Design and Development of Autonomous Delivery Robot

Aniket Gujarathi, Akshay Kulkarni, Unmesh Patil, Yogesh Phalak, Rajeshree Deotalu,
Aman Jain, Navid Panchi

under the guidance of

Dr. Ashwin Dhabale
and
Dr. Shital S. Chiddarwar

Visvesvaraya National Institute of Technology
Nagpur 440 010(India)
2020

ABSTRACT

The field of autonomous robotics is growing at a rapid rate. The trend to use increasingly more sensors in vehicles is driven both by legislation and consumer demands for higher safety and reliable service. Nowadays, robots are found everywhere, ranging from homes, hospitals to industries, and military operations. Autonomous robots are developed to be robust enough to work beside humans and to carry out jobs efficiently. Humans have a natural sense of understanding of the physical forces acting around them like gravity, sense of motion, etc. which are not taught explicitly but are developed naturally. However, this is not the case with robots. To make the robot fully autonomous and competent to work with humans, the robot must be able to perceive the situation and devise a plan for smooth operation, considering all the adversities that may occur while carrying out the tasks. In this thesis, we present an autonomous mobile robot platform that delivers the package within the VNIT campus without any human intercommunication. From an initial user-supplied geographic target location, the system plans an optimized path and autonomously navigates through it. The entire pipeline of an autonomous robot working in outdoor environments is explained in detail in this thesis. We have addressed the problem of semantic segmentation for road and obstacle detection. The common networks used in the literature are reported, along with some motivation for each of them. A general requirement for autonomous navigation is the availability of a high-definition map. The different layers of maps are discussed and a map of the VNIT campus with required details is constructed. The issue of robust localization and sensor fusion is explained in detail in this thesis. The problem of the need of 360-degree vision to the autonomous vehicles is also discussed. Catadioptric cameras that output panoramic views images with very large fields of view. It turns out that the design of such cameras solves plenty of problems including creating a 3D point cloud and providing preliminary visual data to the obstacle detection and motion planning. The proposed solution is characterized by an intricate mixture of optics and geometry as exemplified.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The field of autonomous robots is growing rapidly in the world, in terms of both the diversity of emerging applications and the levels of interest among traditional players in the automotive, truck, public transportation, industrial, and military communities. Autonomous robotic systems offer the potential for significant enhancements in safety and operational efficiency. Due to the meteoric growth of e-commerce, developing faster, more affordable and sustainable last-mile deliveries become more important. Many challenges like reduced capacity, driver shortage, damaged and stolen products, failed delivery attempts, increased traffic congestion, etc. can be solved using autonomous robots. An autonomous robot is designed and engineered to deal with its environment on its own, and work for extended periods of time without human intervention. It must not only carry out its task of delivery properly, but must also consider the various scenarios changing around it and act accordingly. The robot must make quick decisions even in adverse conditions, considering the safety of pedestrians around it[1]. The aim of autonomous robots is to work alongside humans and try to make human life easier.

Currently, many robots are being used in industries [2], homes [3], military applications, disaster management [4], etc., all around the world. The advancements in robotics has made lives easier for humans in many aspects and it provides with a safer and more efficient alternative to perform tasks which are difficult or time consuming for humans. Some of the applications of autonomous robots include cleaning robots like Roomba, delivery robots, autonomous vehicles, and other robots that move freely around a physical space without being guided by humans [5].

In order to make a robot completely autonomous, the robot must be completely cognizant of its surroundings and must be able to perform actions based on the inputs it receives through various modules of the system. For the purpose of achieving a state of complete autonomy, the robot must be able to take information from sensors, perceive the environment, localize itself precisely in the world, and finally devise an optimal plan to achieve its goal. These instructions achieved from the modules mentioned above must be integrated by the robot in real-time and be given to a control node to actually move the system in the real world. The system pipeline of an autonomous robot is shown in Fig. 1.1.

The accuracy and the proper integration of all the modules is of utmost importance for an autonomous robot to operate. A fault in any of the module may cause serious repercussions and may even pose a hazard to humans around it. This thesis aims to implement all the mentioned modules flawlessly in order to achieve a completely autonomous operation of the robot in outdoor environments.

This thesis is organized as follows :

In Chapter 2, the hardware design criteria with the applicable constraints is depicted. Furthermore, the hardware structure including the cyberphysical architecture of the robot is described. The schematic of the power system as well as the renderings of CAD models are illustrated.

Maps are an integral part of an autonomous operation pipeline. Delivery robots need an accurate map suitable for accurate localization, navigation and planning. Furthermore, these maps have to be able to incorporate and respond to the changes in the environment. The standard maps like Google Maps, used by humans to navigate the world cannot be used for the purpose of navigation of an autonomous robot. Chapter 3 illustrates the construction of specialized maps for the purpose of autonomous navigation.

Chapter 4 describes the problem of state estimation and localization of a robot in detail. In order to navigate accurately around the world, the robot must know its location in the world and the map exactly. A robot can move smoothly only if it is properly localized. An inaccurate localization may cause the robot to vary off the roads or behave erroneously which are serious issues when the robot is completely autonomous.

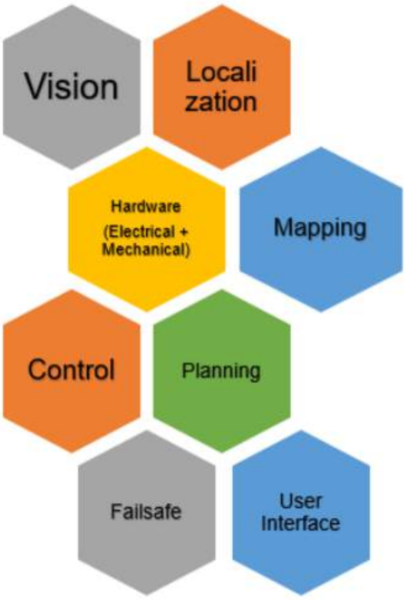


Figure 1.1: System Pipeline

The planning module is the backbone of an autonomous driving pipeline. A planner is responsible for finding optimal paths in the map for the robot to move and generating efficient trajectories and velocity profile for the robot to move locally in the presence of static/dynamic obstacles, obey lane rules, prioritize safety of humans, etc. Chapter 5 describes the hierarchical planning structure adopted for the autonomous driving pipeline and the implementation details of mission planner and local planner.

Chapter 6 addresses the problem of semantic segmentation for road and obstacle detection. This problem involves separating sets of pixels from an image where each separate set has some common attributes. Due to the complexity of the task and the availability of large datasets (with images and corresponding labels), most modern techniques use Supervised Deep Learning. Thus, some of the common networks used in the literature are described, along with some motivation for each of them. Following this, implementation and results of road segmentation are given.

Chapter 7 addresses first in great depth the problem of the need of 360-degree vision to the autonomous vehicles, designing new solutions including catadioptric cameras that output panoramic views of the scene, i.e., images with very large fields of view. It turns out that the design of such cameras solves plenty of problems including creating a 3D point cloud and providing preliminary visual data to the obstacle detection and motion planning. The proposed solution is characterized by an intricate mixture of optics and geometry as exemplified in the second and third sections of the Chapter 7.

Chapter 8 describes the process of intrinsic and extrinsic calibration of the camera-Lidar system.

Chapter 2

Hardware Design

2.1 Overview

The robot is designed to operate as an autonomous mobile robot platform for different applications like in industrial areas for transportation, in hospitals for carrying food and medicines, in unmanned missions and also for security purposes. Our aim is to develop an autonomous delivery mobile robot that can deliver a package autonomously from A to B within the VNIT campus. In this chapter, the design criteria and system architecture are described.

2.2 Design Criteria

There were five major constraints in hardware design. These are outlined below:

1. Modularity: In order to easily add units or parts to the robot, the robot platform has to be modular. These units or parts may be additional navigation sensors, room for payload, extra on-board power, or various devices for effective human-machine interface.
2. Low-cost Production: Even though mobile robots are available in the market, they tend to be expensive, thus increasing research and development costs. Further, the use of a ready-made robot will increase the cost of production even more in case of mass volume production.
3. Truncated Construction: In the prototyping phase, the construction is kept simple and truncated in order to use minimal resources and to focus on designated functionality.
4. Suitability of Environmental Conditions: Robot is planned to be used in outdoor environments, which means that the robot has to move on the roads and be able to pass over small obstacles. Additionally, electronic equipment on the robot must be protected.
5. Originality: To contribute to scientific research and development, the robot has to be different and new.

The hardware and software structure of the robot has been designed by taking into consideration the above criteria.

2.3 Hardware Structure

2.3.1 Hardware Overview

The hardware structure of the robot was made by considering the design requirements. Fig. 2.1 shows the anatomy of the robot. Green lines symbolize signal and communication connections, while red and orange lines are main power connections and blue lines are connections between motors and motor drivers. In this section, every part of the robot is described below according to the design progress.

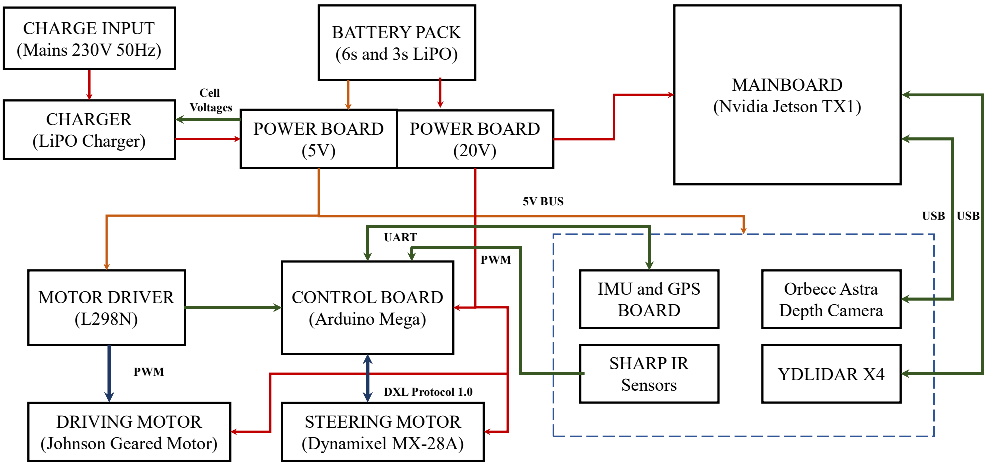

CHARGE INPUT (Mains 230V 50Hz)
BATTERY PACK (6s and 3s LiPO)
MAINBOARD (Nvidia Jetson TX1)
CHARGER (LiPO Charger)
Cell Voltages
POWER BOARD (5V)
POWER BOARD (20V)
5V BUS
USB
USB
UART
PWM
MOTOR DRIVER (L298N)
CONTROL BOARD (Arduino Mega)
IMU and GPS BOARD
Orbecc Astra Depth Camera
SHARP IR Sensors
YDLIDAR X4
PWM
DXL Protocol 1.0
DRIVING MOTOR (Johnson Geared Motor)
STEERING MOTOR (Dynamixel MX-28A)

Figure 2.1: Cyberphysical Architecture of the Robot

2.3.2 Driving System

Firstly, the driving system was designed for the robot. The driving structure is made up of the chassis of the robot, geared motor, Dynamixel smart servo motor, and motor driver. Pre-built chassis was used to accelerate the design and implementation steps. Chassis consists of an aluminum alloy skeleton, spring suspension, gearbox, and Ackerman steering mechanism. It was feasible to turn the available driving shaft by adding a gear to the chassis' gearbox. The 300 rpm Johnson geared motor is placed with the 1:1 gear ratio on the custom made motor mount to drive the shaft. Similarly, the steering mechanism is operated by the MX-28 Dynamixel Smart servo motor.

2.3.3 Power System

The power system was designed according to the requirements of the drive system. The Power system has three parts. These parts are battery pack, power board, and charger. It has two Lithium Polymer (LiPo) battery packs to power its system (6s and 3s). LiPo batteries have higher capacity compared to other battery types in the same size and weight. But LiPo batteries have safety issues. For this reason, LiPo batteries have to be monitored while charging and discharging. The 5V bus is added to the system to power the auxiliaries such as sensors and motor drivers. The battery management system monitors battery status by measuring battery voltages, battery temperatures, and the current which is drawn from the battery pack. The power board MCU in LiPo charger cuts the power in the event of a dangerous situation, such as overvoltage or short-circuit, while charging or discharging batteries. The power panel is provided on the side of the robot to place ON/OFF switches and shifting between charging/discharging modes. The power system schematic of the robot is as given in the Fig. 2.4.

2.3.4 Controller and driver boards

The total three controllers and driver boards mounted on the robot are given as follows.

1. Mainboard (Nvidia Jetson TX1): Mainboard is the brain of the robot. Nvidia Jetson TX1 is used for higher-level control. ROS is used for communication between the various modules. It's Nvidia Maxwell GPU (256 CUDA cores) enables fast inference times for deep neural networks. It runs all other processes like the global planning algorithm, the local planning algorithm, control algorithm, and so on. Further stages of the work, control methods and sensor fusion algorithms will be implemented on the mainboard.

