsense r200. Furthermore, it provides the possibility of using the camera as an obstacle avoidance sensor with image segmentation. The possibility of navigation using end to end deep neural network is provided next.

## 6.1 Obstacle Avoidance with Garmin Lidar Lite

As discussed in 2.2.1 section, Garmin Lidar Lite provides a measurement of the range of an object in front of it. This enables drone to detect any object within a range of 5cm to 40m. The sensor is connected to the companion computer through a serial interface. The development of software system was done offline and tested with Ardupilot SITL[63] where the behavior of the system was observed before deploying it to real drone. Only after successful testing with the simulator, the obstacle avoidance system was integrated into the real drone and tested with a flight. The collision is avoided by stopping the drone before hitting an obstacle or by steering the drone away from the obstacle.

### 6.1.1 Algorithm

With all the system in place, sensors and flight controller integrated, an algorithm for obstacle avoidance was needed in sensor fusion module. For testing and studying the behavior of the system, the algorithm used is deliberately kept simple. Furthermore, due to the simplicity of Garmin lidar lite and simple nature of data obtained from it, complex algorithms were not needed. Garmin lidar continuously samples the range measurements and has the range data available all the time. The companion computer can read these sampled data from serial port. The value of range reading decreases if the drone is approaching the obstacle.
There are several regions of interest depending on the distance to the obstacle. Different region of interests as shown in the figure 6.1 when a drone is approaching the obstacle is an obstacle not visible, an obstacle in the visible range, braking threshold and region susceptible for collision with GPS drift. A drone can avoid hitting the obstacle by applying brake command when it gets close to the obstacle. Depending on the speed of the drone and reaction time required to stop the drone, the drone may get close to the obstacle before coming to complete halt. The situation can be aggravated by GPS drift which may cause a collision with an obstacle. Thus, the safest action will be to navigate drone away from the obstacle and wait for human input. This information along with the identification of different regions was used to design a control flow for the algorithm.
Control flow is designed with a state machine. The state machine used for controlling the drone is shown in figure 6.2. The state machine has four states. The control is in state S1 when there is no obstacle in the range and continues to stay in this state until an obstacle is visible. Once an obstacle is in the range of the sensor, state of the state machine is changed to S2. The control stays in the state S2 without issuing any command if the obstacle is within the range but more than braking threshold. However, if the drone moved away from the obstacle and no obstacle is visible in this state, the state is changed back to state S1. However, if the drone approaches the obstacle and reaches the braking threshold then the state is changed to S3. In the S3 state, a brake command is issued and drone waits in this state until it comes to complete halt. Once the drone is fully in brake mode, the state is changed to S4 and is safely navigated away from the obstacle to prevent unwanted collision due to GPS drift. Once in the safe position, the state is changed back to S2 and drone wait for control from a human operator.

The state transition is summarized in table 6.1.

| **Starting State** | **Ending State** | **Condition** | **Action** |
|---|---|---|---|
| S1 | S1 | Obstacle is not visible | Do nothing |
| S1 | S2 | Obstacle is in visible range | Change state |
| S2 | S2 | Obstacle is in visible range and obstacle range is more than braking threshold | Do nothing |
| S2 | S1 | Drone move away from obstacle and obstacle is not visible | Change state |
| S2 | S3 | Distance is less than or equal to brake pos | Change state and Issue brake command |
| S3 | S3 | Waiting for brake command to execute | Do nothing |
| S3 | S4 | Drone is halted | Change state and Use guided mode to move away from gps drift susceptible zone |
| S4 | S4 | Drone is in gps drift susceptible zone | Do nothing |
| S4 | S2 | Drone is in safe position | Change state and Change mode to loiter and let pilot take control |

Table 6.1: State Transition Table

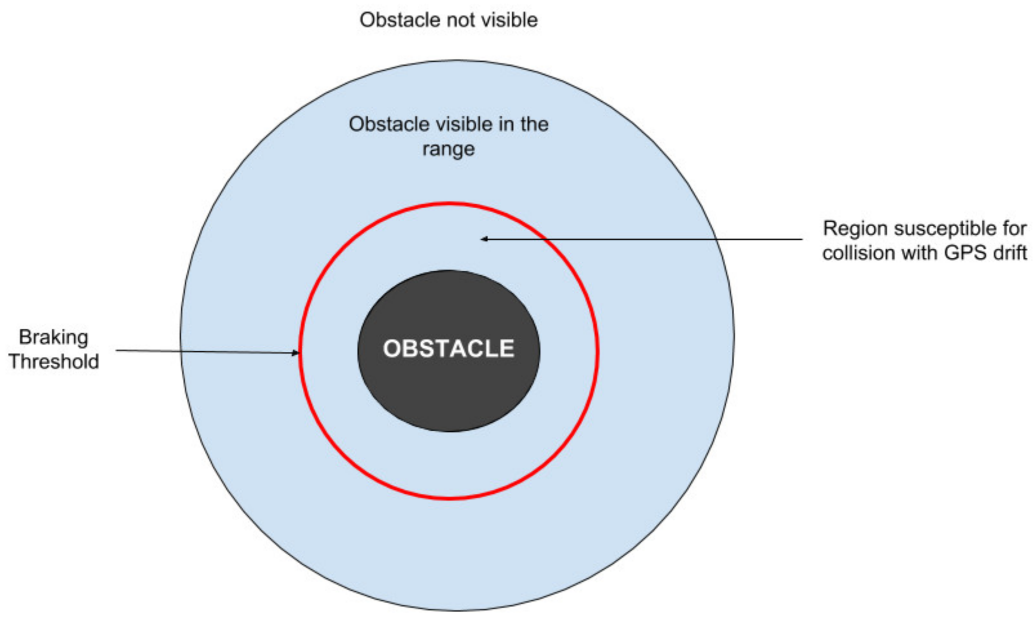

Obstacle not visible
Obstacle visible in the range
Region susceptible for collision with GPS drift
Braking Threshold
OBSTACLE

Figure 6.1: Different regions of interest around the obstacle

### 6.1.2 Simulation Environment

We built a simulation environment for Garmin lidar lite before deploying the implementation in the real drone. For creating such environment, the sensor, and a drone need to be simulated. Instead of implementing a complicated simulation of the sensor, a basic module was developed which mimic the original one only in terms of generation of the data. The generated data was then provided to obstacle avoidance library in the same format as original sensor. The simulated sensor provides distance data varying it continuously such that control flow can traverse through all the nodes and edges of the state machine shown in the figure 6.2. On the other hand, for simulating the drone ardupilot SITL [63], was used which is shown in figure 6.4. It provides the behavior, location and current telemetry of the drone. Despite lacking physics engine and sophisticated solution, it provides basic drone behavior. These behaviors were enough for simulating the obstacle avoidance library and testing the functionalities.

The figure 6.3 summarizes the simulator used for our purpose of testing obstacle avoidance system with Garmin lidar lite. The sensor module connects to Garmin lidar lite simulator instead of a real sensor through interprocess communication whereas mavlink communication module connects

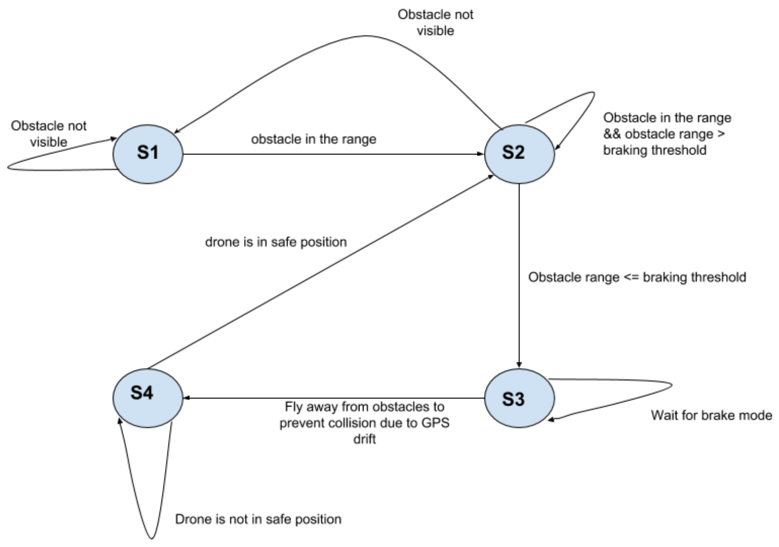

Obstacle not visible
Obstacle not visible
S1
obstacle in the range
S2
Obstacle in the range && obstacle range > braking threshold
drone is in safe position
Obstacle range <= braking threshold
S4
S3
Fly away from obstacles to prevent collision due to GPS drift
Wait for brake mode
Drone is not in safe position

Figure 6.2: State diagram of sensor fusion algorithm

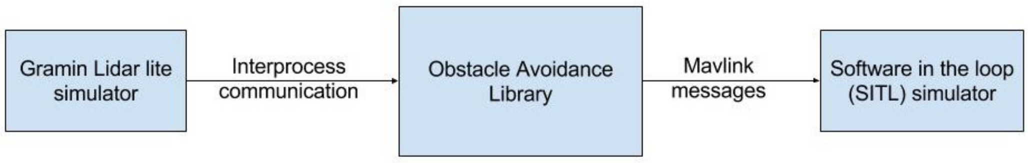
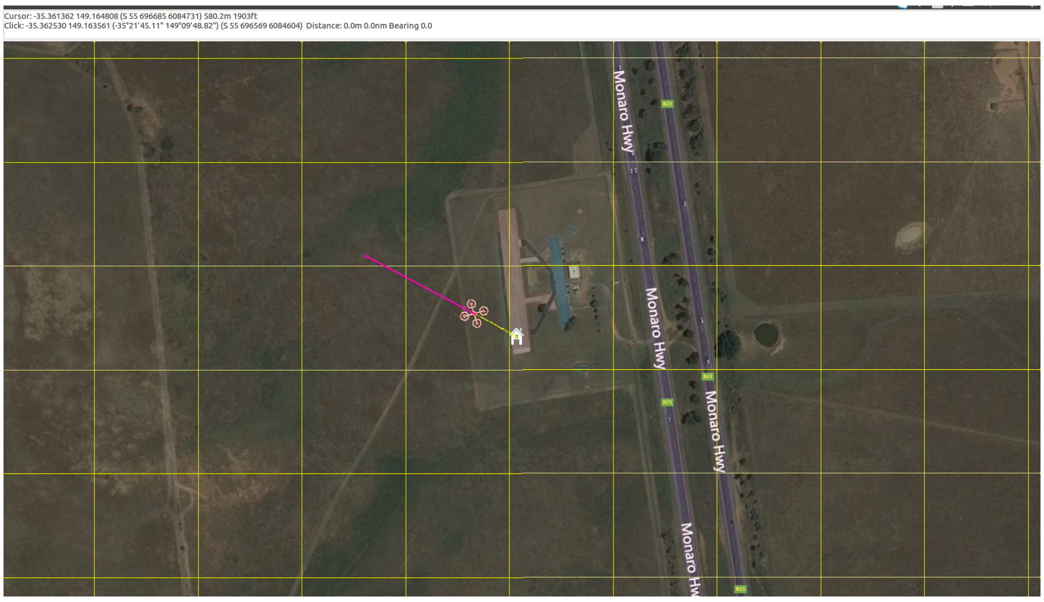

Gramin Lidar lite simulator
Interprocess communication
Obstacle Avoidance Library
Mavlink messages
Software in the loop (SITL) simulator

Figure 6.3: Simulate obstacle avoidance with Garmin Lidar



```
Cursor: -35.361362 149.164808 (S 55 696685 6084731) 580.2m 1903ft
Click: -35.362530 149.163561 (-35°21'45.11" 149°09'48.82") (S 55 696569 6084604)  Distance: 0.0m 0.0nm Bearing 0.0
```

Figure 6.4: Ardupilot sitl with mavproxy for simulating quadrotor drone.

to software in the loop simulator through UDP interface instead of actual flight controller. The distance measurements and reaction to the obstacles were easily observable through the map in the SITL. Some of the behavior verified through simulators were changing the mode, moving away from obstacles, and time taken to execute brake command.