2. Control board (Arduino Mega): Arduino Mega is used for lower-level control. It controls both the motors (main drive and steering), and handles all the sensors. It communicates with the main processor (Nvidia Jetson TX1) through rosserial to get commands for the motors and to publish sensor data.

3. DC motor driver (L298): The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage. The Johnsons geared DC motor is driven by this board.

2.3.5 Sensors

The sensors used in the robot are as follows:

1. Orbecc Astra Depth Camera: RGBD camera mounted to get front view image and front 3D depth map for perception and planning. It has a range of 8 meters for the 3D depth map. Useful for localization (using visual odometry) and for identifying obstacles.

2. YDLIDAR X4: Laser range finder which gives a 2D (planar) 360 degrees depth map, used for perception and planning. It has a 10 meters scanning range. Useful for localization of robot and identifying obstacles.

3. SparkFun IMU Breakout MPU9250: An inertial measurement unit (IMU). It consists of a 3-axis accelerometer, 3-axis gyroscope, and a 3-axis magnetometer. Useful for localization of robot.

4. SHARP IR Sensor: A distance measuring sensor, to be used as the last line of defense against collisions. It has a range of 4-30 cm.

5. Neo-M8N GPS Module: Gives global position (latitude, longitude, and altitude) useful for global planning. The used Neo-M8N GPS Module provides 167 dBm navigation sensitivity and supports all satellite augmentation systems.

The connection schematic circuit diagram of the non optical sensors (except depth camera and LiDAR) is given in Fig. 2.5.

2.3.6 Level Design

The hardware design of the robot is performed at three levels. As seen from Fig. 2.1, there are too many parts on the robot, and one of the design criteria is modularity. These levels are as follows:

1. Body level (Fig. 2.2 b): This level consists of robot chassis, motors, batteries, temperature sensors, and control board. Parts in the body level are stationary and unique to the robot platform.

2. Control unit level (Fig. 2.3 b): Powerboard, mainboard, 5V DC bus, motor driver board, and distance sensors are placed at this level. This level is detachable, so that changes can be made. Control unit level can be used on any other platform, as long as the motor driver and the battery are fitted.

3. Rooftop level (Fig. 2.3 c): This level is designed for optical sensors. The camera and LiDAR is placed in this level to ensure no blockage in the range. In later stages of this work, additional navigation components and application-specific equipment can be added to this level.

2.4 Conclusion

In conclusion, the robot which is seen in Fig. 2.6 is designed and built according to design criteria and open field tests are started. The environmental considerations are fully met and a robust structure has been developed. The robot weighs about 5kg and has a payload capacity of 2kg.

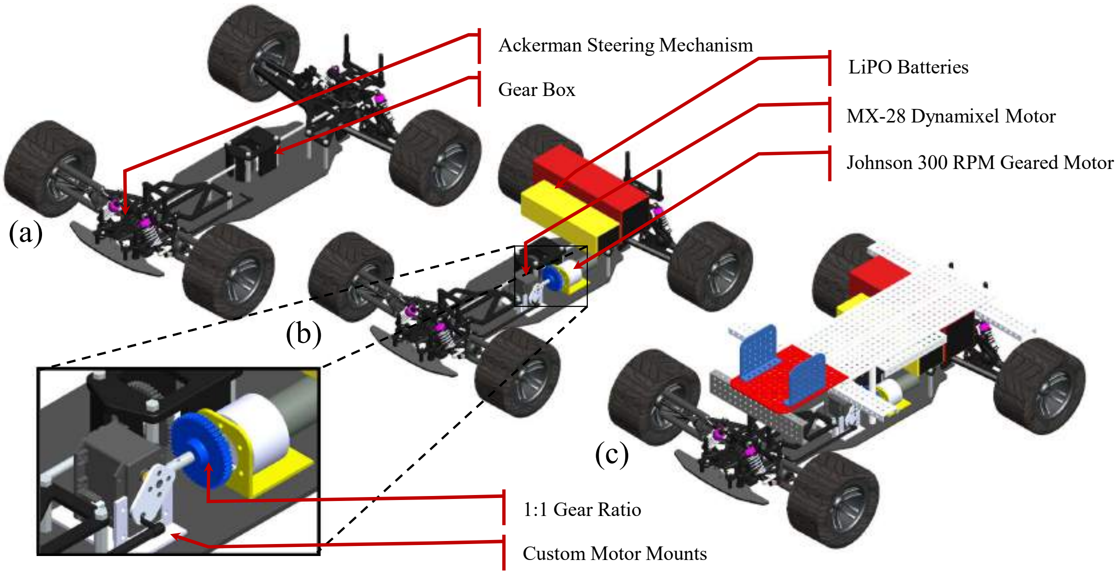


Figure 2.2: (a) Pre-built chassis of 1:10 scaled RC car, (b) Mounted motors and LiPO batteries, (c) Aluminium plate covering frame for stage 1.

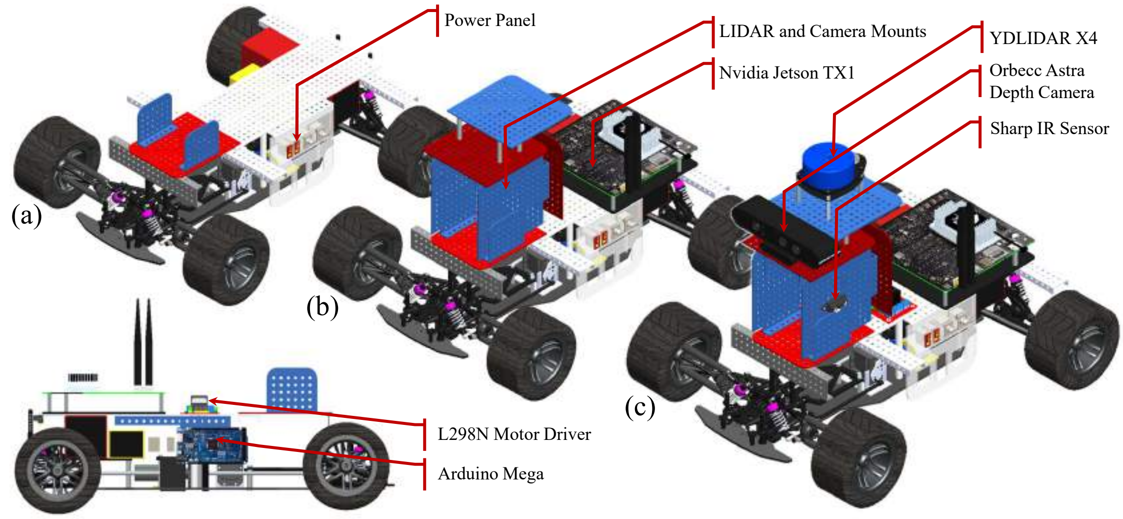


Figure 2.3: (a) Power panel mounted on the frame, (b) LIDAR and Camera mounts, (c) Final version of the Robot

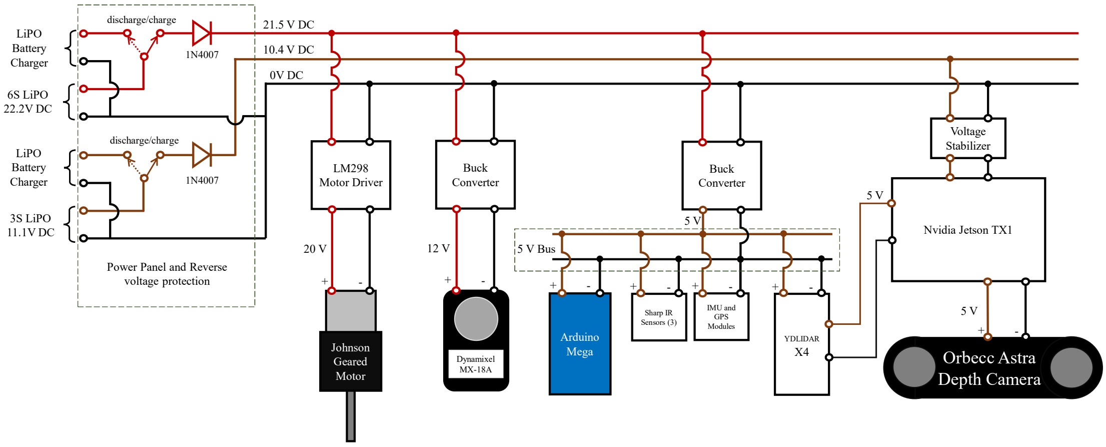


Figure 2.4: Power diagram of the robot

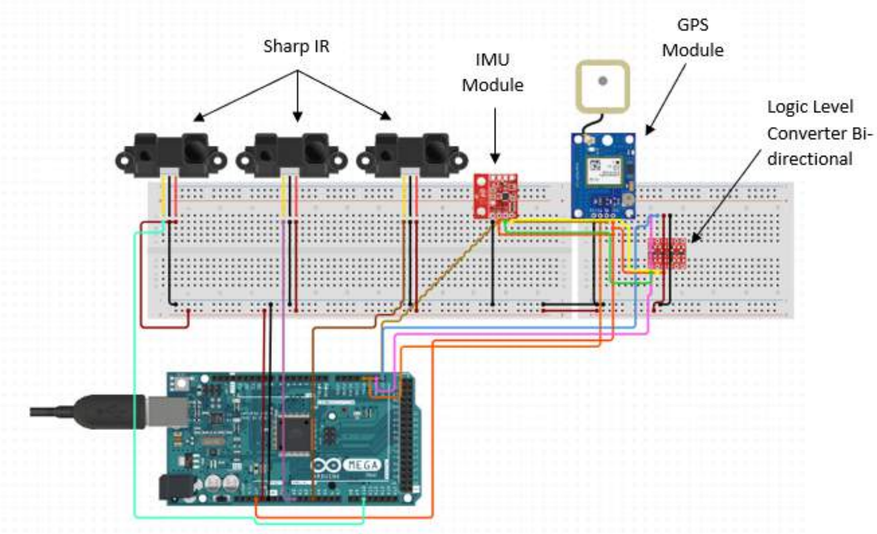


Figure 2.5: Connection schematic circuit diagram of the sensors.

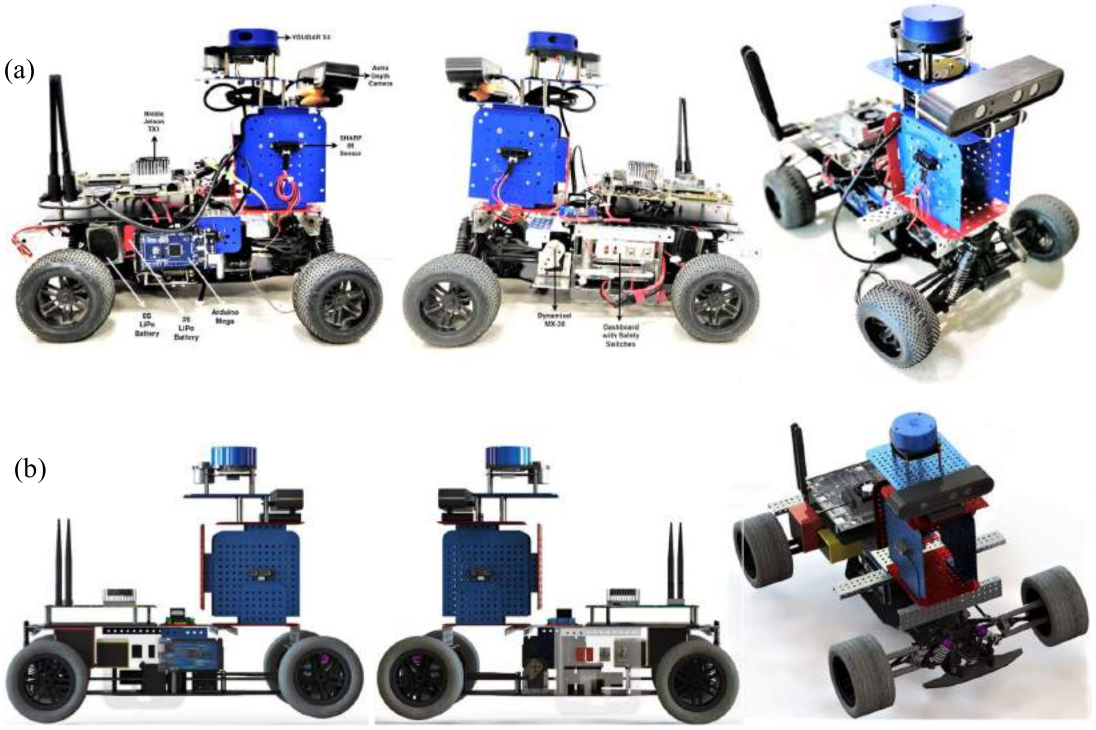


Figure 2.6: (a) The final hardware of the Autonomous delivery robot, (b) The rendered images of respective CAD models.

Chapter 3

Building and Managing Maps for Autonomous Operation

3.1 Overview

One of the major aspects for the navigation of autonomous robots in outdoor environments is the availability of a specialized high-definition (HD) maps. Many web map services like Google Maps in existence are designed specifically for the purpose of humans to navigate the world. However, such maps offer a location resolution of up to a few meters which cannot be used for the purpose of navigating an autonomous robot due to safety reasons, errors, lack of details and information, etc. HD-maps also known as ADAS maps or Vector maps are generally used for autonomous navigation applications. Some benefits of HD-maps are :

1. High accuracy of object locations, upto 10cm.
2. Consist of multiple layers of information about the lanes, which way they travel, road intersections, curbs, 3D point cloud of the environment, etc.

Using the information in the HD-maps, the robot can localize itself in the map and plan a path for navigating around the environment. However, existence of a map prior to starting of the operation is not always necessary. Another way of solving this problem of localization and mapping is through SLAM (Simultaneous Localization and Mapping) [6]. The purpose of SLAM is to generate a map and using the information in the map to simultaneously deduce the location of the robot in the map. The SLAM problem is still an active field of research and is widely used in autonomous robots ranging from indoor robots to outdoor robots, airborne systems and underwater robots.

However, the implementation of SLAM is not in the scope of this thesis. A pre-built map is generated for the purpose of autonomous navigation. The visualization of HD-maps classified in layers is represented in figure 3.1.

For the purpose of this thesis, we will focus mainly on creating a standard definition base map containing 2D geometric information and the semantic map discussed in Chapter 6.

3.2 Literature Review

3.2.1 Map Layers

A map is useless to a robot if it contains no information about its environment. However, by increasing the richness of information in an HD-map leads to an increase in the size of the map. As a result, more processing power would be required to analyze all the information and perform an action based on the information. Hence, what is included in an HD-map and what isn't is decided based on the purpose of the application. For example, a self-driving car would require very high-precision and a detailed map, as it has to take into consideration a lot of factors like the safety of pedestrians, staying in lane, follow traffic rules, etc.

The different map layers in an HD-map as mentioned [8], [7] are:

- Standard Definition Map Layer
- Geometric Map Layer
- Semantic Map Layer
- Map Priors Layer

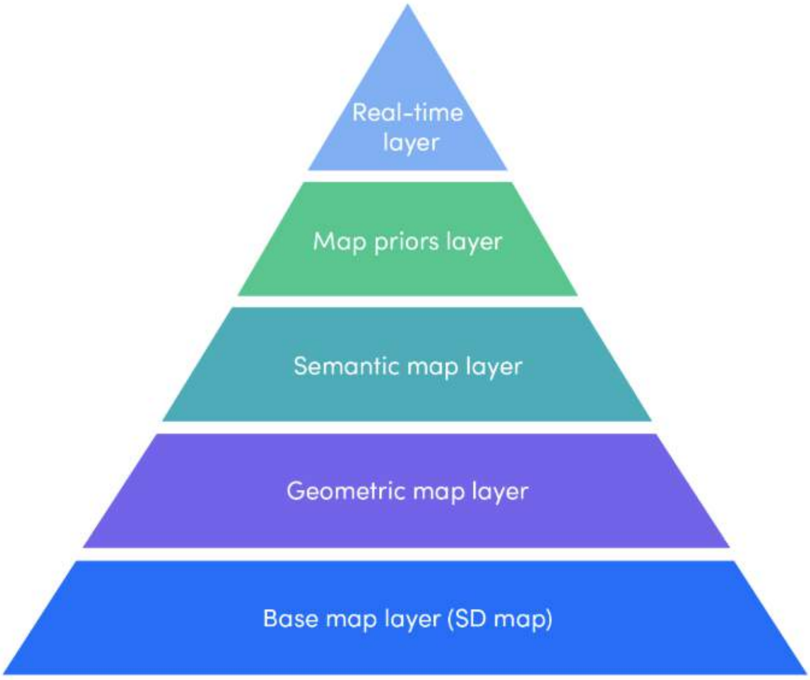


Figure 3.1: HD-map Layers, Source:[7]

Standard Definition Map Layer

At the foundation of our map layer is the standard definition map layer. This represents all of the road segments and the interconnections, how many lanes there are, what direction they travel in, and the entire network of roads. It helps the robot to understand the basic attributes of the environment it is navigating in. Although an SD-map contains the basic information about the roads and their basic details, it is not sufficient to smoothly navigate a robot autonomously. The SD-map is used as a base for the other map layers and is used to perform global path-planning for the robot.

The SD-map of VNIT campus containing the information about lanes, the direction of the vectors, information about curbs, etc. was mapped using OpenStreetMap(OSM) [9]. It is an editable map database built and maintained by volunteers and distributed under the Open Data Commons Open Database License. The map created for this project is shown in Figure 3.2

Geometric Map Layer

The geometric map layer contains 3D information of the world. This information is organized efficiently to support precise calculations. The 3D map is constructed by fusing the raw information from sensors like Lidar, depth camera, IMU readings, GPS readings, etc. The 3D point cloud achieved is then post-processed to produce the corresponding objects in the geometric map. During real-time processing, the geometric layer is used to access the point cloud information.

However, the sensors used to construct this layer require high precision and good resolution, hence are expensive. Our pipeline uses the information obtained from standard definition maps to localize the robot and for path planning.

Semantic Map Layer

The semantic map layer builds on the geometric map layer by adding semantic objects. Semantic objects include 2D and 3D objects found in the surroundings such as intersections, lanes, traffic signs, etc. For this project, we have a well-segmented road robust to various lighting conditions as explained in Chapter 6.

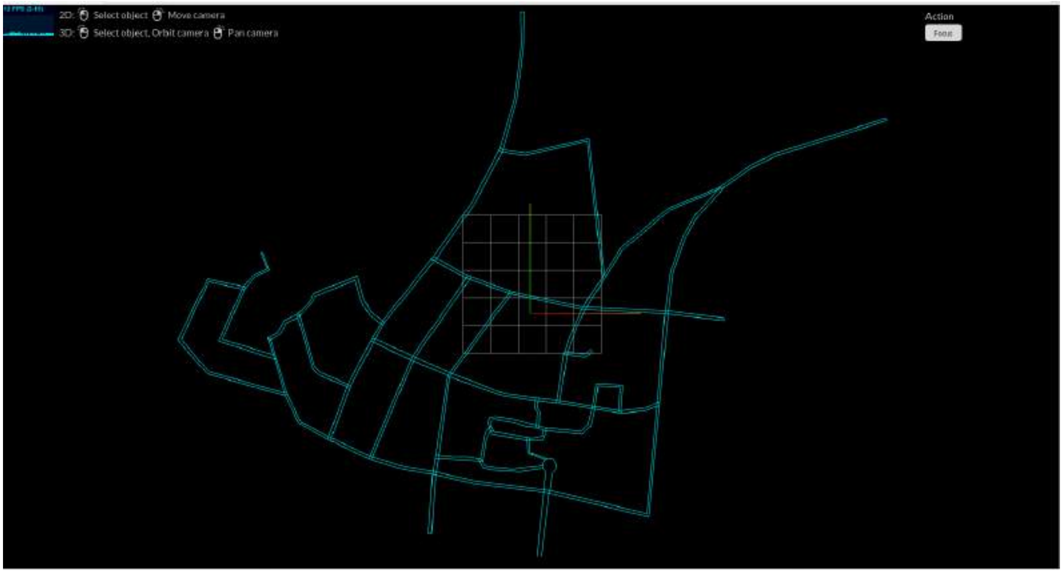

2D: Select object Move camera
3D: Select object, Orbit camera Pan camera
Action
Focus

Figure 3.2: Map of VNIT campus

Map Priors Layer

The map priors layer contains derived information about dynamic elements. Information here can pertain to both semantic and geometric parts of the map. These priors are used by the prediction and planning systems to determine the behaviour of the objects like traffic lights, the time to spend in a state, etc. and act accordingly. For the case of this project, we are assuming an ideal scenario without taking into consideration the various complexities involved while driving.

3.3 Implementation

An SD map of VNIT is generated using the software JOSM as shown in Figure 3.2 containing the features such as the lanes, their direction, curbs, etc. One of the core elements of the OSM data model are the nodes. Nodes are characterized in the data with latitude, longitude and a unique node-id. Some nodes can be assigned special tags in the form of a key-value pair to describe some physical features in the map like building, road, highway, etc. These tagged nodes are used as reference for planning the path to specific locations in the map. The tagged nodes are added in the map manually and the remaining nodes are populated automatically through the JOSM software. The data from the OSM is downloaded in '.xml' format.

To visualize the osm data, we need a visualizer which can interpret the data in the '.xml' format and display the map accordingly with the help of markers. Using the open-source ROS node $osm\_cartography$ from the package $open\_street\_map$, the map can be visualized in the visualizer rviz (rviz is a 3D visualizer for the Robot Operating System (ROS) framework). Simply visualizing the map is not enough. The map is further used by the localization and path planning algorithms and hence its accuracy is of utmost importance.

Chapter 4

State Estimation and Localization

4.1 Overview

Localization is the method by which we estimate the state of a robot within the world. In order to be fully functional, a mobile robot must be capable of navigating safely through an unknown environment while simultaneously carrying out the task it has been designed for. For the purpose of autonomous navigation, the robot has to know where it is in the real world with respect to the map, either provided initially or built simultaneously. However, the robot cannot completely rely on the sensors for accurate localization due to the errors inherent in the sensors. For example, a GPS (Global Positioning System) has an error magnitude in metres, an IMU (Inertial Measurement Unit) readings drift over time and its errors accumulate. Hence, such sensors cannot be trusted to give accurate information about the states of the robot. However, by combining the information obtained by various sensors and using probabilistic filters to reduce the errors due to the sensor readings, a better estimate of the states can be obtained. Based on the prior information about the state of the robot, the robot can estimate the current state and localize itself accordingly. The position and orientation can be considered as the state of the robot. An inaccurate localization can lead to system failure, erratic behaviour of the robot and could cause safety hazards to the people around it. Hence, it is of utmost importance to accurately localize the robot in the environment and reduce the errors accumulated due to faulty sensors.

4.2 Literature Review

The problem of localization can be approached by two methods [10]:

1. Map based Localization - Map is available prior to the process of localization
2. Simultaneous Localization and Mapping - The pose of a robot and the map of the environment are estimated at the same time.

For this project, a map based localization approach is adopted.

Generally sensors like a GPS or GNSS are used to estimate the position in the world using the method of trilateration. However, a GPS may have an error from 1 - 10 metres. The errors may be attributed to a number of errors like :

1. Satellite Geometry
2. Satellite Orbits
3. Multipath Effect
4. Atmospheric Effects
5. Clock Inaccuracies and Rounding Errors

For the application of an autonomous robot, such errors are not sustainable. Similarly, other sensors attached to the robot like an IMU sensor, Lidar, wheel encoders, vision sensors, etc. give certain information about the states of the robot either directly or indirectly. Every sensor consist of some uncertainty or errors. Accumulation of such errors may cause the robot to behave in an aberrant manner and could cause extreme fatalities while operating alongside humans or deviate from its desired path. Hence, due to the uncertainty in readings, it is impossible to accurately calculate the state of the robot using a deterministic algorithm. However, if the information obtained by all the sensors is fused together and using a stochastic approach, a better estimation of the states could be achieved. This is known as sensor fusion. Sensor fusion can be defined as the combination of sensory data or data derived from disparate sources such that the resulting information has less uncertainty than would be possible when these sources were used individually.

To estimate the states of the robot and reduce the uncertainty in measurements accumulated by the sensors, probabilistic filters are used. Some of the common probabilistic filters used for localization are :

1. Bayes Filter [11]
2. Kalman Filter [12]
3. Extended Kalman Filter (EKF) [13]
4. Error State EKF (ES-EKF) [14]
5. Unscented Kalman Filter (UKF) [15]

4.2.1 Probabilistic Estimators

Kalman Filter

Kalman Filter is one of the most widely used probabilistic estimator algorithm. It can be used in all the fields where there is an uncertainty in determining the state of a dynamical system. Using a Kalman filter, an educated guess can be made about the state of the system. The Kalman Filter makes use of the 'prediction' and 'correction' cycle iteratively to estimate the state of the system. Kalman filters are ideal for systems which are continuously changing. They have the advantage that they are light on memory as they don't need to keep any history other than the previous state, and they are very fast, making them well suited for real time problems and embedded systems.

A Linear Kalman filter is considered to be the best linear unbiased filter. This means, there is no estimator for the state which has a linear state model which is better. It assumes the noise is Gaussian. If the noise is Gaussian, then the Kalman filter minimizes the mean squared error of the estimated state parameters. Two assumptions are taken in the Linear Kalman filter :

1. Kalman Filter will always work with Gaussian Distribution.
2. Kalman Filter will always work with Linear Functions.

The algorithm for implementing a Kalman filter is shown in Figure 4.1

Unfortunately, systems in real life rarely show linear characteristics. A Kalman filter will give accurate results if the operating range is in the linear zone. Although, quite often, the systems are generally non-linear or have a small range for linear operation. In such cases, a Kalman filter may not be the best choice for an estimator. Hence, even if the Kalman filter is the best linear filter, it is not good enough for a non-linear system. Most real world problems involve non-linear functions and hence we would have to consider a suitable filter accordingly. For this case, a common non-linear filter used is the Extended Kalman Filter (EKF) [13].