### 6.1.3 Integration with real drone

For the integration with a real drone, the obstacle avoidance library is left intact, but the simulator is replaced with Garmin lidar lite which is connected serially to the companion computer. The mavlink communication module in the obstacle avoidance library now connects to the flight controller through USB interface instead of UDP connection to the SITL. Besides these changes, rest of the system is same as the simulation. The figure 6.5 shows the integration interfaces for different components.

### 6.1.4 Observations and Results

**Simulation helped fix several bugs** The development of simulation environment proved useful. Many of the software bugs were detected and fixed during simulation phase. The functionality of the framework was tested with

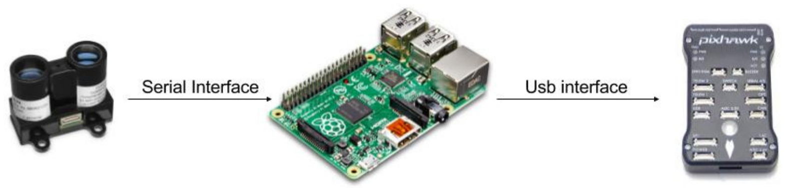


Figure 6.5: Integrate obstacle avoidance with Garmin Lidar

simulated data. The simulation helped verify how sensor fusion algorithm reacted with different distance measurements and the reaction of drones for different commands such as braking and RTL. Once drone behaved as expected and changed its state for different distance measurements as stated in table 6.1 then it was considered ready for deployment. The integration steps itself was straightforward and only needed connecting sensor hardware and drones to proper port and changing few lines in a configuration file.

**Simulation do not represent entire environment** Despite the simple nature of Garmin lidar lite, simulating a sensor behavior was not accurate. GPS readings, attitude pose readings and Garmin lidar readings were free of noises in the simulation environment. However, it was a lot noisy during execution.

**GPS quality can change** In our early test flight, we realized that GPS is not completely reliable and drifts near tall trees and buildings. The figure 6.6 shows the variation of relative altitude measured with GPS when flying in the vicinity of tall buildings. We had few crashes due to this. Thus, to prevent such scenario we improvised our algorithm to account for GPS drift.

**Different sensors have different frequency of sampling** The obstacle avoidance library was getting information regarding the distance from Garmin lidar lite which had sampling frequency configurable from 30 Hz to 500 Hz. However, we were getting location related data such as GPS and attitude from flight controller with a substantially lower frequency in the range of 2-10 Hz. The difference between the sampling of Garmin lidar and readings from ardupilot is summarized in the figure 6.7. This necessitated appropriate sensor fusion algorithm to predict the missed readings.

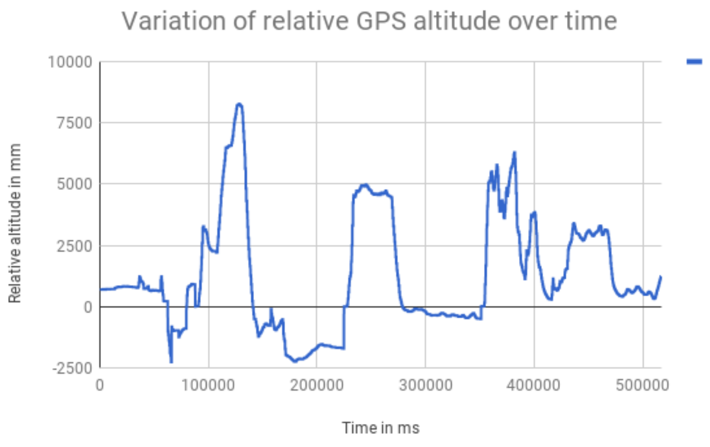


Figure 6.6: Measurement of relative altitude with GPS amidst tall buildings.

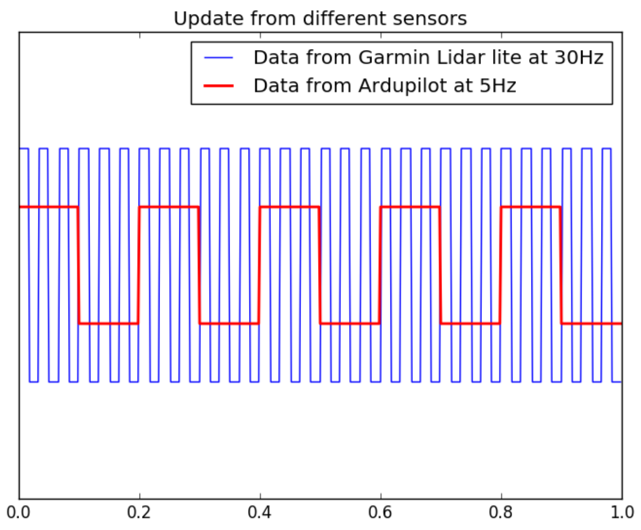


Figure 6.7: Update rate of different sensors.

**Garmin lidar is sensitive to the reflective surfaces** The distance measured by Garmin lidar lite was precise and accurate in a closed environment with white reflective surfaces. However, the performance degraded with other reflective surfaces. Also, due to the low resolution of just 1cm, measuring rough surface was difficult, and it was easy to miss the target. Trials which were done in front of trees and uneven surfaces resulted in several crashes as it was not able to detect them until it was very close. The distance measurements readings collected from same distance but different reflective surfaces are shown in figure 6.8. Nevertheless, the performance of the device was good on the smooth surface. The drone could detect the obstacle and prevent crashing of the drone.

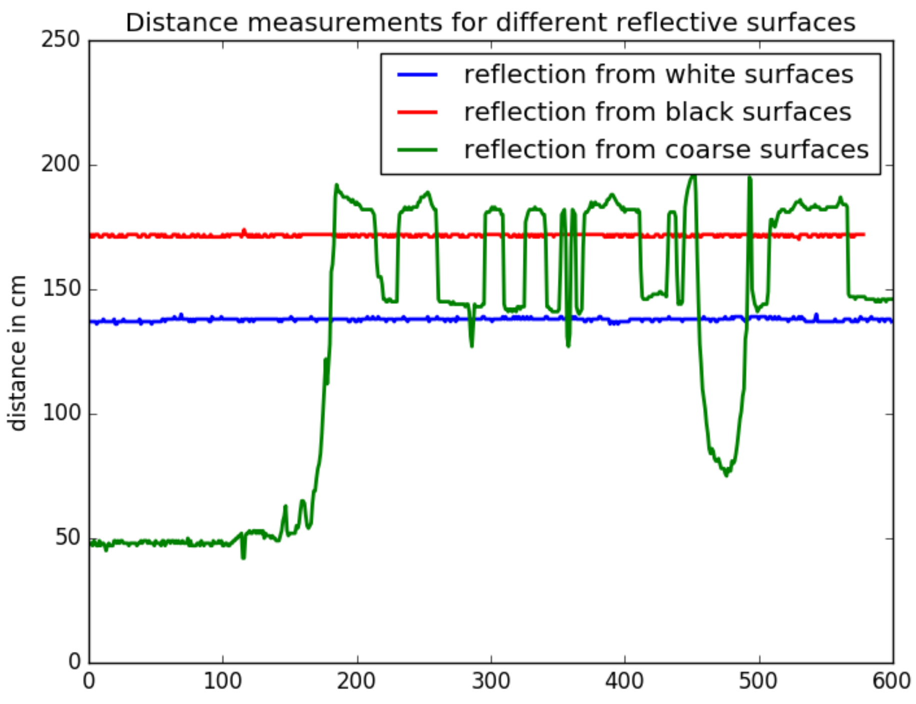


Figure 6.8: Distance measured with garmin lidar from same distance but different reflective surfaces.

## 6.2 Obstacle Avoidance with RealSense Camera

Garmin Lidar lite was a very basic range sensor with a single entry for a narrow field of view. To obtain a large view, the next sensors we tested was Realsense camera r200 which was discussed in chapter 2.2.4. Development for r200 incorporated all the lessons learned from the previous development of Garmin lidar lite. Furthermore, it used the same framework discussed in section 5.2 which was already verified with Garmin lidar. The only modification required to our unified framework was an addition of r200 sensor modules. The sensor module implements the polar histogram algorithm and calculates the polar obstacle density. The calculated polar obstacle density is fed into the sensor fusion module which analyses the data after fusing with input from other sensors and makes a decision about the presence of obstacles. If any obstacles are observed on the path of the drone, then the command is issued to the flight controller for braking, landing or returning to home. In addition to finding an obstacle, large field of vision of Realsense camera assists sensor fusion algorithm to find the escape route as well. The architecture of interfacing Realsense camera to the companion computer on the drone is shown in figure 6.9. Interface for connecting Real sense camera

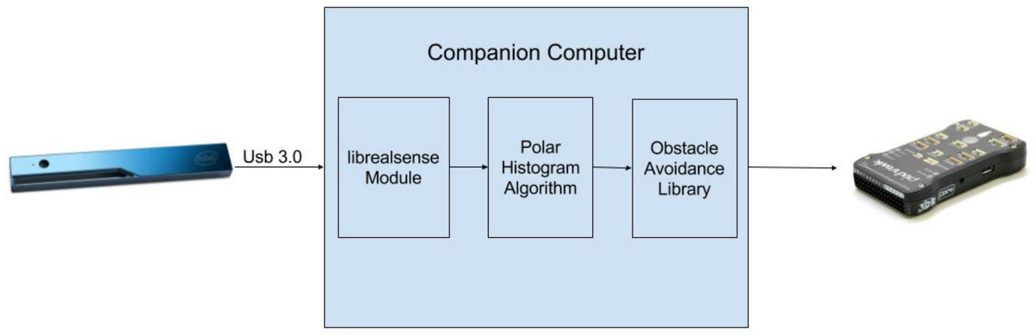


Figure 6.9: Integrate obstacle avoidance with Realsense(r200)

to the companion computer is USB3.0. Depth data is streamed continuously after the connection is established with companion computer. Furthermore, USB connection is also used for controlling and configuring the camera as per the need. The process of obtaining depth streams and changing the settings is done easily by using an SDK provided by Intel and available in [15]. The sensor module of our obstacle avoidance library subscribes to depth streams from the SDK. The subscribed streams are updated to the sensor module which calculates polar obstacle density and provides the calculated information to sensor fusion module. Different modules used for obtaining depth streams and avoiding obstacles are described in detail in the following sections.

### 6.2.1 Librealsense module

The module is an open source SDK provided by Intel and hosted on Github at [15]. It is a cross-platform library with functionality such as native streaming of depth, infrared and color images, synthetic streaming of rectified images, calibration and support for hardware configurations, multi-camera capture and motion tracking. The SDK simplifies the complicated process of accessing Realsense hardware for starting the stream and changing the configurations. For the thesis, the SDK is used for acquiring depth stream and for changing configuration of the hardware.