In case of an EKF, the mean of the Gaussian on the non-linear curve is calculated and a number of derivatives are performed to approximate it using the Taylor's Theorem. As the function needs to be linearized, only the first derivative of the Taylor's series is considered. The algorithm of EKF is similar to the Kalman Filter. Hence for this project EKF is to be implemented for the purpose of localization.

Extended Kalman Filter

EKF is undoubtedly the most widely used non-linear estimator techniques that has been applied in the last decade. As mentioned by Lawrence Schwartz and Edwin Stear in [17]:

> 'It appears that no particular approximate [nonlinear] filter is consistently better than any other, though ... any nonlinear filter is better than a strictly linear one.'

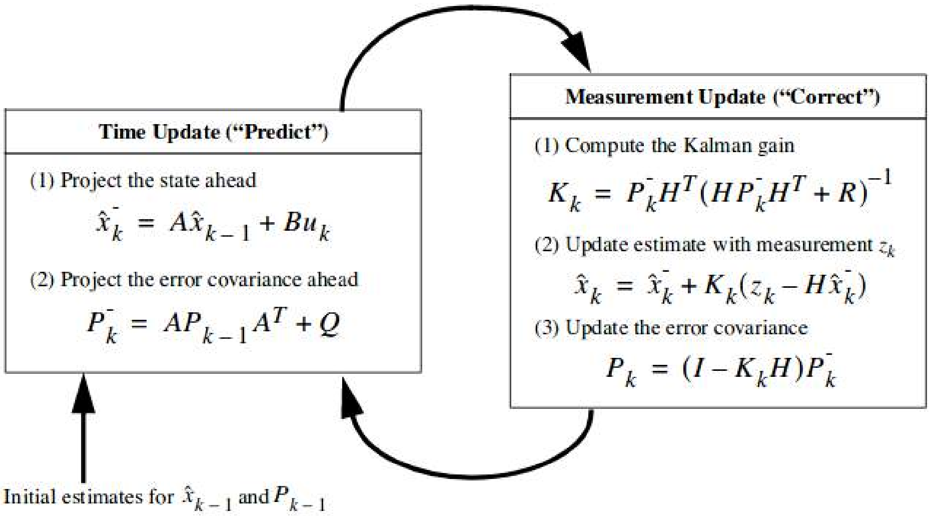

Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
(2) Project the error covariance ahead
$P_k^- = AP_{k-1}A^T + Q$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_k H)P_k^-$
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Figure 4.1: Kalman Filter Equations
Source:[16]

EKF is based on linearizing the non-linear functions using the first-order Taylor series expansion. Higher order approaches to non-linear filtering is also possible which provide better results than EKF, but at the expense of greater complexity and computational cost. Hence, EKF is generally used as it is a light-weight non-linear filter as compared to higher order estimators.

Consider the following general non-linear system:

$$\begin{aligned}\dot{x} &= f(x, u, w, t)\\ y &= h(x, v, t)\\ w &\sim (0, Q)\\ v &\sim (0, R)\end{aligned}$$

The system equation $f(.)$ and measurement equation $h(.)$ are non-linear functions, where x represents the states of the system, u is the input, w is the process or motion noise which is a Gaussian function with zero mean and Q covariance. v is the measurement noise with R covariance. A zero mean white Gaussian noise model is generally taken to mimic the random processes that occur in nature.

Taylor series is used to linearize the non-linear functions about a nominal control u_0, nominal state x_0, nominal output y_0 and nominal noise values w_0 and v_0. These nominal values are generally based on a priori guesses of what the system might look like. As mentioned in [18], in EKF the Kalman filter estimate is used as the nominal state trajectory. We linearize the nonlinear system around the Kalman filter estimate, and the Kalman filter estimate is based on the linearized system. As shown in figure 4.2, an operating point 'a' is selected and a linear approximation is carried out using the first-order Taylor series.

$$\begin{aligned}\dot{x} &\approx f(x_0, u_0, w_0, t) + \frac{\partial f}{\partial x}|_0(x - x_0) + \frac{\partial f}{\partial u}|_0(u - u_0) + \frac{\partial f}{\partial w}|_0(w - w_0)\\ &= f(x_0, u_0, w_0, t) + A\Delta x + B\Delta u + L\Delta w\\ y &\approx h(x_0, v_0, t) + \frac{\partial f}{\partial x}|_0(x - x_0) + \frac{\partial f}{\partial v}|_0(v - v_0)\\ &= h(x_0, v_0, t) + C\Delta x + M\Delta v\end{aligned}$$

After the linear approximation of the non-linear functions, the Kalman filter equations can be applied to get the estimate as mentioned in [18].

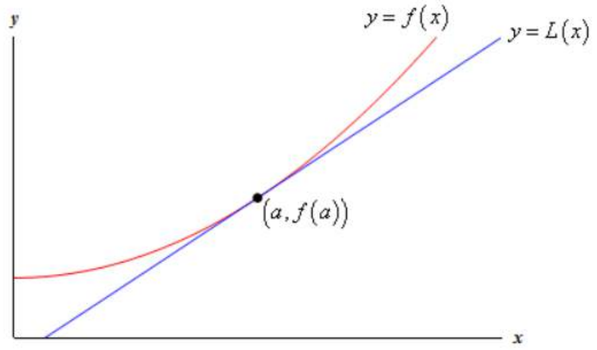


Figure 4.2: EKF Linearization
Source:[19]

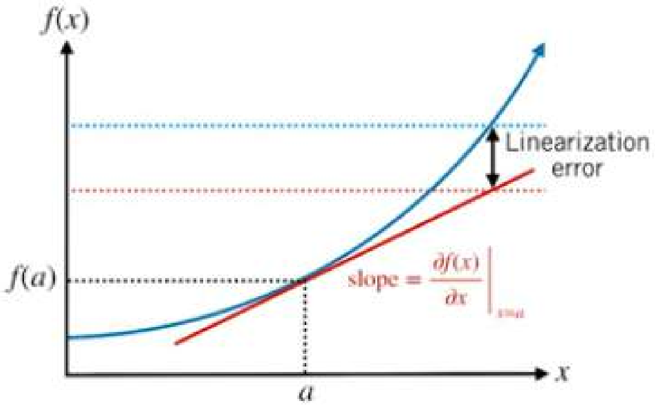


Figure 4.3: Linearization Error
Source:[20]

$$
\begin{aligned}
\hat{x_0} &= E[x(0)] \\
P(0) &= E[(x - x_0)(x - x_0)^T] \\
\dot{\hat{x}} &= f(\hat{x}, u, w_0, t) + K[y - h(\hat{x}, v_0, t)] \\
K &= PC^T \tilde{R}^{-1} \\
\dot{P} &= AP + PA^T + \tilde{Q} - PC^T \tilde{R}^{-1} CP \\
\tilde{Q} &= LQL^T \\
\tilde{R} &= MRM^T
\end{aligned}
$$

Although the EKF gives better results than a linear Kalman filter, it has several drawbacks as explained in [20]:

1. Linearization Error : The difference between the linear approximation and the non-linear function is called linearization error as shown in figure 4.3. The linearization errors generally depend on:
   (a) Non-Linearity of the function
   (b) How far away from the operating point the linear approximation is being used
2. Computing Jacobians :
   (a) Analytical differentiation is prone to human error.
   (b) Numerical differentiation can be slow and unstable.

(c) Automatic differentiation (e.g., at compile time) can behave unpredictably.

3. For highly non-linear functions, the EKF estimate can diverge and become unreliable.

There are several other filters like the particle filter, UKF, ES-EKF, etc. which give better performance than the EKF, but at the cost of higher computational requirements. Hence, there is always a trade-off between choosing the filter with better performance and the computational cost associated with it. As EKF is light-weight as compared to other filters and gives sufficiently good enough results, it is widely used as a suitable probabilistic estimator. Hence, for this project, we are going to use an EKF as the estimator used for localization.

4.3 Sensor Fusion

Sensor fusion is the method of combining the measurement readings from different sensors attached on the robot to get a better estimate. As discussed earlier, an autonomous robot cannot depend on the information provided by one or two sensors to localize itself precisely in the environment. Hence by extracting the information from multiple sensors and fusing the data using the probabilistic models explained earlier, we reduce the uncertainty and obtain better results.

The major question that arises is how many sensors are actually essential and how to choose the sensors required? Increasing the number of sensors surely increases the performance, but also increases the computational cost and also the overall cost of the robot. For the purpose of localization, we need a sensor to get the global position estimate, a sensor to determine the orientation of the robot and other sensors to find the odometry of the robot. In this project, we are using a GPS for getting the global position update, an IMU sensor to find the orientation, a 360° 2D range scanner (YDLIDAR X4) and a depth camera (Orbbec Astra) for this purpose. The mentioned sensors were chosen for the following reasons [21]:

1. The error dynamics are completely different and uncorrelated.
2. IMU provides smoothing of the GPS readings.
3. GPS provides absolute positioning information, which reduces the IMU drift.
4. The depth camera provides accurate local positioning within known maps.
5. Lidar provides odometry data in places where the camera fails to give output, e.g., in dark conditions, in presence of direct sunlight on the IR receiver, false loop closure, etc.

4.3.1 Global Positioning

The Global Positioning System(GPS) is a satellite-based navigation system consisting of a network of 24 orbiting satellites around the earth. The GPS is a US owned utility that provides the users with position, navigation and timing services. In order to obtain the location of a GPS receiver, it uses a process called trilateration. For the process of trilateration to work, at least four satellites must be visible to the location on earth. Each satellite transmits information about its position and current time at regular intervals. These signals, travelling at the speed of light, are intercepted by the GPS receiver, which calculates how far away each satellite is based on how long it took for the messages to arrive. Figure 4.4 shows the visualization of the process of trilateration used in a GPS.

For the purpose of an autonomous robot, the GPS plays a crucial role. However, as mentioned before, due to the errors in GPS readings, it can't be completely trusted to give accurate results.

4.3.2 Orientation Estimate

For the purpose of determining the orientation of the robot, we use an inertial measurement unit(IMU). An IMU is generally present in most of the robots and even smartphones. An IMU has wide scope of applications such as activity tracking, pose estimation, smart-phone applications, gaming, etc. As mentioned in [23], a basic IMU mainly consists of:

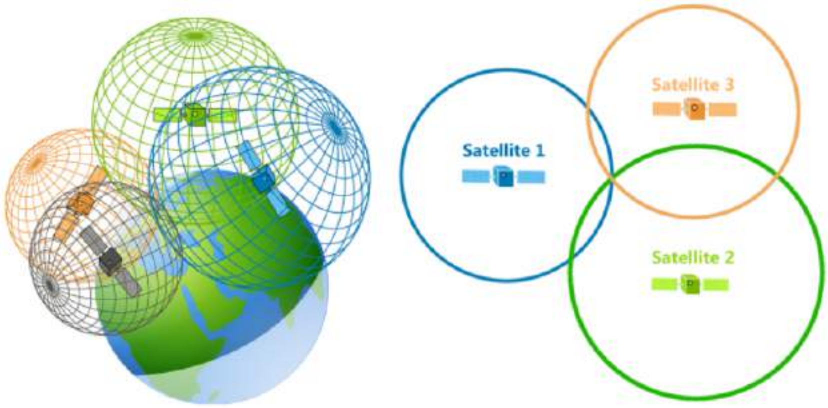


Figure 4.4: Trilateration
Source:[22]

1. Gyroscope which measures the angular rotation rates about three axes. A gyroscope can be considered as a spinning disc that maintains a specific orientation with respect to the inertial space, thus providing an orientation reference as shown in figure 4.5. Due to recent advancements in the field of Microelectromechanical systems(MEMS), the size and cost of a gyroscope have reduced drastically. However, these systems give noisy readings and the measurements drift over time. The measurement model of gyroscope can be given as :

$$\omega(t) = \omega_s(t) + b_{gyro}(t) + n_{gyro}(t)$$

where, $\omega_s(t)$ is the angular velocity of the sensor with respect to the reference frame, $b_{gyro}(t)$ is the gyro bias evolving over time and $n_{gyro}(t)$ is the white Gaussian additive noise term.

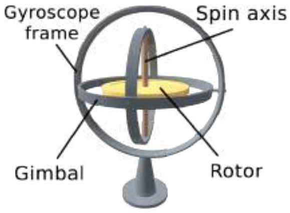


Figure 4.5: Gyroscope
Source:[24]

2. Accelerometer which measures the acceleration relative to the gravitational force, also known as specific force. An accelerometer can also be used to measure gravity as a downward force. Integrating acceleration once reveals an estimate for velocity, and integrating again gives you an estimate for position. However, due to the integration, even the errors get accumulated and give erroneous results over time. Hence, this technique of getting the velocity and position estimate using an accelerometer is not recommended.

3. Some IMUs also contain a magnetometer. It can detect fluctuations in Earth's magnetic field, by measuring the air's magnetic flux density at the sensor's point in space. Through those fluctuations, it finds the vector towards Earth's magnetic North. Using this data it can be fused with the accelerometer or gyroscope readings to get an absolute heading of the robot.

Hence, IMU sensor is one of the most important sensor attached on an autonomous robot. An IMU not only gives information about the heading of the robot, but also about the acceleration of the robot which can be fused with other sensors to get an accurate position or velocity estimate.