### 6.2.2 Polar Histogram Algorithm

A separate sensor module is created in obstacle avoidance library discussed in section 5.2 which provides support for the Realsense camera. This module implements sensor interface and is configurable to instantiate from factory method in sensor fusion algorithm. As discussed in section 5.2, sensor module also implements sensor specific filtering or calculations. Polar histogram algorithm defined and used in this section is specific to the Realsense camera and is implemented by the sensor module. The algorithm was discussed in detail in section 2.2.5. The algorithm is used to generate a polar obstacle density in front of the drone. The polar obstacle density is fed to sensor fusion module which makes a decision of the presence or absence of obstacles. Based on the decision, a mavlink message is sent to the drone either to stop it or to take appropriate actions.
The implementation was inspired by the obstacle avoidance library developed for Intel Aero board[14]. The depth image received from the Realsense camera is used as an input for range values required to build polar obstacle density. The first step is selecting the vertical sweep regions on either side of the center of a depth image. The process of selecting window with a height twice the size of vertical sweep strips off some part of the image along vertical axis while keeping constant width. The window is then divided into many vertical segments based on the number of histograms bin needed. The number of histograms bin is selected based on whether fine-grained detail of obstacle is needed or some coarse detail of obstacle is enough. The second step is selecting the lowest pixel value for each segment within the vertical

regions as closest obstacle distance. This ensures that each bin in the segment measures the lowest distance to an obstacle in that region. To get output in terms of angle, each bin is converted to angles. The entire process of dividing the frames and creating vertical segments is summarized in figure 6.10. Similarly, the high level flowchart of the algorithm is shown in figure

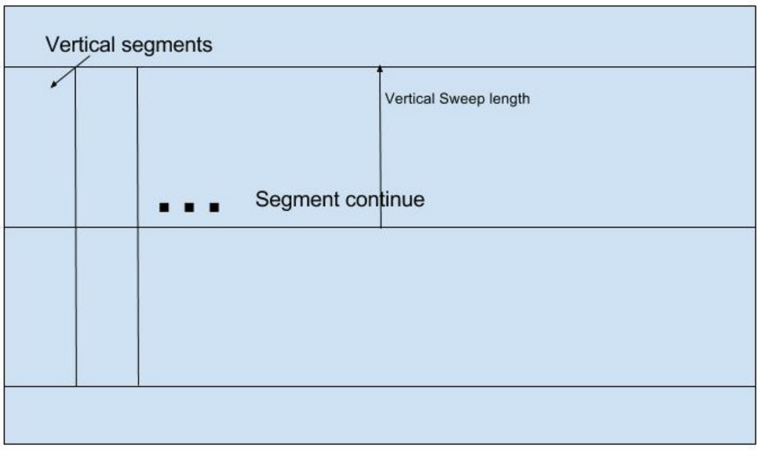


Figure 6.10: Segmentation of depth image for calculating polar histogram

6.11.

### 6.2.3 Simulation

The simulation was done using gazebo [26]. The simulator instantiates vehicle as well as an environment where testing is done. For using the Realsense camera and ardupilot sitl, plugins such as gazebo-realsense [33] and gazebo-sitl [32] are needed. The [14] provides the foundation for building such simulation environment; however, the obstacle avoidance library used for testing was developed by ourselves. The images from the Realsense camera is obtained from the vehicle in the gazebo simulator as shown in figure 6.12. After setting up the environment and starting the simulator, test of the reaction of the drone is done by subscribing depth image data from the gazebo-realsense camera which are processed with obstacle avoidance library. The obstacle avoidance library issued an appropriate command based on the presence of obstacles in the simulator. The behavior of the drone is visible in the gazebo with the help of gazebo-sitl.

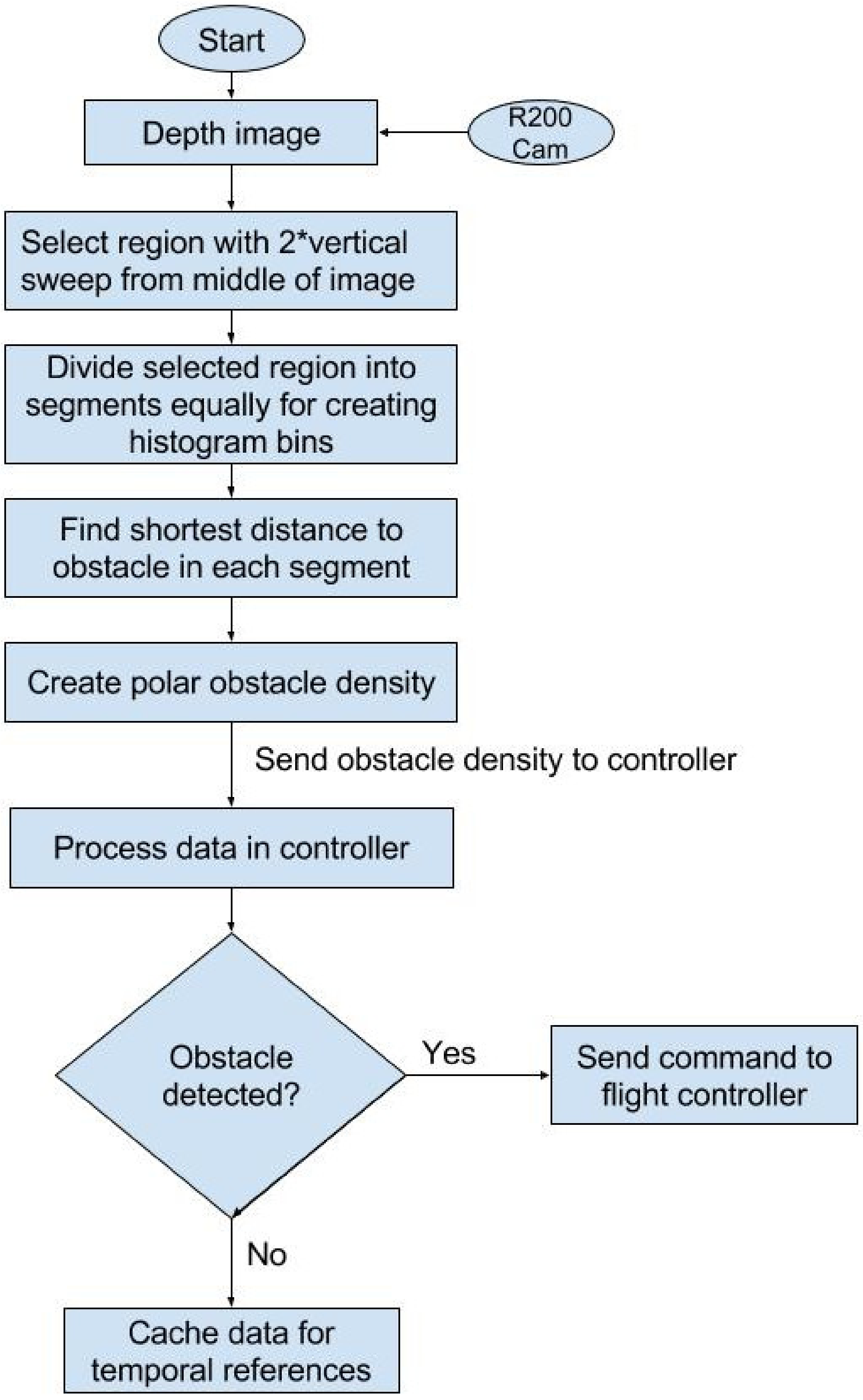

Start
Depth image
R200 Cam
Select region with 2*vertical sweep from middle of image
Divide selected region into segments equally for creating histogram bins
Find shortest distance to obstacle in each segment
Create polar obstacle density
Send obstacle density to controller
Process data in controller
Obstacle detected?
Yes
Send command to flight controller
No
Cache data for temporal references

Figure 6.11: Flow chart for calcualating POD and avoiding obstacles

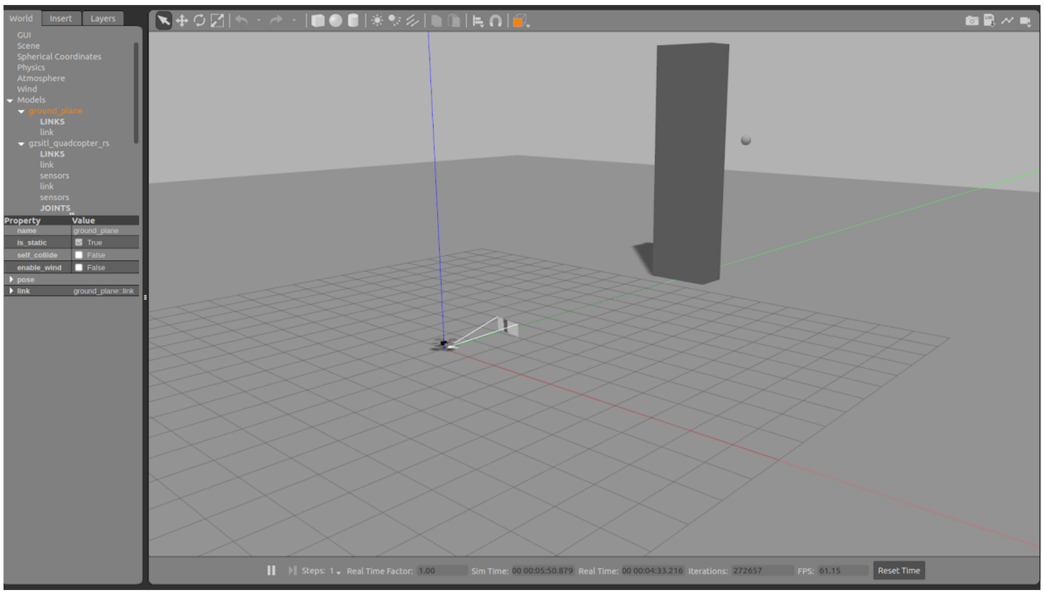
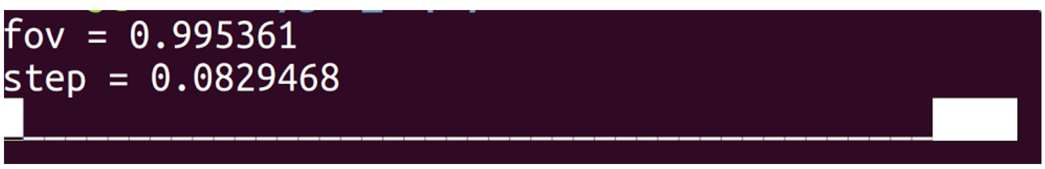


Figure 6.12: Simulate obstacle avoidance with Realsense(R200) in gazebo

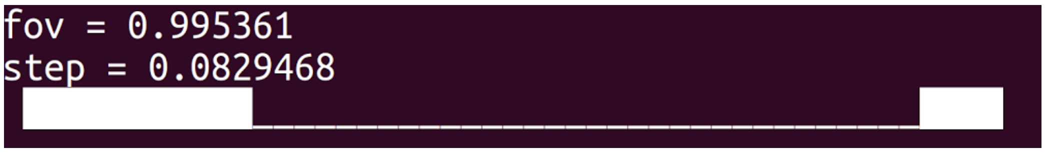
### 6.2.4 Interfacing with real drone

The interfacing is done as shown in figure 6.9 by connecting r200 module to companion computer. Companion computer needs librealsense library [15] for the module to operate correctly. Once the set up is done, companion computer is ready to receive depth images from r200 modules. Obstacle avoidance library in the companion computer receives these depth images through librealsense Application Programming Interfaces (API). Calculation of polar obstacle density in the sensor module starts after receiving depth images. Forty-eight bins or segments of polar obstacles are created for analyzing the presence of obstacles. Figures 6.13, 6.14, 6.15 and 6.16 shows the polar obstacle densities when obstacles were presented in different position in relation to the drone. All these obstacle densities were observed for the indoor position. Figure 6.13 shows the presence of obstacles on either end, but obstacles were present only on the right side, and small obstacles shown on the left is a false positive. Similarly, obstacles present on two corners is shown in figure 6.14, obstacles present on left and middle is shown in the figure 6.15 and figure 6.16 shows obstacles in the middle.



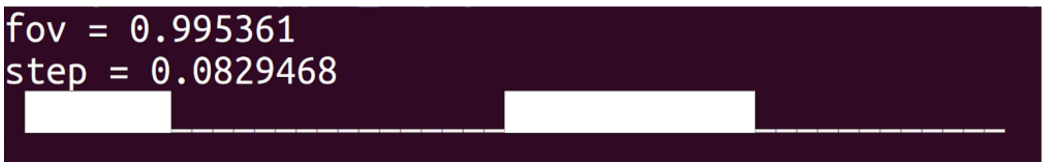
```
fov = 0.995361
step = 0.0829468
```

Figure 6.13: POD for obstacles on the right as bright and error on the left



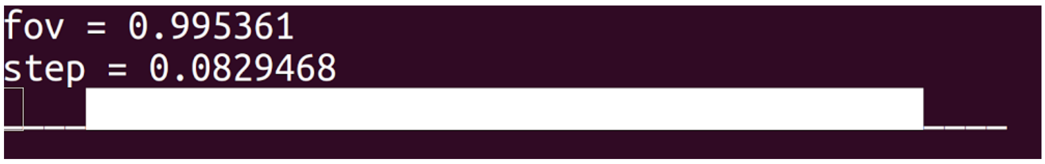
```
fov = 0.995361
step = 0.0829468
```

Figure 6.14: POD for obstacles on the right and left



```
fov = 0.995361
step = 0.0829468
```

Figure 6.15: POD for obstacles on the left and middle



```
fov = 0.995361
step = 0.0829468
```

Figure 6.16: POD for obstacles only in the middle

### 6.2.5 Observations and Results

**Gazebo has better simulation environment** Compared to the simulation environment shown in figure 6.4, the simulation environment in the gazebo shown in figure 6.12 is more robust and realistic. Despite using the same simulation backend environment of ardupilot sitl [63], it provides an additional benefit of rendering the behavior in three dimensions. Additionally, it models the sensors more realistically which provides proper data rendered from gazebo environment and is close to real life scenario. Using such simulator to test the algorithm to verify the functionalities is useful. It helped in debugging the algorithm and making modifications until desired reactions were observed in the simulator. Eventually, the algorithm performed well in the gazebo simulator, and the drone was successfully able to avoid the obstacles by both stopping and going around it.

**Noise in the real world is a lot more than gazebo simulation** Depth images in the real world, however, were different from those in the simulator. The depth data observed were noisier, and polar histogram often showed false positive readings for obstacles. Testing for obstacles was done for both indoor and outdoor. Comparatively the performance of the camera was much better inside the room than outside. The depth images from outdoor were noisier which made it difficult for identifying obstacles. The figure 6.17 shows an image from a real device.

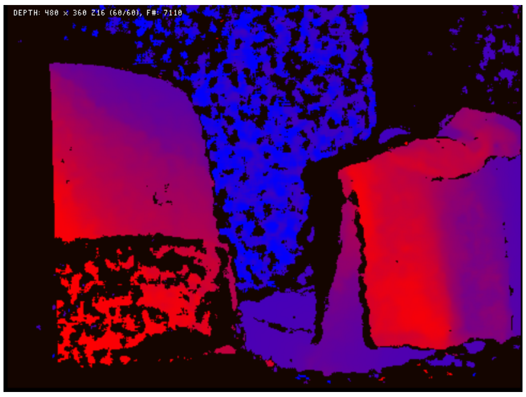


Figure 6.17: Noisy data from Realsense camera r200 module.

**Larger field of view at the expense of lower obstacle range** Field of view is increased with realsense camera [15] unlike Garmin lidar lite. However, it does not provide the obstacle range provided by the Garmin Lidar. The obstacle range reduces to 3-5m indoor and 8-10m outdoor.

**Unified architecture helped in faster development** The unified architecture helped in the faster development of avoidance system for the realsense camera as it just required small enhancement in the architecture. R200 sensor modules were added to the obstacle avoidance library while reusing most of the sensor fusion algorithms and all of the communication module.

**Comparison to the open source avoidance library [14]** Opensource implementation of collision avoidance library is available in [14] which is easy to build and deploy. However, this opensource library has the limitation of supporting Realsense camera only, has fixed sets of algorithms and is specific to Intel aero board. However, our implementation is flexible to use in different architecture.

## 6.3 Camera as an obstacle avoidance sensor

The architecture of autonomous cars has proactive and reactive avoidance system which was discussed in section 3.3. The proactive system incorporates avoidance during the path planning phase whereas reactive system acts as the last line of defense in preventing a crash. The implementation of basic algorithms with Garmin lidar and polar histogram with Realsense are reactive obstacle avoidance system. For boosting the system with proactive capabilities, obstacle avoidance library needs a better understanding of its surroundings such that avoidance of obstacles is done during path planning. However, clear attributes of obstacles such as size, distance from the drone and relative position are needed. The problem of better understanding obstacles can be mitigated to some degree by using camera empowered with AI in an existing system. The AI algorithm can be a simple search problem as discussed in section 2.3.1 to CNN discussed in 2.3.4 section to reinforcement learning problem as discussed in 2.3.7 and 2.3.8 sections.
Image segmentation is used in this thesis for better understanding nature of an obstacle. The segmented image gives the location of various objects in the image. This information despite not having information about depth values of obstacles can be useful for augmenting with range sensors using sensor fusion. The prior work has been done for obstacle avoidance using image segmentation in [11]. For verifying the concept and testing the framework, image segmentation is done with the open source aerial drone datasets from Nvidia [51]. The segmentation module developed can find the regions of

ground terrain and sky. This differentiation is used for obstacle avoidance by redirecting the drone towards the sky. The flow chart for image segmentation pipeline is shown in figure 6.19. The obstacle avoidance library used is same as that of Garmin lidar and Realsense. The output from AI component is provided to sensor fusion module. The basic architecture of the segmentation module for obstacle avoidance is shown in figure 6.19. The figure shows that segmentation module implements sensor interfaces and updates its result to sensor fusion module.

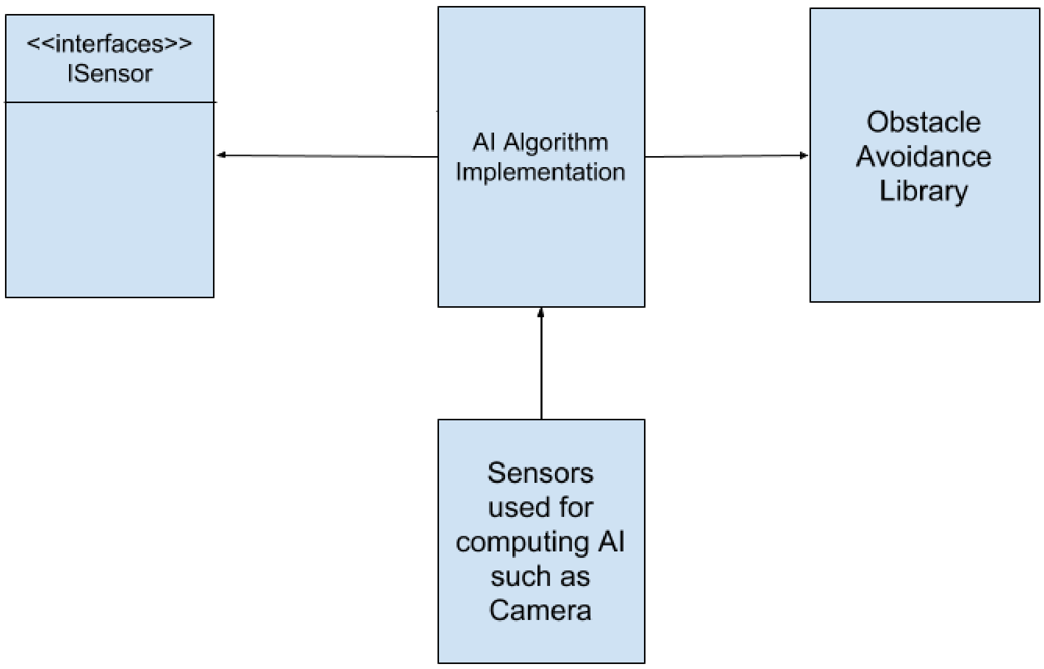

<<interfaces>>
ISensor
AI Algorithm Implementation
Obstacle Avoidance Library
Sensors used for computing AI such as Camera

Figure 6.18: Obstacle Avoidance Architecture with AI

### 6.3.1 Fully Convolutional Neural Network for image segmentation (FCNN)

Convolutional Neural Network is discussed thoroughly in section 2.3.4 and FCNN is an extension to the network discussed there. The final layer of CNN is a fully connected layer which does the task of classifying using SVM or softmax. The input to the fully connected layer are features extracted from convolution layers. However, fully connected layers can be replaced with equivalent convolutional layer by keeping the size of the filter in CNN same as the size of the input to the fully connected layer. As it can be

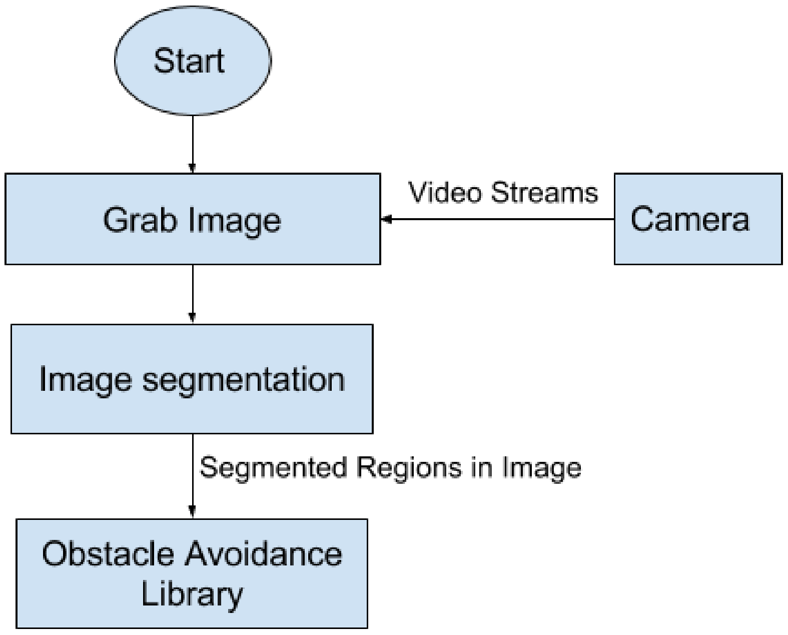


Figure 6.19: Flow Chart of image segmentation

seen from the figure 6.20, 4096 features are extracted from the image, these features can be fed into the fully connected layers for classification or else replaced with a convolutional layer with 4096 filters of dimension 1x1. The segmentation happens pixelwise by generating one probability distribution per pixel in the image. As the input image moves down the convolutional network, input pixels are compressed into coarse and higher-level feature representations. The process of extracting these features is done through deconvolutional layers[46].

### 6.3.2 Training of Aerial Drone Dataset

The datasets are open source and are available for downloading from [16]. The dataset is used to train fully convolutional neural network. The network used for training is a variant of Alexnet. The number of epochs selected was 30 with snapshots saved for each epoch. Learning rate is decreased with increasing epochs to fine tune the network as shown in figure 6.21. Training and validation errors were also observed and are plotted in figure 6.22. All these figures are generated with digits framework[17].