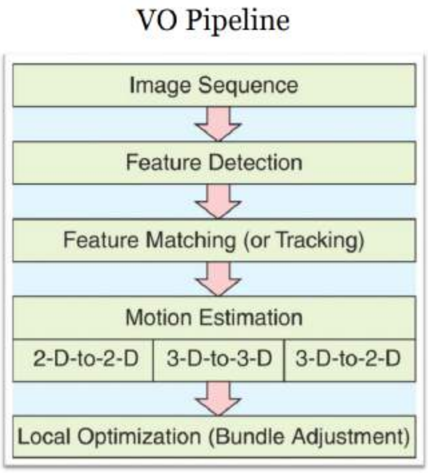


Figure 4.6: Visual Odometry
Source:[25]

4.3.3 Visual Odometry

Odometry is the estimation of the change in the position and orientation over time. The odometry data received from the GPS, IMU and wheel encoders are frequently subjected to mechanical errors, drift, bias, slipping and skidding of the wheels, numerical integration errors, etc. Hence, these sensors are supplemented with visual sensors or laser based sensors. Due to the advancements in the field of computer vision in the past decade, a visual sensor has become an integral part of every robot. Nowadays, a simple camera can be used to perform extremely convoluted tasks like object detection, scene understanding, odometry estimation, etc. and these intricate tasks are performed with very high precision using the computer vision algorithms being developed today. The RTAB-Map(Real-Time Appearance Based Mapping) is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. This package is quite robust to provide stable outputs. In this project, the rgbd_odometry node of the ROS package rtabmap is used which publishes the odom topic used as an input in the estimator.

Visual Odometry makes use of the estimation of the motion of the camera using sequential images i.e ego-motion. The pipeline for visual odomery is depicted in Figure 4.6. A basic algorithm for visual odometry can be explained as mentioned in Algorithm1:

Algorithm 1 Visual Odometry Algorithm

```
1. Capture new frame $I_k$
2. Extract and match features between $I_{k-1}$ and $I_k$
3. Compute essential matrix(computed from feature correspondence using epipolar constraint)
4. Decompose essential matrix into $R_k$ and $t_k$ and form $T_k$
5. Compute relative scale and rescale $t_k$ accordingly
6. Concatenate transformation by computing $C_k = C_{k-1}T_k$
7. Repeat from 1
```

where, R_k is the Rotation matrix, t_k is the Translation matrix and T_k is the Transformation matrix.

The advancements in the field of computer vision has led to quite accurate means of generating precise odometry information. These methods of obaining the odometry information help in providing a source to mitigate the errors caused due to the gps, imu and other sensors prone to noisy readings. Although, visual odometry is accurate in appropriate lighting conditions, it has several challenges as mentioned in [25]:

1. Robustness to lighting conditions

2. Lack of features / non-overlapping images

3. Without loop closure the estimate still drifts

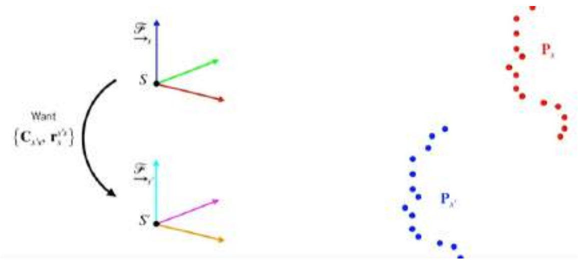

$\underrightarrow{\mathcal{F}}_s$
S
Want
$\{\mathbf{C}_{s's}, \mathbf{r}_s^{s's}\}$
$\underrightarrow{\mathcal{F}}_{s'}$
S'
$\mathbf{P}_s$
$\mathbf{P}_{s'}$

Figure 4.7: Visual Odometry
Source:[21]

Hence, in order to make our estimator more robust to the challenges mentioned above, we have opted to fuse visual odometry with Lidar odometry.

4.3.4 Lidar Odometry

Lidar(Light Detection and Ranging) sensor is one of the most widely used sensor in autonomous robots and self-driving cars. It has been an enabling technology for autonomous robots to visualize its environment in 360°and the Lidar provides very accurate range information. The Lidar uses a simple principle of time-of-flight to generate the point clouds of the environment. A single channel 2D Lidar consists of a laser transmitter. A laser pulse is emitted from the transmitter and on collision with an object, it is reflected back which is then captured by the receiver. The transmitter-receiver system is mounted on a motor rotating in 360°. The Lidar consists of an inbuilt timer circuit and an encoder to accurately estimate the time-of-flight of the laser pulses emitted and reflected. Using the time-distance equation of time-of-flight, a point cloud is generated of the environment. In case of multiple channel Lidars, a 3D point cloud can be generated containing precise information about the surrounding.

As the Lidars use laser beams for its operation, they are robust to lighting conditions. This provides a way for autonomous robots to visualize the environment at night or places with low visibility. The high definition maps generated by Lidars are generally used in most of the self-driving cars nowadays. The Lidars not only provide accurate information about the surroundings, the point clouds can be processed to provide valuable information like odometry data, dynamic obstacles, occupancy grid generation, etc.

For achieving odometry information from Lidar data, the Iterative Closest Point (ICP) algorithm is most widely used. This problem of finding a spatial transformation to align two point clouds is known as the point cloud registration problem. In the ICP algorithm, one point cloud is fixed as a reference, while the other point cloud(source) is transformed to best match the reference. For example, suppose a reference point cloud is returned by the Lidar at time t_1 with respect to a co-ordinate frame S. After some time t_2 and forward movement of the robot, another point cloud if returned by the Lidar with respect to a co-ordinate frame S'. Now, the point set registration problem states that given two point clouds in two different co-ordinate frames, and with the knowledge that they correspond to the same object in the world, how to align them such that the relative motion of the robot can be estimated. As shown in Fig. 4.7, if the correct correspondences are known, the correct relative rotation/translation can be calculated in closed form.

The problem can be formulated as:

1. Given: two corresponding point sets:

$$P_s = \{x_1, ..., x_n\}$$
$$P'_S = \{p_1, ..., p_n\}$$

2. Wanted: translation t and rotation R that minimizes the sum of the squared error:

$$E(R, t) = \frac{1}{N_p} \sum i = 1 N_p \|x_i - Rp_i - t\|^2$$

where x_i and p_i are corresponding points.

The ICP algorithm is explained in Algorithm 2:

Algorithm 2 Iterative Closest Point

```
1. Get an initial guess for the transformation $\{\check{C}_{S'S}, \check{r}^{S'S}\}$
2. Associate each point in $P'_S$ with the nearest point in $P_S$
3. Solve for optimal transformation $\{\hat{C}_{S'S}, \hat{r}^{S'S}\}$
4. Repeat until convergence
```

This is an overview of the point set registration problem which is solved using the ICP algorithm to provide with accurate odometry information. However, the detailed explanation of the algorithm is not in the scope of this thesis. In this project, the icp_odometry node from the ROS package rtabmap is used to find the odometry from the laser scanner using ICP.

To summarize, by fusing the global position estimate from the GPS, orientation estimates from the IMU sensor, visual odometry and Lidar odometry, we obtain robust localization of the robot. We make use of the EKF probabilistic estimator to fuse the data obtained from the various sensors attached on the robot and to accurately localize the robot in the map and the world. The estimator is robust to comprehend partial failures of the sensors and give continuous readings, so that the robot does not drift with time.

4.4 Implementation

An Extended Kalman Filter(EKF) was used for the task of localization and sensor fusion. The detailed information about the sensors used for the localization process has been described in detail in the literature review. Although, the robot was designed to inculcate the IMU and GPS sensors, it was observed that the cheap sensors gave highly erroneous measurements unable to be solved by sensor fusion. Hence, in order to get a relatively better estimate an android phone was attached on the robot to provide the IMU and GPS readings for testing purpose. The entire pipeline has been implemented in the Robot Operating System(ROS). The following sensors are being used for state-estimation and localization:

1. android_sensors_driver - For the purpose of getting accurate gps and imu data, an android mobile phone containing the app android_sensors is mounted on the robot. The app publishes the GPS fixes as sensor_msgs/NavSatFix and the accelerometer/magnetometer/gyroscope data as sensor_msgs/Imu.

2. Orbbec Astra - Astra is a powerful and reliable 3D camera. It is used for obtaining visual odometry and for semantic segmentation. The technical specifications of Astra camera are given in Table 4.1. Two ROS packages - astra_camera and astra_launch, are needed for running the camera on ROS.

3. YDLIDAR X4 Lidar - YDLIDAR X4 Lidar is a 360-degree two-dimensional laser range scanner (Lidar). It is used for obtaining the Lidar odometry. A yd_lidar ROS package is available for operating the Lidar using ROS. The technical specifications of the Lidar are given in Table 4.2.

Table 4.1: Specifications of Astra RGBD Camera

Sr.No.	**Specifications**	**Technical Details**
1.	Range	0.6m – 8m
2.	FOV	60°H x 49.5°V x 73°D
3.	RGB Image Res.	640 x 480 @30fps
4.	Depth Image Res.	640 x 480 @30fps
5.	Size	165mm x 30mm x 40mm

4.4.1 Simulations

For the purpose of localization, it is important to form a motion model and a measurement model for the estimator to carry out the prediction and correction cycle. For initial simulations, the following assumptions are taken:

Table 4.2: Specifications of YDLIDAR X4

Sr.No.	Specifications	Technical Details
1.	Range Frequency	5000 Hz
2.	Scanning Frequency	6-12 Hz
3.	Range	0.12-10 m
4.	Scanning angle	0-360°
5.	Range resolution	< 0.5 mm (Range < 2 m), < 1% of actual distance (Range > 2 m)
6.	Angle resolution	0.48-0.52°
7.	Supply Voltage	4.8-5.2 V

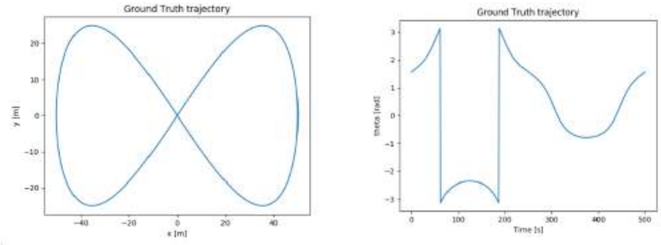


Figure 4.8: Ground Truth

1. The robot to be equipped with a very simple type of Lidar sensor, which returns range and bearing measurements corresponding to individual landmarks in the environment.

2. The global positions of the landmarks are assumed to be known beforehand.

3. Known data association, that is, which measurement belong to which landmark.

The data for the simulation are taken from [26] The robot motion model receives linear and angular velocity odometry readings as inputs, and outputs the state (i.e., the 2D pose $\begin{bmatrix} x & y & \theta \end{bmatrix}^\intercal$) of the vehicle. The motion model is determined as:

$$x_k = x_{k-1} + T \begin{bmatrix} cos\theta_{k-1} & 0 \\ sin\theta_{k-1} & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + W_k \right)$$

The measurement model relates the current pose of the robot to the Lidar range and bearing measurements $y_k^l = \begin{bmatrix} r & \phi \end{bmatrix}^\intercal$:

$$y_k^l = \begin{bmatrix} \sqrt{(x_l - x_k - dcos\theta_k)^2 + (y_l - y_k - dsin\theta_k)^2} \\ atan2(y_l - y_k - dsin\theta_k, x_l - x_k - dcos\theta_k) - \theta_k \end{bmatrix} + n_k^l, n_k^l = N(0, R)$$

x_l and y_l are the ground truth coordinates of the landmark. d is the known distance between robot center and laser rangefinder (Lidar). The ground truth data of the trajectory is shown in Fig. 4.8 and the simulation results are shown in Fig. 4.9

By tuning the values of the variances, it was observed that:

1. One of the most important aspects of designing a filter is determining the input and measurement noise covariance matrices, as well as the initial state and covariance values.

2. If the sensors are noisy, they will affect the performance of the estimator.

3. Hence it is necessary to tune the measurement noise variances in order for the filter to perform well.

4. Improper tuning leads to noisy output as shown in Fig. 4.10

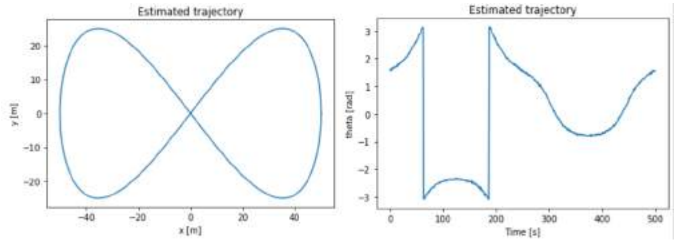


Figure 4.9: Simulation Result

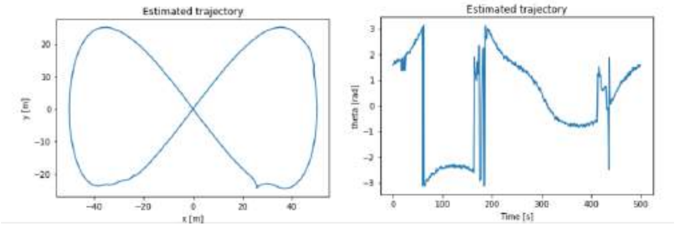


Figure 4.10: Noisy Result

4.4.2 Tests

Localization on hardware platform was tested using a ROS package - robot_pose_ekf. A couple of indoor and outdoor tests were performed with the robot equipped with all the sensors. Based on the tests, certain conclusions are drawn.

Indoor Test

For indoor testing of state-estimation, the robot was moved along circular trajectories of different radius. As mentioned in the literature review, in indoor conditions with proper lighting conditions, visual odometry has been proven to be very accurate. Hence, for the test, the output from visual odometry is assumed as the ground truth. The results of visual odometry received from rtabmap and the state-estimation received from robot_pose_ekf are plotted in Fig. 4.11

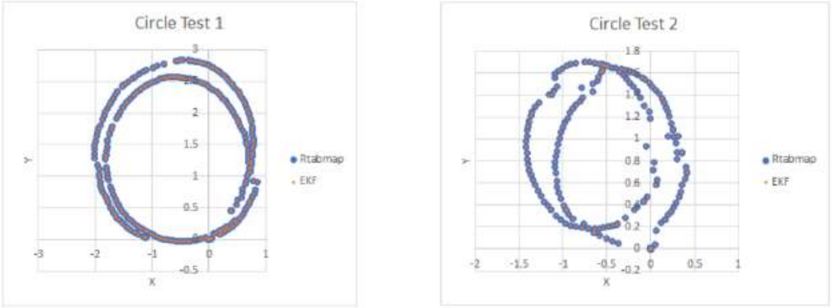


Figure 4.11: Indoor Circle Test

The following things are observed from the indoor tests:

1. As shown in Fig. 4.11, the visual odometry readings are quite accurate as they depict the circle accurately along which the robot is moved in the world.
2. The readings obtained from the EKF estimator overlap mostly with the ground truth, thus indicating accurate results.

Outdoor Test

For outdoor testing, the robot was traversed along a path in the VNIT campus. The sensor data, visual odometry, Lidar odometry received from the ICP algorithm and estimator output were noted for the duration of the test. The results of the test are shown in Fig. 4.12

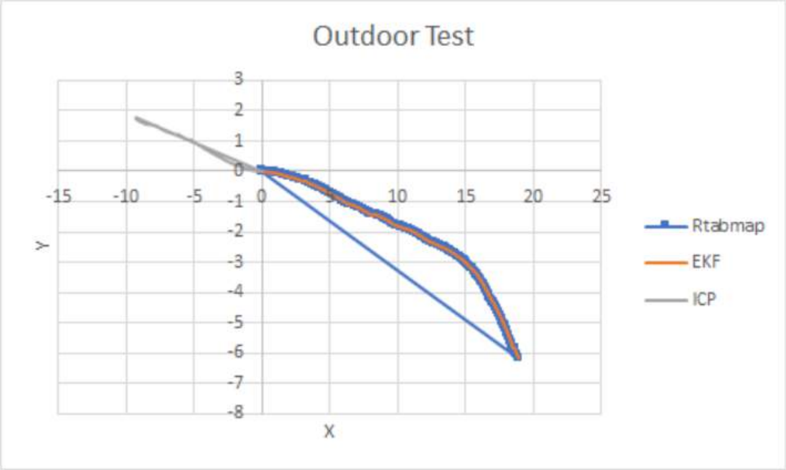


Figure 4.12: Outdoor Test

The following things are observed from the outdoor test:

1. As shown in Fig. 4.12, in outdoor environment, the data from ICP is not very accurate. This error may be attributed to the sensor noise and false transforms between the sensor frame and odometry frame.
2. As the test was carried out in daylight, due to proper lighting conditions, the visual odometry data is accurate, but is seen to lose some data. Thus, failing to provide continuous readings.
3. However, even if the odometry data from visual odometry and Lidar odometry are not continuous and accurate, the EKF estimator is robust enough to provide continuous and precise readings.

4.4.3 Conclusion

As shown in the Indoor and Outdoor Tests, the following can be concluded:

1. In indoor environments, visual odometry gives accurate readings and can be used reliably. However, in the outdoor environments, the visual odometry node tends to fail occasionally, thus providing erroneous readings.
2. During tests conducted at night, visual odometry node fails completely due to insufficient inliers and the estimator must depend on Lidar odometry for accurate information.
3. The robot_pose_ekf package fails to provide readings when there is a discrepancy between the timestamps(10 seconds) of the readings from different sensor nodes. This error occurs when two sensor inputs have timestamps that are not synchronized. As sometimes, the visual odometry fails to give feedback due to certain conditions, the readings are not continuous and hence the estimator fails in such cases.

The importance of accurate state-estimation and localization has been inferred from the various tests carried out on the hardware platform. Localization is the base of any autonomous system. Without precise information about the pose of the robot, it is impossible to function autonomously.

Chapter 5

Planning Algorithms for Outdoor Environment

5.1 Overview

The planning algorithms decide how the robot will achieve its goal of moving from initial starting point to the destination in an efficient way. The "efficient way" may suggest finding the shortest path, finding the quickest path to reach the goal or finding the path which utilizes the least energy. These constraints are considered in the planning algorithm and an output is a path which the robot can traverse satisfying the required conditions.

Planning at its origin was just a search for a sequence of logical operators or actions that transform an initial world state into a desired goal state. However, the field has flourished to deal with complications such as real world uncertainties, multiple bodies, and dynamics. Presently, planning includes many decision-theoretic ideas such as imperfect state information, Markov decision processes, and game-theoretic equilibria. Nowadays major research focus in this field is to optimize the process of generating most efficient paths and developing algorithms that unite planning and control.

In the context of this project, the planning algorithm needs to decide where the robot should head next. For this purpose, the algorithm takes inputs from perception and localisation units and takes decisions based on our developed procedure. There are three main goals the algorithm needs to achieve:

1. Reaching the desired goal state.
2. No collisions with other entities(dynamic or static).
3. Take the best path (shortest or quickest)

5.2 Literature Review

The planning architecture is generally divided into a heirarchical structure as shown in Fig. 5.1

5.2.1 Mission Planner

This is the highest level of optimization problem. In the mission plan, the focus is on map level navigation. The constraints such as finding the shortest path or the quickest path are generally of the major concern here. Other issues such as obstacle avoidance, estimating the time to collision, generating velocity profile, considering the rules of the road etc. are not considered in the mission planner. As mentioned in [27], the mission plan instead focuses on aspects such as speed limits, road length, traffic flow rates, road closures etc. The mission plan is also referred to as Global Plan, as the goal of the mission plan is to find the path the robot will follow in the global map that we generated prior to the operation as mentioned in Chapter 3.

In terms of optimality, the global planner considers the amount of time or distance taken by the chosen path to reach the goal Graph based algorithms are used to find the path. A graph is a data-structure consisting of vertices and edges $G = f(V, E)$ as shown in Fig. 5.2

In context to the problem of path planning, the vertices may be considered as nodes in the global map and and the edges are the road networks joining the vertices. In this sense, a contiguous road network can be discretely represented in the form of a graph. Many graph searching algorithms have been developed such as :

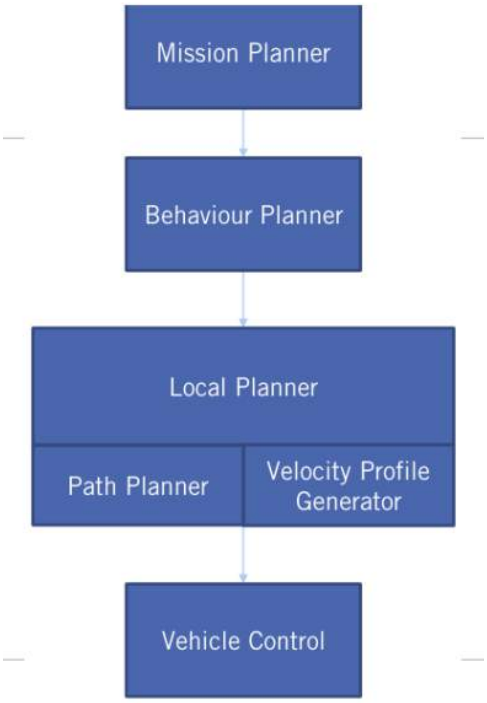


Figure 5.1: Heirarchical Planning Architecture
Source:[27]

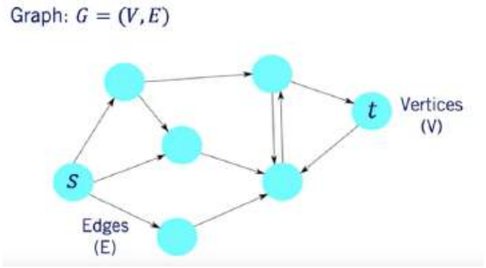


Figure 5.2: Graph
Source:[28]

1. Breadth First Search (BFS)
2. Depth First Search (DFS)
3. Djikstra's Algorithm
4. A-star (A*) Search
5. RRT, etc.

The BFS and DFS algortihms are basic algorithms which can find the shortest path between the start position and the goal position, but are computationally expensive, not optimal and may get stuck in bug-traps. One of the major drawbacks to the DFS and BFS algorithms are that they do not use weighted edges and thus provide non-optimal solutions. However, the A* search algorithms undertakes a heuristic based approach to find the optimal path. For this thesis, the A* algorithm is implemented for finding the optimal global path.

A-Star Algorithm

Before diving in the details of the A* algorithm, certain terminologies are important for understanding the process :

1. agent - An agent is an entity that perceives the environment and acts upon the environment.
2. state - A state is the configuration of the agent in the environment.
3. initial state - It is the state where the agent begins its operation.
4. actions - These are the choices that can be performed in a given state.
5. transition model - It is a description of what the state results from performing an action in a state.
6. state space - It is the set of all possible states reachable from the initial state by any sequence of actions.

Unlike the DFS and BFS algorithms, A* algorithm uses weighted edges which may represent the road lengths between two nodes in the mission planner case. To increase the efficiency of the search algorithm, search heuristic is used. In the context of path planning, a search heuristic is an estimate of the remaining cost to reach the destination vertex from any given vertex in the graph.

$$h(v) = \|t - v\|$$

However, any heuristic used will not be exact as it would then mean knowing the answer to the problem already. But using the search heuristic helps to find the optimal path faster and prevents getting stuck in bug-traps.

The A* algorithm is explained in Algorithm 3 [27].

Algorithm 3 A-Star Algorithm

```
open $\leftarrow$ MinHeap()
closed $\leftarrow$ Set()
predecessors $\leftarrow$ Dict()
open.push(s, 0)
while !open.isEmpty() do
    u, ucost $\leftarrow$ open.pop()
    if isGoal(u) then
        return extractPath(u, predecessors)
    end
    for all $v \in$ u.successors() do
        if $v \in$ closed then
            continue
        end
        uvCost $\leftarrow$ edgeCost(G, u, v)
        if $v \in$ open then
            if uCost + uvCost + h(v) < open[v] then
                open[v] $\leftarrow$ uCost + uvCost + h(v)
                costs[v] $\leftarrow$ uCost + uvCost
                predecessors[v] $\leftarrow$ u
            else
                open.push(v, uCost + uvCost)
                costs[v] $\leftarrow$ uCost + uvCost
                predecessors[v] $\leftarrow$ u
            end
        end
    end
    closed.add(u)
end
```

5.3 Global Planning Implementation

For the purpose of this project, A* algorithm was implemented for finding the global path. As mentioned in Chapter 3, a map of the VNIT campus was constructed using OSM. The osm map existing in the xml file format, consists all the information about the map like the nodes, traversable paths, road lengths, node co-ordinates etc.

OSMnx [29] package was used to find the nearest nodes with respect to the nodes in the map and to read the information available in the xml file of the map. Using the A* algorithm explained previously, the shortest path was determined for the robot as shown in Fig. 5.3.

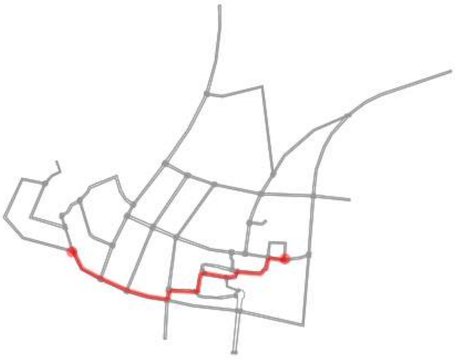


Figure 5.3: Global Plan

Based on the road network available in the map, an optimal shortest path is obtained through the A* algorithm. This path is just the global map the robot will follow. The global plan is given as an input to the behavioral planner and the local planner. These lower level planners will figure out how to generate obstacle free optimized paths considering the behavior of other dynamic agents in the surrounding, avoiding any collision and also providing a smooth motion to the robot.