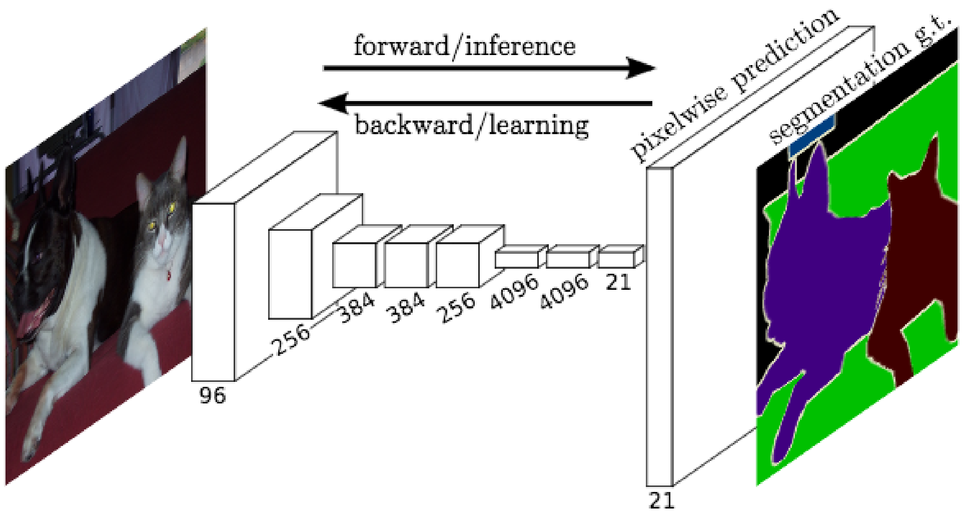


Figure 6.20: Fully convolutional neural network for image segmentation [46]

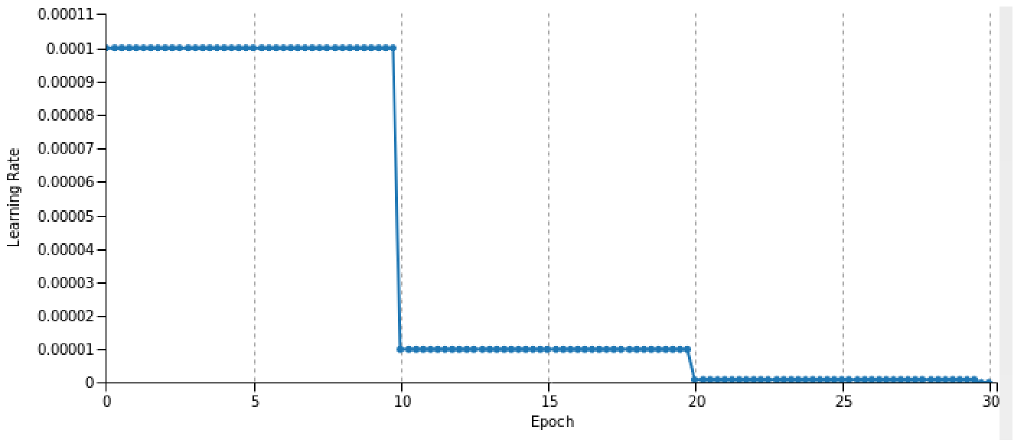


Figure 6.21: Learning rate in different epochs generated from DIGITS

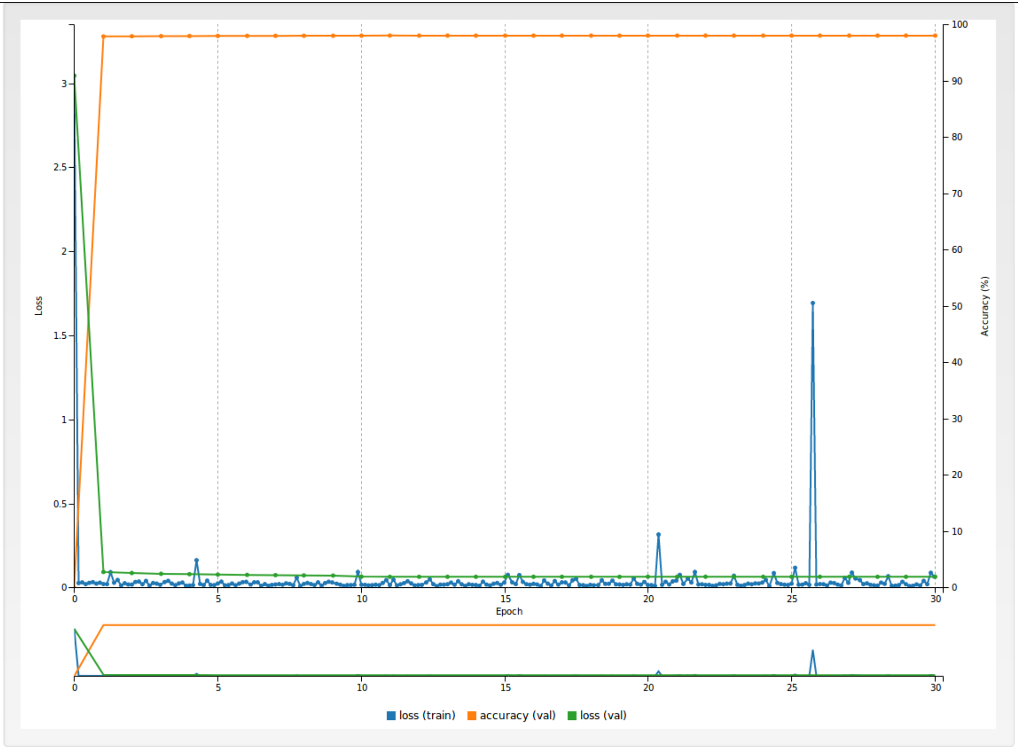


Figure 6.22: Training and validation error in different epochs generated from DIGITS

### 6.3.3 Deploying to Jetson

Once the model was trained with a dataset, the snapshot was downloaded which is .caffemodel, along with deploy prototxt, labels, and colors for them. The deployment can be done using tensorrt model for deploying in Nvidia GPUs. This module is used for testing with collected few test images. One of those test cases is shown in figure 6.23. Figure 6.23 is the input image and output after segmentation is figure 6.24. It can be seen that the sky is segmented from the ground. However, implementing it for a real-life scenario with near real-time computation needs much fine-tuning. Image segmentation with Nvidia Jetson is a proof of concept for using AI for obstacle avoidance on the drones.

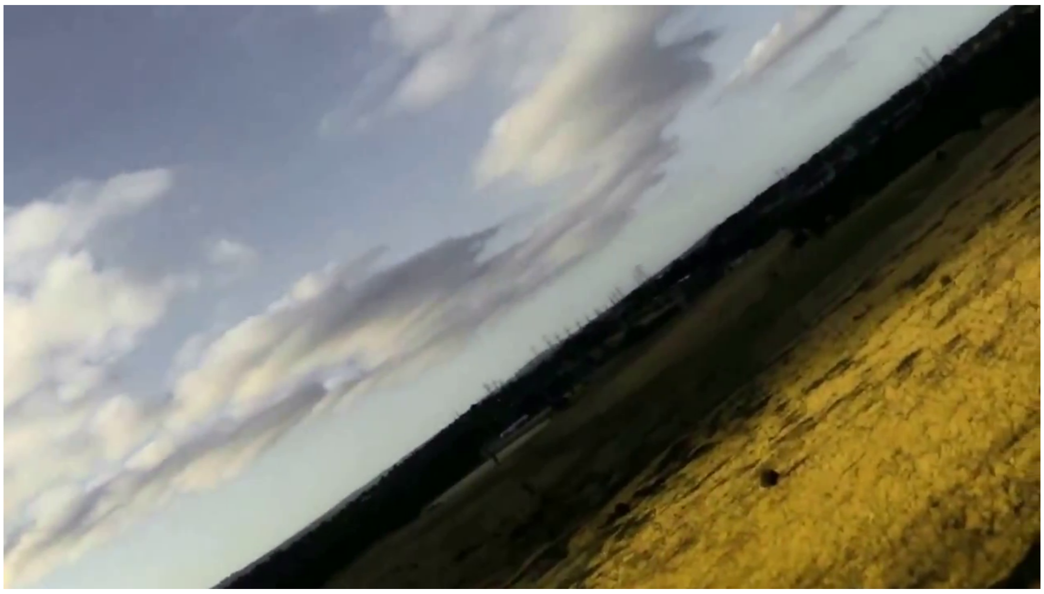


Figure 6.23: Test input data from [16]

### 6.3.4 Observations and Results

**Incorporating system with image segmentation proved to be tricky** Despite providing pixel level segmentation of the images, it proved difficult to analyze and use the information directly to the obstacle avoidance unit.

**Comparison with Garmin Lidar and Realsense r200 module** Garmin Lidar has a point of 1 cm resolution to detect an obstacle. Similarly, polar

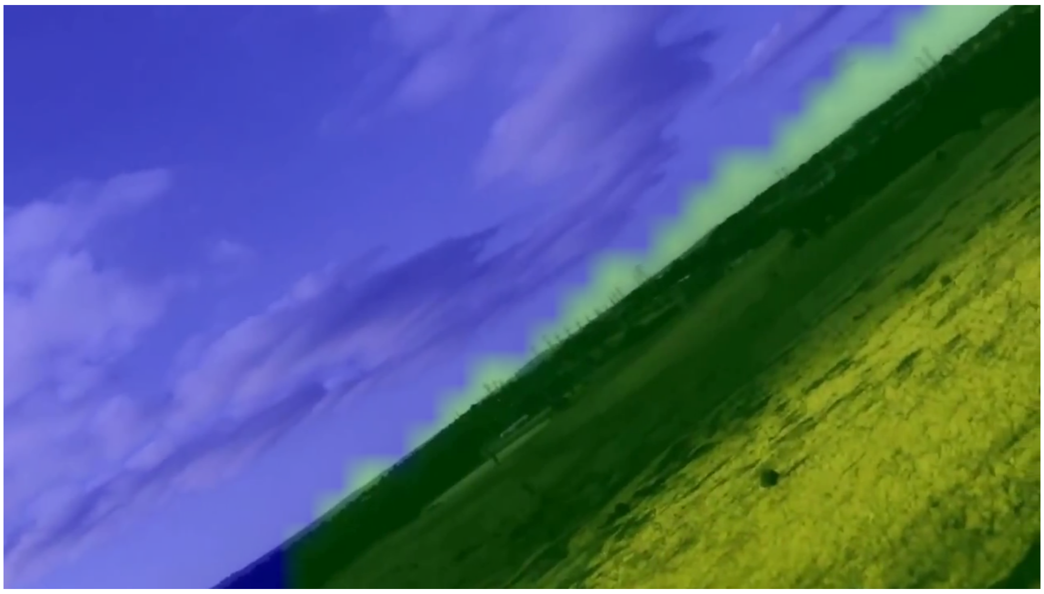


Figure 6.24: Output from the model for test input data

histogram with r200 module can provide the depth information of obstacles in front of the drone in a single dimension. However, with segmentation, we can identify the obstacles up to the pixel level along two dimensions. The nature of the obstacles can be easily found out with this information and the intrinsic parameters of the camera.

**Using already available frameworks such as DIGITS helped** Most of the cleaning and managing the datasets are taken care by the framework and easy interface served in the faster development of the module.

**Possibility for extension** The experiment is a proof of concept in using AI for obstacle avoidance, and it can be easily extended to use with custom models and datasets. Instead of using image segmentation, any other AI techniques can be used. For much robust implementation and outcome from using AI module, training data can be collected and trained as per the need.

**Limited to only testing offline test images** Testing was limited to offline with still images due to time constraints. Obstacle avoidance library does include functionality for obtaining live video stream from the camera and the functionalities currently tested can be extended to an end to end control of the drone.

## 6.4 Navigation using AI

Navigation ensures drone can travel from one waypoint to another. The navigation task requires drone to first localize itself to known map and estimate its position and velocity. Then it must apply thrust to fly towards its destination. The obstacle avoidance library developed above can be extended for navigation as well. Additional sensors which can assist in navigation can be added to sensor modules. An example of such sensors is visual odometry or optical flow sensor which was discussed in chapter 2.1.3. Sensor fusion algorithm can extend or override the algorithm provided by obstacle avoidance library for supporting navigation. The extended algorithm needs to plan the navigation path in addition to keeping track of the obstacles. Mavlink communication module must implement additional mavlink messages for sending a command to flight controller and navigate based on the relative position or relative attitude. Some of such commands which are used for navigation are SET_POSITION_TARGET_LOCAL_NED and SET_ATTITUDE_TARGET. The use of optical flow can also help in mitigating problem of flying in an area with bad GPS signals such as forests, cities with tall buildings and tunnels. Artificial intelligence can also provide assistance for auto navigation of the drones in such places.

### 6.4.1 Experiment

The usage of AI for obstacle avoidance is already discussed in chapter 6.3. This section explains experimenting with a similar approach for auto navigation of drones. The dataset is used from [30]. We have already discussed similar approach in self-driving cars in section 3.6. The similar approach for the drone helps in verifying the possibility of auto navigation with an end to end neural network. The architecture of the network is resnet with 87 layers. The output of the architecture is six softmax unit. Three out of six outputs are for generating left, middle and right facing of the camera, whereas other three outputs are for generating left, middle and right offsets of the drone from the trail. In such approach, vision is used as the main sensor for localizing. Optical flow is used for estimating the position and velocity of the drone enabling drone to fly in the absence of GPS as well. Experimentation is done using redtail project from Nvidia [57]. The pre-trained model used is also from the same project [57] for experimenting with auto navigation. Due to the limitation of time, it was not possible to implement the architecture in the real drone, but the experimentation and behavior of the drone are observed in the Gazebo simulator. The behavior was verified for the drone by continuously passing an image and checking out the behavior of the drone in the gazebo.

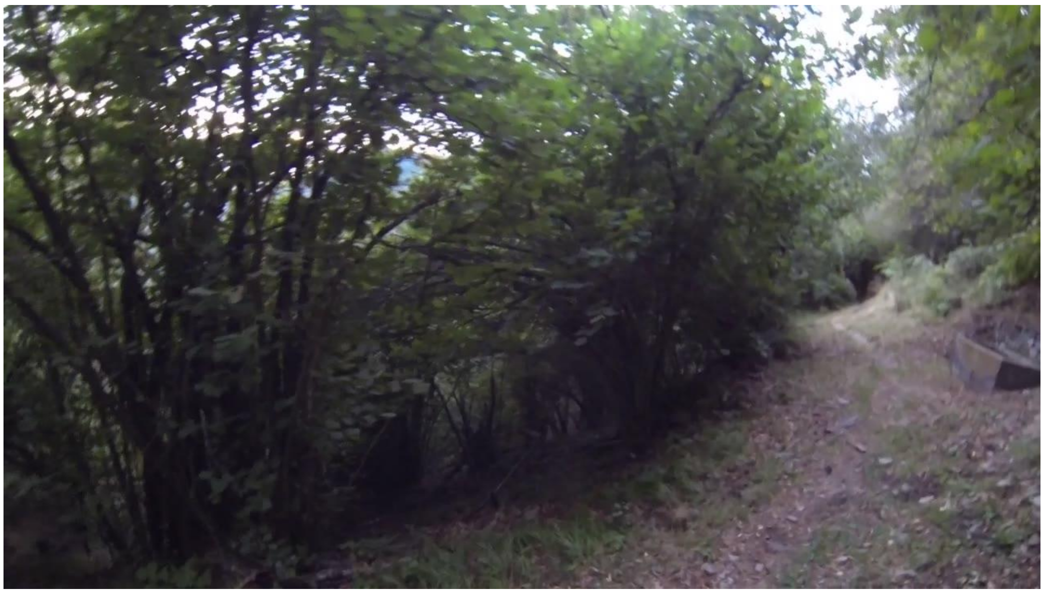


Figure 6.25: Image facing Left [30]

**Image facing left** The input to the deep neural network for trail following was given with figure 6.25. That produced output from a deep neural network as [0.9458076357841492, 0.05343988910317421, 0.0007525121327489614, 0.7766426205635071, 0.08141788095235825, 0.1419394463300705]. First three parameters from the output are angular offset denoting angular deviation from the center whereas last three parameters are position offset denoting position deviation from the center of the trail. It represents left facing, middle facing, right facing, left position offset, the middle position offset and right position offsets. The output from the image shows that 94.58% probability that it is facing left, 5.3% facing middle and negligible chance that it is facing right. Similarly, it is in 77.55% left offset and 8.1% middle and 14.19% right offsets. When this input was fed into the controller, it gave a turn angle of -15.80. The corresponding response in the gazebo is shown in 6.26.

**Image facing Right** The output of the network for input figure 6.27 is [ 0.003537290496751666, 0.9662114381790161, 0.030251335352659225, 0.30399036407470703, 0.02726331539452076, 0.668746292591095]. The DNN network thinks it is facing middle and has offset towards the right. As a

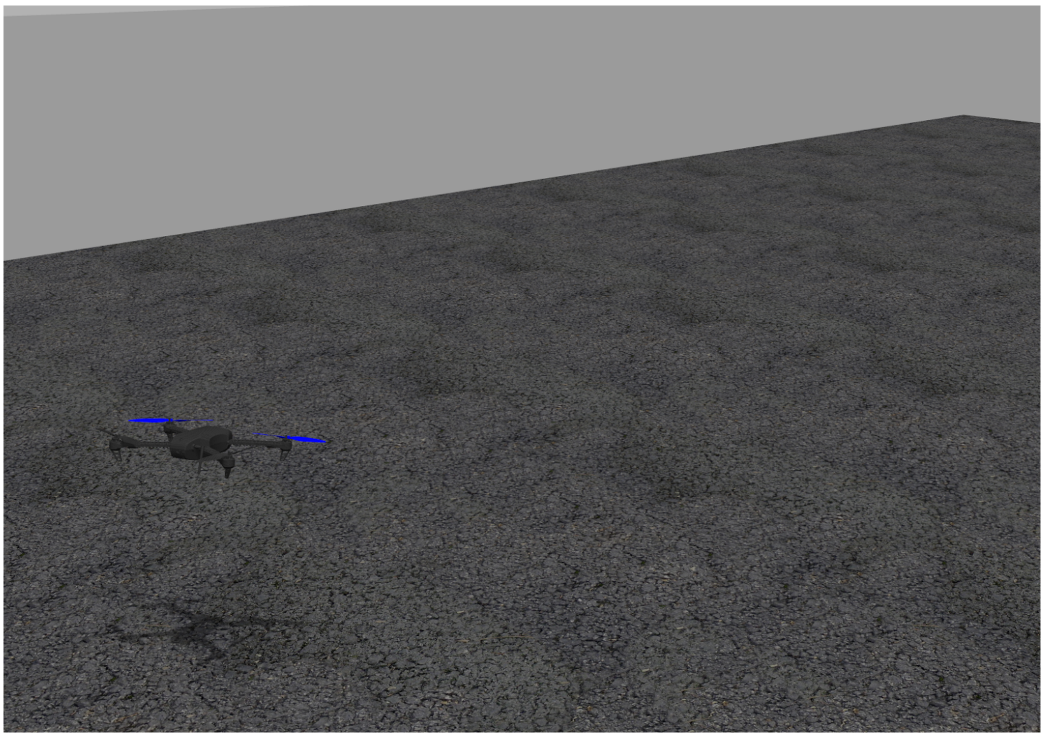


Figure 6.26: Output of gazebo for image facing left

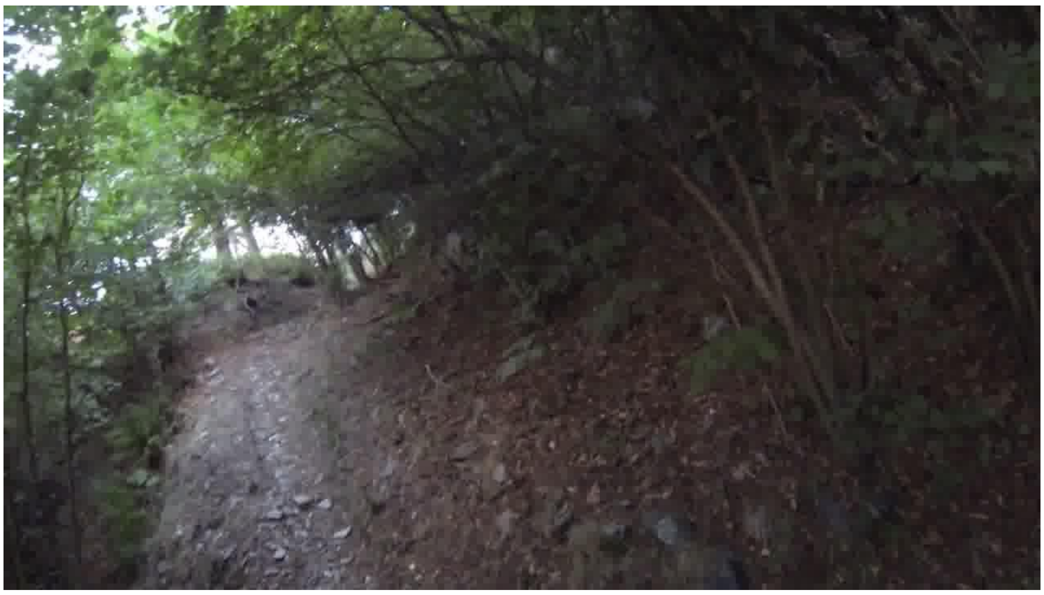


Figure 6.27: Image facing right [30]

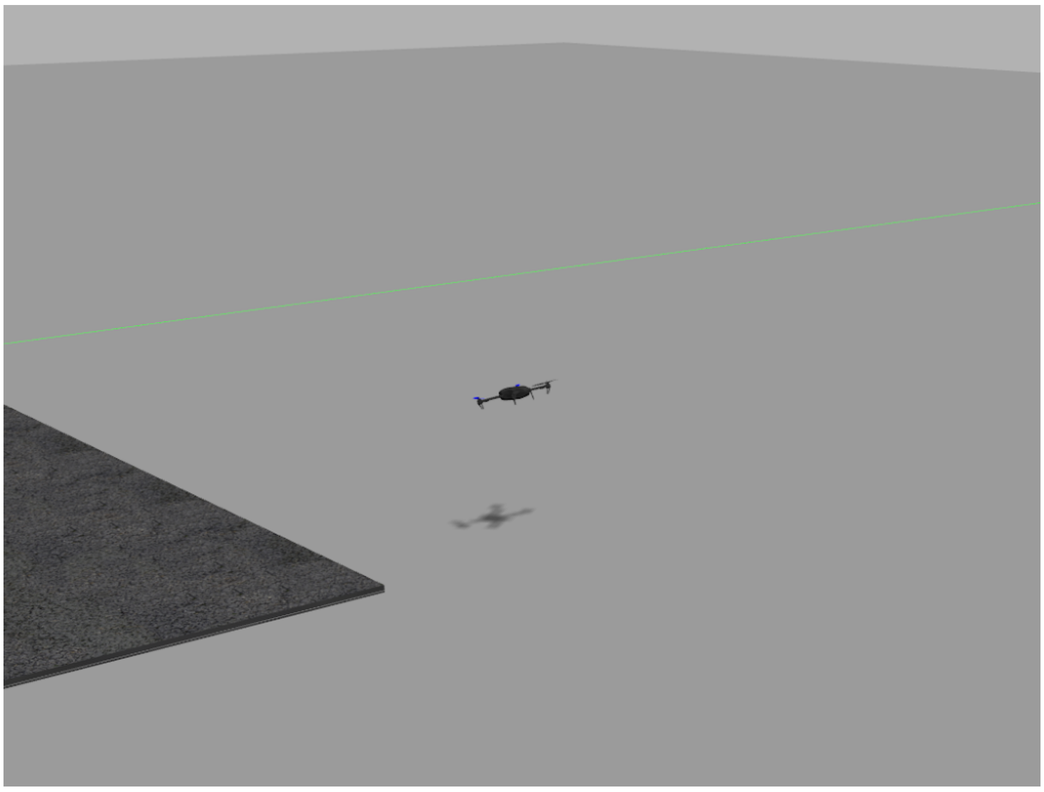


Figure 6.28: Image facing right response in Gazebo

result, turn angle of 3.91-degree is issued for the controller. The images in 6.28 which is a simulation in gazebo shows drone trying to turn left to compensate the right offset but the angle of rotation is low.