5.4 Local Planner

The local planner deals with generating an efficient trajectory and velocity profile for the robot to traverse using the data obtained from other modules of the system. The task of a mission planner or a global plan was to generate a global plan to reach the goal position from the start position as mentioned in the previous section. However, it does not account for the various complexities involved in local motion of the robot. For example, the global plan does not account for generating efficient local trajectories for the robot to follow, it also does not include the information of the immediate surrounding in order to avoid collision with dynamic objects, follow lane, predict the time-to-collision and subsequently avoiding the trajectory leading to collision, etc. The local planner ensures smooth operation of the robot on a well defined, efficient trajectory with proper velocity profile required for the robot to move freely in the environment without any aberration. The local planner is one of the most important part of an autonomous robot as it accounts for the safety of the robot as well as the objects in its surrounding like pedestrians, dynamic vehicles, etc. Slightest of mistake in the local planner can lead to hazardous effects.

5.4.1 Basic Terminologies

Some basic terminologies required before proceeding in this section are mentioned below:

1. Configuration space: A set of all possible configurations of a robot in a given world. It is also called C-space denoted by C. The space occupied by a robot is denoted by A.

2. Obstacle space: The space which is already occupied by an obstacle at a given instant of time denoted by O. The C_{obs} is defined as a set of all configurations of the robot whose intersection with obstacle space is not null.

3. Free space: It can be simply defined as C/C_{obs} . That is the space remained by subtracting obstacle space from configuration space.

4. Path: A continuous function of configurations of robot which will lead the robot to desired state from initial state.

5. Query: A pair of initial state (qI) and goal state (qG) of robot provided as input to the planning algorithm by human or some other algorithm.

5.4.2 Formulation of motion planning problem

The basic motion planning problem is conceptually very simple using C-space ideas. The task is to find a path from qI to qG in C_{free}. The entire blob represents $C = C_{free} \cup C_{obs}$. The motion planning problem is shown in Fig. 5.4.

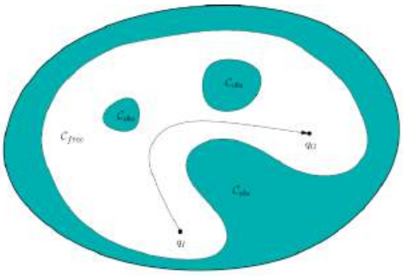


Figure 5.4: Motion Plan

- A world W (2D or 3D)
- A semi-algebraic obstacle region $O \subset W$ in the world.
- A semi-algebraic robot is defined in W. It may be a rigid robot A.
- The configuration space C determined by specifying the set of all possible transformations that can be applied to the robot. From this, C_{obs} and C_{free} are derived.
- A configuration, $qI \in C_{free}$ or the initial configuration.
- A configuration $qG \in C_{free}$ or the goal configuration.

A complete motion planning algorithm must compute a (continuous) path, $\tau : [0, 1] \Rightarrow C_{free}$, such that $\tau(0) = qI$ and $\tau(1) = qG$, or correctly deduces that such a path does not exist. It was shown that this problem is PSPACE-hard, which implies NP-hard.

5.4.3 Literature Review

The currently developed classic methods are variations of four general approaches: Roadmap, Cell Decomposition, Potential fields, and mathematical programming.

Roadmap approach: In this approach, the free C-space, i.e., the set of feasible motions reduced to, or mapped onto a network of 1D lines. This approach is also called the Skeleton, or Highway approach. The search for a solution is limited to the network, and the whole problem becomes a graph-searching problem.

1. Cell Decomposition: In Cell Decomposition (CD) Algorithm, the free C-space is decomposed into a set of simple cells, and then adjacency relationships are computed among the cells. A collision-free path is found by first identifying the two cells containing the initial state and the goal state and then connecting them with a sequence of connected cells.

2. Potential fields: This concept was first introduced by Oussama Khatib.In this method, a robot is treated as a point represented in C-space as a charged particle under the influence of an artificial potential field U. The potential function can be defined over free space as the sum of an Attractive potential attracting the robot toward the goal configuration, and a Repulsive potential repelling the robot away from the obstacles to avoid collisions.

3. The Mathematical programming approach: This represents the requirement of obstacle avoidance and shortest path with a set of inequalities on the configuration parameters and an objective function. Planning problem is formulated then as a mathematical optimization problem that finds a smooth curve between the start and goal configurations minimizing a certain cost function.

5.4.4 Planning architecture

The general architecture for planning includes the representation of a pipeline of processing perception inputs to generate optimal plans of motion. It contains several unit operations which are represented in the chart. This gives an overall idea of what's happening inside the processor.

Overall Structure

First let's see the entire architecture of the robot. See Fig. 5.5

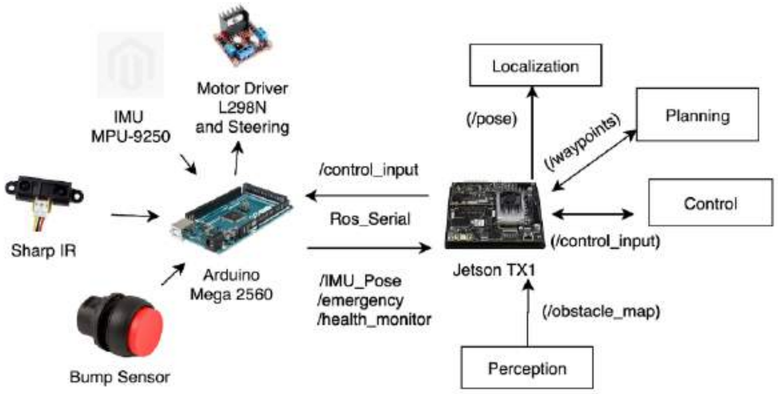


Figure 5.5: Cyber physical architecture of the robot. Showing all the main processes and the names of ROS topics are mentioned above the arrows to get the intuition of the data transfer. The complete working of each unit is explained throughout the thesis.

Now Let's look at planning architecture. See Fig. 5.6.

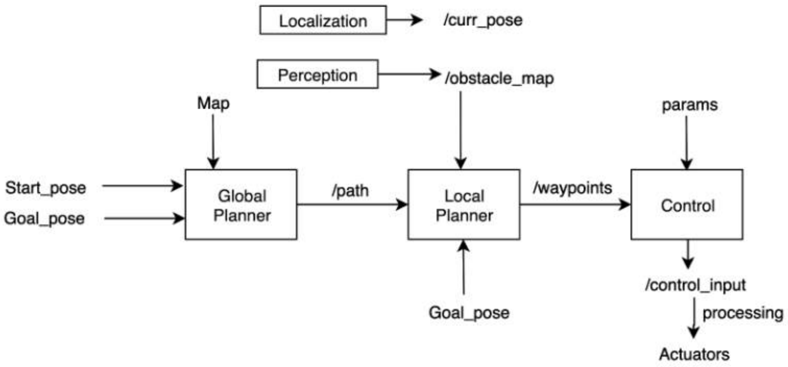


Figure 5.6: Planning architecture of the robot. The algorithmic details of the above blocks are illustrated in this section. The ROS topic names are shown above arrows to get intuition of the system data transfer.

Input processing

The inputs to the local planner are :

1. Global planner: A-star based searching algorithm which uses a VNIT map to find a path from A to B in terms of nodes which are basically way-points which are a feet away from each other including all the turn nodes. The output of the search is the set of way-points which is published on a ROS topic.

2. Localisation: A modified EKF is used to fuse the data from VO, IMU and LIDAR based odom to get a final pose of the robot in the real world. The pose is then transformed into different frames as per requirement. The odometry is published on the ROS topic.

3. Obstacle data: A segmentation map generated by a neural net is processed to get the road contour or a boundary of a traversable area on the road. This contour is merged with the cropped lidar contour to get a 360 degree view of the traversable area. The merged contour is published on the ROS topic.

5.4.5 Collision checking and avoidance

Overview

Collision checking

It is one of the fundamental operations in robotic motion planning. This operation can be divided into static and dynamic collision checking. Static checking refers to checking amounts to testing a single configuration for testing spatial overlaps. Dynamic checking needs to answer if all the configurations on a path in C-space are collision free. There are three major methods for dynamic checking as, Feature tracking , bounding volume and swept volume methods.

A common approach is to sample paths at any fixed, pre-specified resolution and statically test each sampled configuration. This approach is not guaranteed to detect collision whenever one occurs, and trying to increase its accuracy by refining the sampling along the entire path results in slow checking. Researchers have found optimal ways to do this by varying sampling resolution as per the requirements in real time. But in our case we have used a fixed resolution to avoid excess computations and we have followed a popular circle based checking method.

Collision avoidance

The purpose of obstacle avoidance algorithms is to avoid collisions with the obstacles.These algorithms deal with moving the robot based on the feedback of the sensor information. An obstacle avoidance algorithm is modifying the trajectory of the robot in real time so that the robot can prevent collisions with obstacles detected on the path.

We can divide the collision avoidance problem into “global” and “local”. The global techniques involve path planning methods relying on availability of a topological map defining the robots work-space and obstacle space. The entire path from start to goal can be planned, but this method is not suitable for fast collision avoidance due to its complexity. On the other hand the local greedy approaches of using pure obstacle avoidance methods are unable to generate an optimal solution. Another problem is that when using a local approach the robots often get into a local minimum. Because of these shortcomings, a reactive local approach representing obstacle avoidance cannot be considered as a complete solution for robot navigation. Due to this reason, we need to combine both obstacle avoidance and path planning techniques to develop a hybrid system overcoming the cons of each of the methods. In our architecture we have also used such a combination to avoid obstacles while planning paths recursively.

Circle-Point based checking

Imagine there is a circle $C1$ with center $(x1, y1)$ and radius D'. Let’s assume a point $P1$. Imagine there is a line running between center and $P1$. The distance from the center point to $P1$ when $P1$ is on edge of circle is D'

So: Any greater distance than D' and the circle won’t collide with the point. Any less distance than D' and then collision will happen. The Fig. 5.7 explains it clearly. We have sampled the obstacle contours into a set of discrete points and we have sampled circles on the global path to check intersection with obstacle contours.

Safety considerations

Considering the dimensions of the robot, we have calculated a radius of circle for collision checking which involved an additional cushion along the surroundings of the robot and apart from this the robot is equipped with three sharpIR sensors which are calibrated for a safe distance measurements if the robot goes too

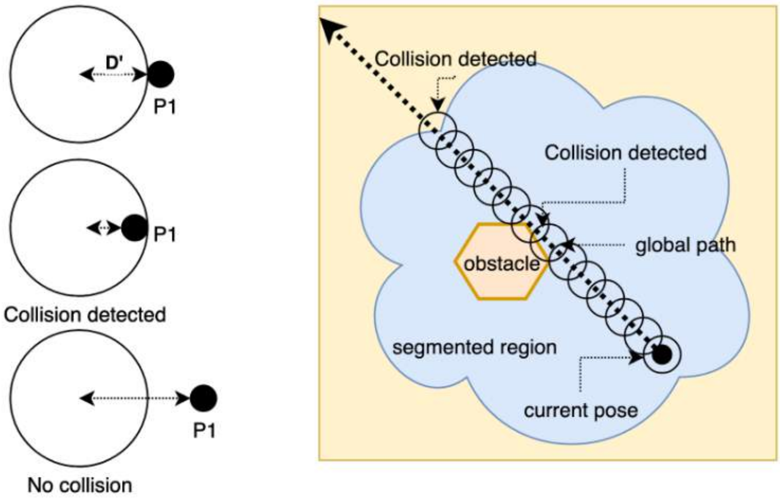


Figure 5.7: Collision checking method

close to some obstacle then the emergency stop behaviour is invoked by SharpIR sensors, stopping robot immediately and replanning the route.

The last layer of safety is the bump sensor. If apart from all the algorithmic and hardware provisions, the robot still bumps into something then the bump sensor immediately shuts down the whole system to prevent stalling of actuators and mechanical damage.

5.4.6 Local planning algorithm

Specific use case

Considering the environment, motion planning can be either static or dynamic. We say a static environment when the location of all the obstacles is known priori. Environment is dynamic when we have partial information about obstacles prior to robot motion. Initially the path planning in a dynamic environment is done. When the robot follows its path and identifies new obstacles it updates its local map, and changes the trajectory of the path if necessary. In our case the environment is dynamic. Also, our robot is supposed to be travelling along roadsides. On roadsides there might also be some parked vehicles or other static obstacles which needs to be considered while designing planning algorithms. Other than static obstacles there can be humans and other moving vehicles. So we need to be quick and accurate in planning our path. It means that the algo should be computationally efficient, robust and should have predefined emergency behaviours. This defines our use case, now in the next subsection we have mentioned the actual working of the algorithm.

Designed algorithm

Planner receives the input of obstacle way-points from segmentation contour and Lidar contour. Way-points are received as input from global planner which is A* based planner searching nodes in a map of VNIT. If collision is detected then way-points are slided along a line perpendicular to the slope of the line joining the consecutive way-points or path and the direction of sliding is away from obstacles. The distance for sliding is fixed and decided by using dimensions of bot and max turning radius. Sliding is operated like a chain. Every second slide of a way-point is followed by sliding of its immediate next and previous way-points.The collision checker function and avoiding function operate in recursion. The planner returns a new set of way-points to avoid the obstacle. This algorithm will surely return a path if it exists otherwise it reports that a path is not possible. Step-wise algorithm is mentioned in the Fig. 5.8. The Fig. 5.9 shows an example of how the planner works if sliding to the left slide is not an option.