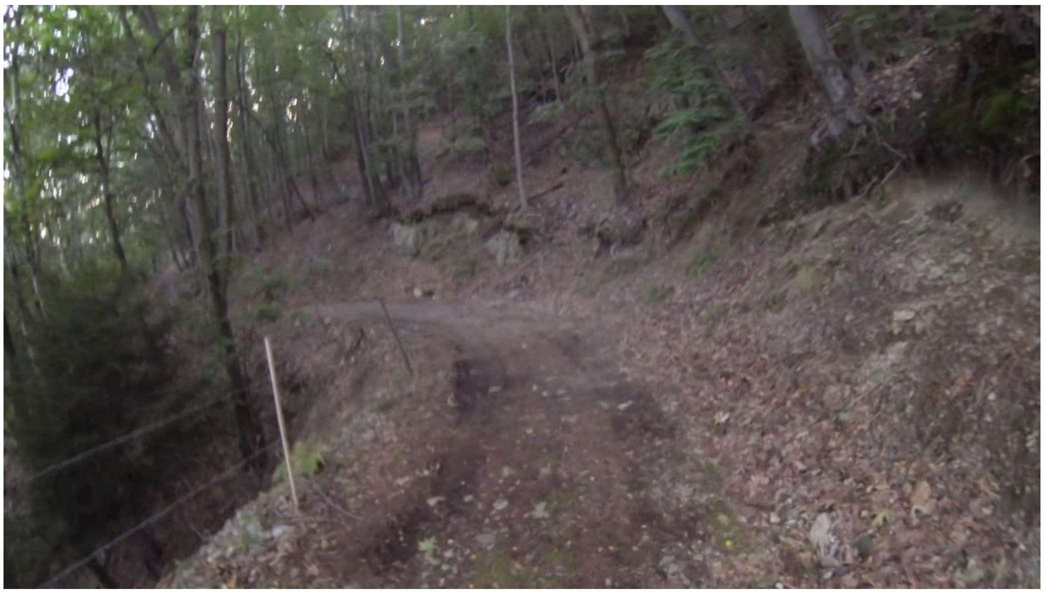


Figure 6.29: Image facing middle [30]

**Image facing middle** The output of DNN when the input is figure 6.29 is [0.008007721975445747, 0.9594460129737854, 0.032546259462833405, 0.4634335935115814, 0.011919651180505753, 0.5246467590332031] The probability of the drone facing forwards is 95.9% and it has equal left and right offset so it continues to move in the same path. Only 0.86-degree rotation angle is calculated for the drone to rotate. The corresponding response in gazebo is shown in 6.30.

### 6.4.2 Observations and Results

**Model is used and tested as a proof of concept based on [57]** The module was used to test the feasibility of end to end training using deep neural network based on [57]. Due to the limitation of time, it was not able to be tested on real hardware but was tested only in the simulated gazebo environment.

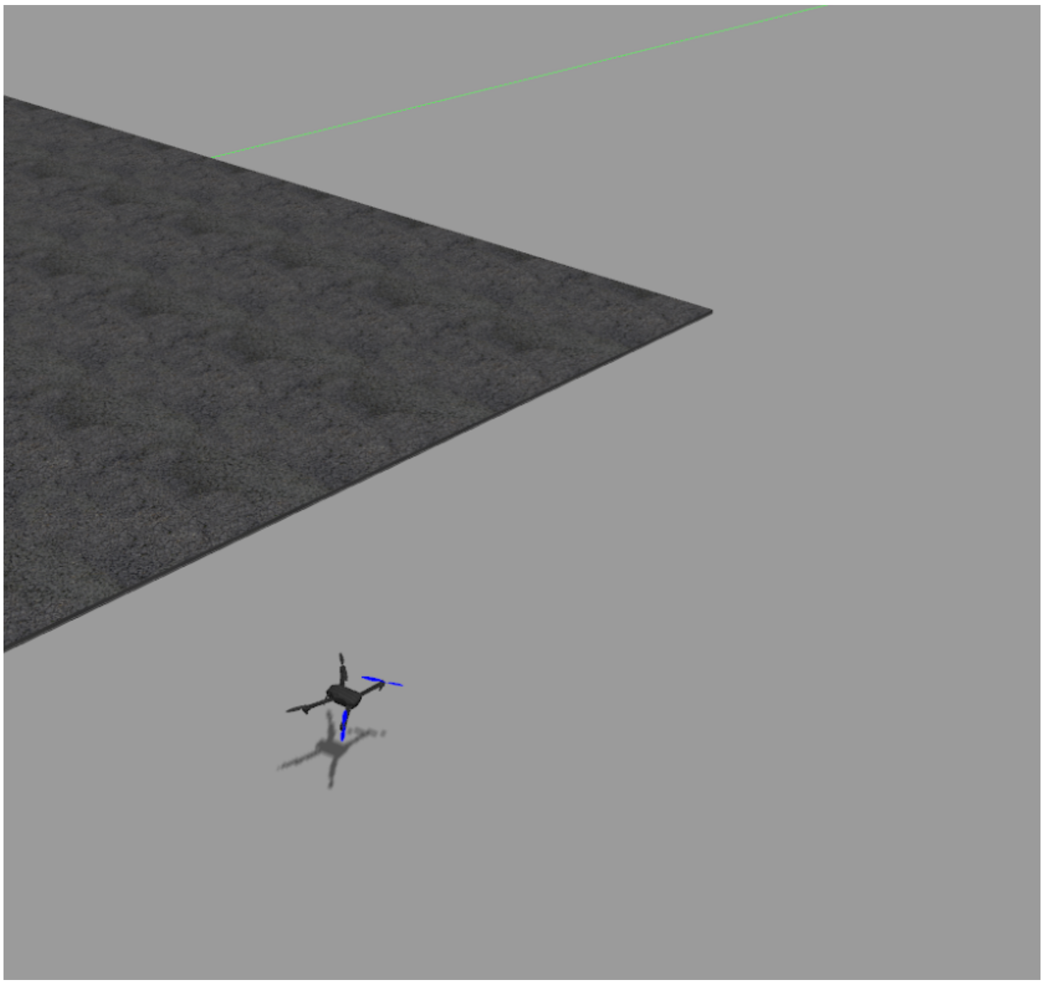


Figure 6.30: Image facing middle response in Gazebo

| **Criteria** | **Garmin lidar lite** | **Realsense (R200)** |
|---|---|---|
| Measurement distance | 25-30m | 3-5m indoor, 8-10 m outdoor |
| Weight | 22 gram | 65 gram |
| Performance outdoor | Same as indoor | Very bad |
| Noise level | Low | Very noisy |
| Light condition | No affect | Gets affected |

Table 6.2: Comparison between Garmin lidar and R200

**Navigation of the drone is possible with an end to end training of deep neural network** Observing the reaction of the drone solely on the gazebo, it was seen that the drone was reacting to the input images and was trying to follow the forest trail.

## 6.5 Summary

Garmin Lidar and Realsense camera were integrated into the obstacle avoidance library. The comparison between two is presented in 6.2. The test was done in the simulator as well as the real drone. The functionalities of the design discussed in chapter 5 was verified. The same architecture was used for developing different obstacle sensors as well as image segmentation and navigation. Most of the components in the architecture were reused and can be easily upgraded with the requirements of small modifications. Furthermore, the experimentation with image segmentation and navigation using AI proved that artificial intelligence can boost autonomy in the drone as it is doing in autonomous cars. Next chapter will now discuss the major lesson learned and outcome of this thesis.

# Chapter 7

# Discussion

The thesis has explored different aspects of drone and autonomy. Mainly thesis has revolved around the problem of navigation and obstacle avoidance in the drone. The possibility for the solutions of the problem using AI was also explored. This chapter presents the outcome of the thesis, lesson learned, achievements and mistakes made.

## 7.1 Software architecture

**Simple and extendable architecture** The architecture developed was simple and extendable. The development of extendable architecture proved to be useful. This allowed fast experimentation with different sensors and algorithms. There is no need to implement an entire system for each trial and development. This reusability allows reusing the tested framework which prevents unwanted bugs. If the framework is well tested with one sensor then it will be robust to use again for other sensors as well. The simplicity of the architecture allowed the possibility to easily port it to any system.

**Parameters configurable** There are several parameters required by the algorithms. Changing the parameters in the code itself proved to be expensive as it required building the binaries and porting the code to the drone. It required larger development cycle especially for fine tuning the parameters. However, the architecture supported configurable parameter for testing. The parameters can be easily modified in the configuration file without modifying the code. This helped in reducing the development time.

**Easy to simulate** The architecture was so designed that it was easy to interface with a simulator. The simulation proved to be useful in finding several typos and bugs in the software system. This was especially helpful as testing with simulator only took few seconds compared to hours required to prepare and test with real hardware.

**Simulation is helpful but not a replacement for real test** Simulation proved to be useful but simulated environment was far from the real environment. Testing in the real environment with real hardware was required to measure the performance of the sensor or algorithm in the real-life scenario.

**Supports ardupilot flight controller for advanced obstacle avoidance** Ardupilot has limited support for obstacle avoidance which is mitigated by the development of this library. The library supports for avoidance in a different mode of operations, supports various sensors and is capable of using different algorithms.

## 7.2 Obstacle Avoidance

**Weight Limitation** Unlike autonomous cars or robots which does not have much implication due to additional weights, it is difficult for the drones to carry heavy and big sensors. Thus, a drone is primarily limited to sleek and low weight sensors for obstacle avoidance.

**Range and field of view** When selecting a sensor for obstacle avoidance, the important metrics to consider are range and field of view. The range is of vital importance and the range required by the sensor is directly proportional to the speed with which drone is traveling. For avoiding obstacles for the drones traveling at high speed, it is required to sense obstacles from far away compared to the one traveling at slow speed. Similarly, the field of view is needed if we want to plan an escape route. If it is enough to just stop the drone in front of the obstacle or detect the obstacle in the path of the drone, then a narrow field of view is enough. However, for planning an escape route, the field of view should be large.

**Reaction should be in realtime** The reaction of obstacle avoidance system should be real-time to ensure prevention of the crash. However, ensuring real-time performance was not an easy task. First of all, the operating system we were running our software on was Ubuntu which is not a real-time OS. Furthermore, it was difficult to ensure reactiveness of each module especially the telemetry data obtained from the flight controller.

## 7.3 Artificial Intelligence

**Supports proactive avoidance system** The obstacle avoidance system developed are usually reactive where an action is taken when an obstacle is detected. This ensures the current path planning is canceled and the decision is made for rerouting. However, with the support from AI, dynamic path planning can be done by continuously taking input regarding the obstacles in the surroundings and can be maneuvered smoothly.

**End to end training for navigation** Classic computer vision problem use feature definition, feature extraction, and control training. Instead of that, the end to end model focuses on providing camera images as an input and pose estimation as output so that all the processes are included in the deep neural network.

**Collection of training data is not easy** Collecting entire training set for machine learning is not an easy task. Especially with deep learning and increased depth of neural network where it is easy for the model to overfit. This is preventable with a large volume of data. However, collecting the data, processing it and augmenting data itself required a huge amount of time. Thus, we decided to use open datasets for developing proof of concept experiments.

**Development cycle is slow** Depending on the size of the training data, it took few hours to a couple of days for training the model. This required few days to test a model and if changes in the model are required it would take another couple of days. Thus, a shallower model was tested for functionalities before trying deeper network.

**Performance should be measured in deployment environment** The development environment is usually equipped with much powerful hardware compared to the deployment environment. In our case, training was done in GTX 1080 graphics card and deployment hardware was jetson which is less powerful. This caused a reduction in throughput performance. For segmentation with real images, it dropped from 15-20 fps for the development environment to 2-3 fps for deployment hardware.

## 7.4 Thesis planning and implementation

**Time Management and staying focused** Due to the vastness of the topic, it was overwhelming time and again which resulted in deviating away from the core topic of research. Thus, during the start of the thesis, the search was too broad and diverged touching different components and technologies. After understanding that straying into too many topics would not result in sound output, the search was concentrated in particular issues such as sensors and their mode of operations, obstacle avoidance algorithms and fundamentals of artificial intelligence.

**Scope of the thesis** The scope of the thesis proved to be too large and unmanageable at different period. During the start of the project, the scope and knowledge level required for implementing was not considered thoroughly. It required a huge amount of time for understanding the fundamentals of quadrotor and ardupilot system alone. Implementing and testing in real drone was not easy either. There were several components required and procuring them was not easy. The decision to use only simulation for image segmentation and end to end navigation and to use open datasets were made to keep the scope manageable for the thesis.

**Decision to research on autonomous cars proved fruitful** Autonomous cars had already solved many of the problems which this thesis was trying to solve for drones. It warranted a thorough study of autonomous vehicles undertaken for getting inspiration from previously available technology. Understanding of autonomous cars was fruitful later while designing software architecture for navigation and obstacle avoidance for the drone.

## 7.5 Future Research

Both Artificial Intelligence and drone technology are getting popular in the recent days and also the research to bring the two field together. Many problems inherent in the drones today can be solved with the use of artificial intelligence. This thesis can serve as a guide for anyone who wants to start with artificial intelligence on the drone, but it also has many possibilities to improve. One of the most important ones is using reinforcement learning for training the drone to fly. There is much ongoing research on imitation learning, and this can be a viable extension to the current thesis. Furthermore, we are only talking about image segmentation and using monocular vision for obstacle avoidance and navigation, but drones have many sensors build with them, and input from these sensors can also be used for predicting several scenarios like accidents or crashing and prevent such emergency situations intelligently. Also, for navigation and obstacle avoidance other sensors such as sonar, radar or lidar can also be used to train for better results and performance. Use case specific research such as agriculture or industry surveillance can be developed by analyzing the video from the drone.

# Chapter 8

# Conclusions

A robust obstacle avoidance system was integrated to ardupilot and tested. The shortcomings of available obstacle avoidance system were mitigated with the new design. With the availability of AI and deep learning model, obstacle avoidance system can have a better understanding of nature of obstacles. This information can help in better analyzing the obstacles and calculating the evasive route. End to end deep neural network model can also be used for navigation purposes. Such model can generate a pose directly based on input images.

# Bibliography

[1] Mavlink micro air vehicle communication protocol. `http://qgroundcontrol.org/mavlink/start`. Accessed 11.29.2017.

[2] Px4 autopiot. `https://pixhawk.org/`. Accessed 11.8.2017.

[3] Raspberry pi. `https://www.adafruit.com/product/3055`. Accessed 16.6.2017.

[4] Up bridge the gap. `http://www.up-board.org/up/`. Accessed 2.12.2018.

[5] A. Bork. Inertial measurement units, 2017. `https://tams.informatik.uni-hamburg.de/lehre/2014ws/seminar/ra/presentations/2014-12-01_andreas_bork-inertial_measurement_units.pdf`. Accessed 6.9.2017.

[6] Araujo dos Santos, L. Artificial intelligence. `https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/reinforcement_learning.html`. Accessed 10.9.2017.

[7] B. Chacos. Breaking moore's law: How chipmakers are pushing pcs to blistering new levels, 2013. `https://www.pcworld.com/article/2033671/breaking-moores-law-how-chipmakers-are-pushing-pcs-to-blistering-new-levels.html`. Accessed 20.12.2017.

[8] Behere, S., and Torngren, M. A functional architecture for autonomous driving. In *Automotive Software Architecture (WASA), 2015 First International Workshop on* (2015), IEEE, pp. 3–10.

[9] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[10] Borenstein, J., and Koren, Y. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation 7*, 3 (1991), 278–288.

[11] Byrne, J., and Taylor, C. J. Expansion segmentation for visual collision detection and estimation. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on* (2009), IEEE, pp. 875–882.

[12] Caffe. Model zoo cafe. =https://github.com/BVLC/caffe/wiki/Model-Zoo. Accessed 24.11.2017.

[13] Chakravarty, P., Kelchtermans, K., Roussel, T., Wellens, S., Tuytelaars, T., and Van Eycken, L. Cnn-based single image obstacle avoidance on a quadrotor. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (2017), IEEE, pp. 6369–6374.

[14] Corporation, I. Collision avoidance library, 2016. `https://github.com/intel/collision-avoidance-library`. Accessed 9.6.2017.

[15] Corporation, I. Intel realsense cross platform api, 2016. `https://github.com/IntelRealSense/librealsense/tree/v1.12.1`. Accessed 9.6.2017.

[16] Corporation, N. Aerial drone dataset, 2017. Download available: `https://nvidia.box.com/shared/static/ft9cc5yjvrbhkh07wcivu5ji9zola6i1.gz`. Accessed 12.9.2017.

[17] Corporation, N. Deep learning gpu training system (digits), 2017. `https://github.com/NVIDIA/DIGITS`. Accessed 14.12.2017.

[18] D. Hart. Introduction to global positioning systems gps, Feb 2007. `https://aqua.wisc.edu/CPR/Default.aspx?tabid=80`. Accessed 6.9.2017.

[19] DeepLearning.net. Convolutional neural networks (lenet), Dec 2015. `http://deeplearning.net/tutorial/lenet.html`. Accessed 10.12.2017.

[20] Documentation. Vector field histogram, 2017. `https://se.mathworks.com/help/robotics/ug/vector-field-histograms.html`. Accessed 5.3.208.

[21] Dodge, S. F., and Karam, L. J. A study and comparison of human and deep learning recognition performance under visual distortions. *CoRR abs/1705.02498* (2017).

[22] Driss, S. B., Soua, M., Kachouri, R., and Akil, M. A comparison study between mlp and convolutional neural network models

for character recognition. In *SPIE Conference on Real-Time Image and Video Processing* (2017), vol. 10223.

[23] Editors, S. Tech alert: Nhtsa, sae define 5 levels of vehicle automation, Mar 2017. https://www.sema.org/sema-enews/2017/11/ettn-tech-alert-nhtsa-sae-define-5-levels-of-vehicle-automation. Accessed 26.11.2017.

[24] ES310 Introduction to Naval Weapons Engineering. Introduction to sonar, 1998. https://fas.org/man/dod-101/navy/docs/es310/uw_acous/uw_acous.htm. Accessed 10.6.2017.

[25] F. Corrigan. 10 top lidar sensors for uavs and so many great uses, Jun 2017. https://www.dronezon.com/learn-about-drones-quadcopters/best-lidar-sensors-for-drones-great-uses-for-lidar-sensors/. Accessed 15.6.2017.

[26] Foundation, O. S. R. Gazebo simulator. http://gazebosim.org/. Accessed 25.9.2017.

[27] games, B. Python 3 programs versus c++ g++. https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp. Accessed 10.1.2018.

[28] Gandhi, D., Pinto, L., and Gupta, A. Learning to fly by crashing. *CoRR abs/1704.05588* (2017).

[29] Garmin. Lidar-lite v3, 2017. https://buy.garmin.com/en-US/US/p/557294. Accessed 1.4.2018.

[30] Giusti, A., Guzzi, J., Cireşan, D. C., He, F.-L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G., et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters 1*, 2 (2016), 661–667.

[31] Grave, R. The vehicle architecture of automated driving level 2/3, Jun 2017. http://www.embedded-computing.com/embedded-computing-design/the-vehicle-architecture-of-automated-driving-level-2-3. Accessed 30.6.2017.

[32] guiccbr, anselmolsm, and rchiossi. Gazebo sitl plugin. Hosted online at https://github.com/intel/gazebo-realsense. Accessed 20.9.2017.

[33] guiccbr, anselmolsm, rchiossi, mbelluzzo, and dakerfp. Gazebo realsense plugin, 2017. Hosted online at `https://github.com/intel/gazebo-realsense`. Accessed 20.9.2017.

[34] Honegger, D., Meier, L., Tanskanen, P., and Pollefeys, M. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (2013), IEEE, pp. 1736–1741.

[35] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014).

[36] Kahn, G., Villaflor, A., Pong, V., Abbeel, P., and Levine, S. Uncertainty-aware reinforcement learning for collision avoidance. *CoRR abs/1702.01182* (2017).

[37] Karhunen, J. Machine learning and neural networks, 2017. `https://mycourses.aalto.fi/pluginfile.php/381754/mod_resource/content/1/lect2-2016.pdf`. Accessed 16.12.2017.

[38] Karpathy, A. Convolutional neural network for visual recognition, 2017. `http://cs231n.github.io/neural-networks-1/`. Accessed 20.12.2017.

[39] Kelchtermans, K., and Tuytelaars, T. How hard is it to cross the room?–training (recurrent) neural networks to steer a uav. *arXiv preprint arXiv:1702.07600* (2017).

[40] Keselman, L., Woodfill, J. I., Grunnet-Jepsen, A., and Bhowmik, A. Intel realsense stereoscopic depth cameras. *CoRR abs/1705.05548* (2017).

[41] Kleeman, L. Understanding and applying kalman filtering. `http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf`. Accessed 5.3.2018.

[42] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[43] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[44] Liang, O. Build a quadcopter from scratch hardware anatomy, Mar 2017. https://oscarliang.com/build-a-quadcopter-beginners-tutorial-1/. Accessed 5.3.2018.

[45] Liu, S., Tang, J., Zhang, Z., and Gaudiot, J.-L. Computer architectures for autonomous driving. *Computer 50*, 8 (2017), 18–25.

[46] Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 3431–3440.

[47] Muoio, D. These 19 companies are racing to put driverless cars on the road by 2020. http://www.businessinsider.com/companies-making-driverless-cars-by-2020-2016-8?r=US&IR=T. Accessed 3.3.2018.

[48] Murphy, J. An overview of convolutional neural network architectures for deep learning. https://pdfs.semanticscholar.org/64db/333bb1b830f937b47d786921af4a6c2b3233.pdf. Accessed 5.3.2018.

[49] Nielsen, M. A. Neural networks and deep learning, Aug 2017. http://neuralnetworksanddeeplearning.com/chap2.html. Accessed 20.11.2017.

[50] Nilsson, N. J. Artificial intelligence: A modern approach: Stuart russell and peter norvig,(prentice hall, englewood cliffs, nj, 1995); xxviii+ 932 pages, 1996.

[51] Nvidia Corp. Aerial drone dataset, 2017. Download link: https://nvidia.box.com/shared/static/ft9cc5yjvrbhkh07wcivu5ji9zola6i1.gz. Accessed 6.9.2017.

[52] Nvidia Developer. Nvidia jetson tx2 module, 2017. https://developer.nvidia.com/embedded/buy/jetson-tx2. Accessed 19.11.2017.

[53] Product Datasheet. Intel realsense camera r200, Jun 2016. Download link: http://www.mouser.com/pdfdocs/intel_realsense_camera_r200.pdf. Accessed 5.3.2018.

[54] RoboPeak. Rplidar a2 360 laser scanner, 2017. https://www.elprocus.com/radar-basics-types-and-applications/. Accessed 16.9.2017.

[55] Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (2013), IEEE, pp. 1765–1772.

[56] Salleb Aouissi, A. Artificial intelligence, 2017. `https://www.edx.org/course/artificial-intelligence-ai-columbiax-csmm-101x-4`. Accessed 10.9.2017.

[57] Smolyanskiy, N., Kamenev, A., Smith, J., and Birchfield, S. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. *CoRR abs/1705.02550* (2017).

[58] Sturm, J., Cremers, D., and Kerl, C. Autonomous navigation for flying robots, 2015. `https://www.edx.org/course/autonomous-navigation-flying-robots-tumx-autonavx-0`. Accessed 12.6.2017.

[59] T. Agrawal. Radar-basics, types applications, 2015. `https://www.elprocus.com/radar-basics-types-and-applications/` Accessed 16.9.2017.

[60] Team, A. D. Ardupilot autopilot suite. `http://ardupilot.org/ardupilot/index.html`. Accessed 8.10.2017.

[61] Team, A. D. Apm planner 2 home, 2016. `http://ardupilot.org/planner2/`. Accessed 2.14.2018.

[62] Team, A. D. Copter object avoidance, 2016. `http://ardupilot.org/dev/docs/code-overview-object-avoidance.html`. Accessed 2.1.2018.

[63] Team, A. D. Sitl simulator (software in the loop), 2016. `http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`. Accessed 2.1.2018.

[64] Thrun, S., Burgard, W., and Fox, D. *Probabilistic robotics*. MIT press, 2005.

[65] Tong, C. H., and Barfoot, T. D. A comparison of the ekf, spkf, and the bayes filter for landmark-based localization. In *Computer and Robot Vision (CRV), 2010 Canadian Conference on* (2010), IEEE, pp. 199–206.

[66] WIFIGURU10. Apm stack question, apm vs px4, Sep 2016. `https://discuss.ardupilot.org/t/apm-stack-question-apm-vs-px4/11497`. Accessed 5.3.2018.

[67] ZHANG, G., AND CHANSON, H. Application of local optical flow methods to high-velocity free-surface flows: Validation and application to stepped chutes. *Experimental Thermal and Fluid Science 90*, 186–199.