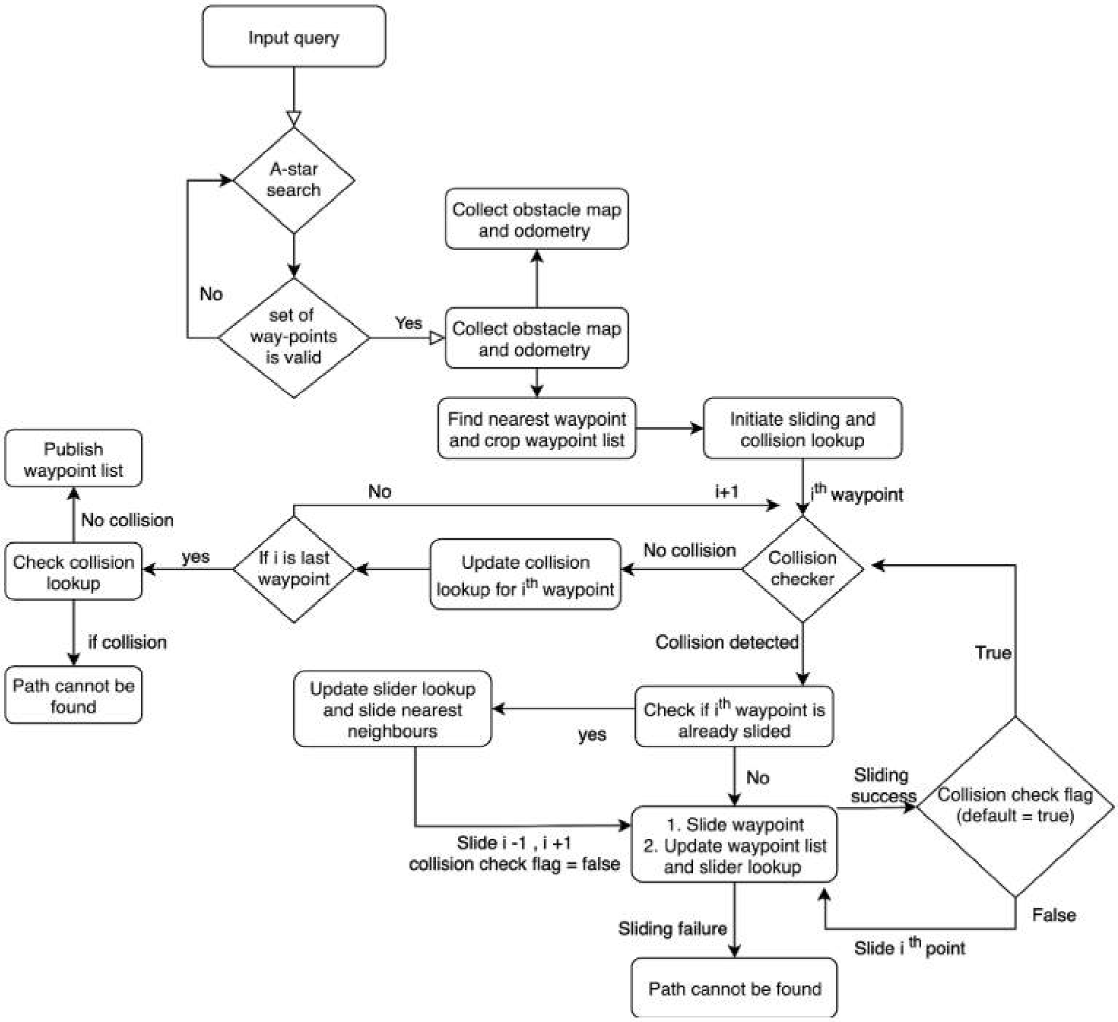

Input query
A-star search
No
set of way-points is valid
Yes
Collect obstacle map and odometry
Collect obstacle map and odometry
Find nearest waypoint and crop waypoint list
Initiate sliding and collision lookup
i th waypoint
Publish waypoint list
No collision
No
i+1
Check collision lookup
yes
If i is last waypoint
Update collision lookup for i th waypoint
No collision
Collision checker
if collision
Path cannot be found
Collision detected
True
Update slider lookup and slide nearest neighbours
yes
Check if i th waypoint is already slided
No
Sliding success
Collision check flag (default = true)
Slide i -1 , i +1 collision check flag = false
1. Slide waypoint
2. Update waypoint list and slider lookup
Sliding failure
False
Slide i th point
Path cannot be found

Figure 5.8: Planning algorithm block diagram

5.5 Conclusion and Future Work

This thesis explains the hierarchical planning structure and the implementation of A* algorithm to find the shortest path from starting position to the goal in the map along with the local planner responsible for creating smooth and efficient trajectory for the robot with collision avoidance. Further research could be carried out for the optimization of problems related to occupancy grid generation, finding the time to collision, generating energy efficient paths etc.

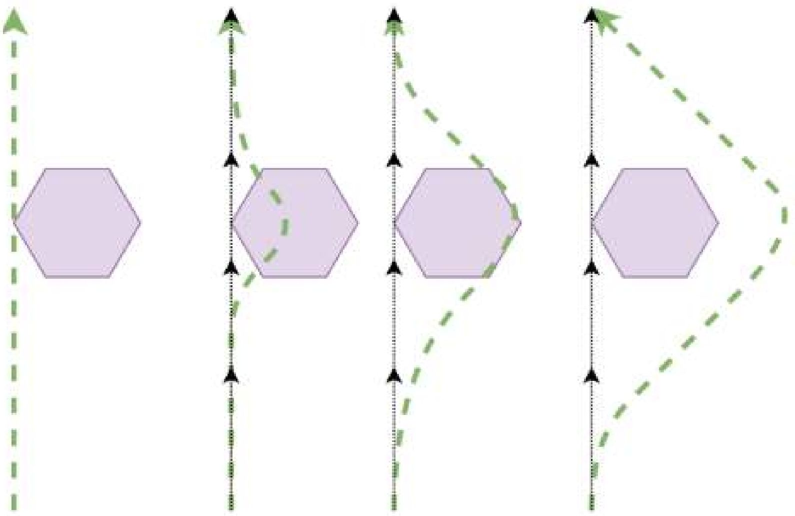


Figure 5.9: Planning algorithm working explanation. The hexagon indicates an obstacle. The figure is showing a step-wise algorithm when going from the left side of an obstacle is not allowed. The dotted line indicates the path to be followed and the line with the arrow indicates the global path.

Chapter 6

Semantic Segmentation for Road and Obstacle Detection

6.1 Overview

Image Segmentation is the partitioning of an input image into multiple segments (sets of pixels). The goal is to represent the image as something simpler to analyze and more meaningful. As per [30], some of the subtasks involved in image segmentation are :

- Semantic Segmentation [31]: Each pixel is classified into one of the predefined set of classes such that pixels belonging to the same class belongs to a unique semantic entity in the image. Note that the semantics (logic) in question depends not only on the data but also the problem being addressed.
- Saliency Detection [32]: Focus on the most important object in a scene.
- Instance Segmentation [33]: Segments multiple instances of the same object in a scene.
- Segmentation in the temporal space [34]: Object tracking requires segmentation in the spatial domain as well as over time (temporal domain).
- Oversegmentation [35] [36]: Images are divided into extremely small regions to ensure boundary adherence, at the cost of creating a lot of spurious edges. Region merging techniques are used to perform image segmentation.
- Color or texture segmentation: Also found to be useful for certain applications.

This work focuses on semantic segmentation because that is sufficient for our purpose of identifying the road and obstacles in traffic/road scenes.

Note: Some prior knowledge of Convolutional Neural Networks [37] is required for proper understanding of this chapter.

6.2 Literature Review

Most modern techniques make use of Supervised Deep Learning, majorly involving Convolutional Neural Networks [37], due to their success in Image Classification tasks. These convolutional networks consist of sequential application of convolutional filters, pooling layers and non-linear activation functions. An example is shown in Figure 6.1. This particular architecture is used for image classification. It is a mapping (function) between the image and the output class.

However, semantic segmentation is different, and thus modifications are required to the architecture. Particularly, Fully Convolutional Networks (FCN) [38] are used for segmentation. These do not involve fully connected (dense) layers. Further, output size reduces in conventional CNNs, so it cannot be used for pixel-level classification. In FCN, an interpolation layer is used which upsamples the intermediate outputs (called feature maps) to the size of the input image. It is interesting to note that this interpolation is done using bilinear interpolation and is not learnable. An example is shown in Figure 6.2.

Fully connected networks (conventional deep CNNs like Figure 6.1) used flattening of 2D feature maps to perform classification. However, flattening results in loss of spatial relation between pixels in the feature map. FCNs overcome this by avoiding the use of flattening and using upsampling followed by pixel-level

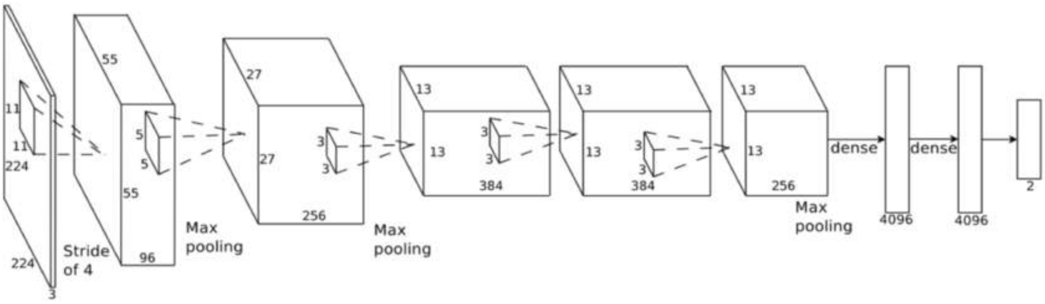


Figure 6.1: AlexNet: Deep Convolutional Neural Network architecture from [37]

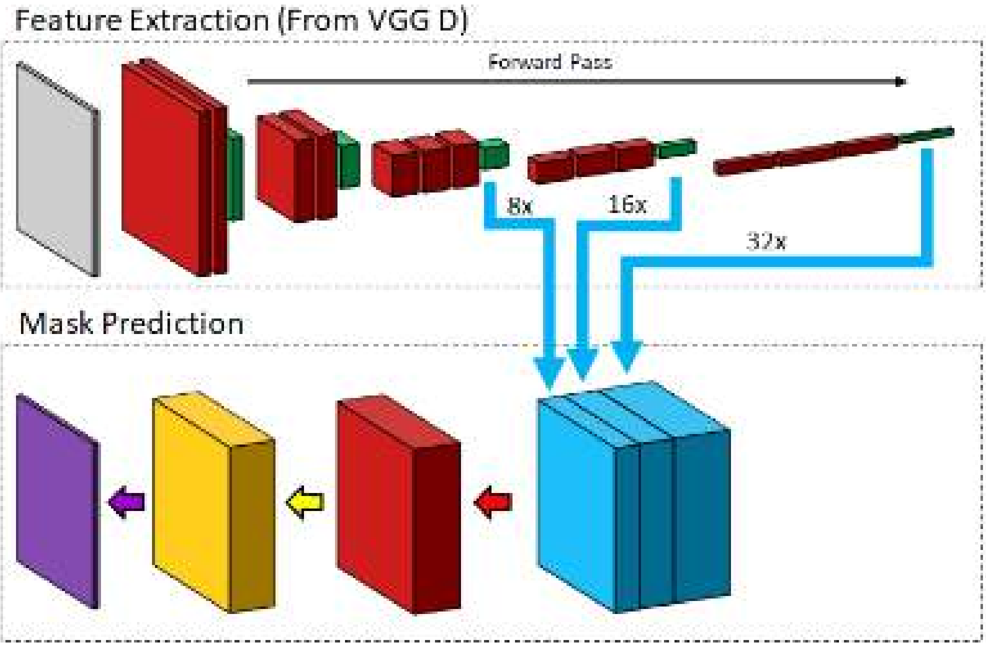


Figure 6.2: Fully Convolutional Network architecture from [39]

classification. However, the issue with FCNs is the loss of sharpness due to intermediate subsampling. This issue has been addressed using skip connections and other methods.

Another approach is the use of Convolutional Autoencoders [40]. These are traditionally used for Representation Learning. An autoencoder has 2 parts, encoder and decoder. Encoder encodes the raw input to a lower dimensional representation, while the decoder attempts to reconstruct the input from the encoded representation. The decoder's generative nature can be modified to achieve segmentation tasks.

The major benefit of these approaches is generation of sharper boundaries without much complication. Unlike classification approaches, the decoder's generative nature can learn to generate delicate boundaries using the extracted features. Another benefit is that it does not restrict input size. The commonly used technique for decoding is transposed convolutions (learnable) or unpooling layers. However, a possible issue with such approaches is over-abstraction of images during the encoding process, i.e. the network starts memorizing the training images instead of learning filters that are useful for compression and reconstruction.

Another technique is the use of skip connections, first introduced in [42]. Linear skip connections are often used to improve gradient flow for large number of layers. Skip connections are also useful to combine different levels of abstraction from different layers to produce sharp segmentation output. An example is shown in Figure 6.3.

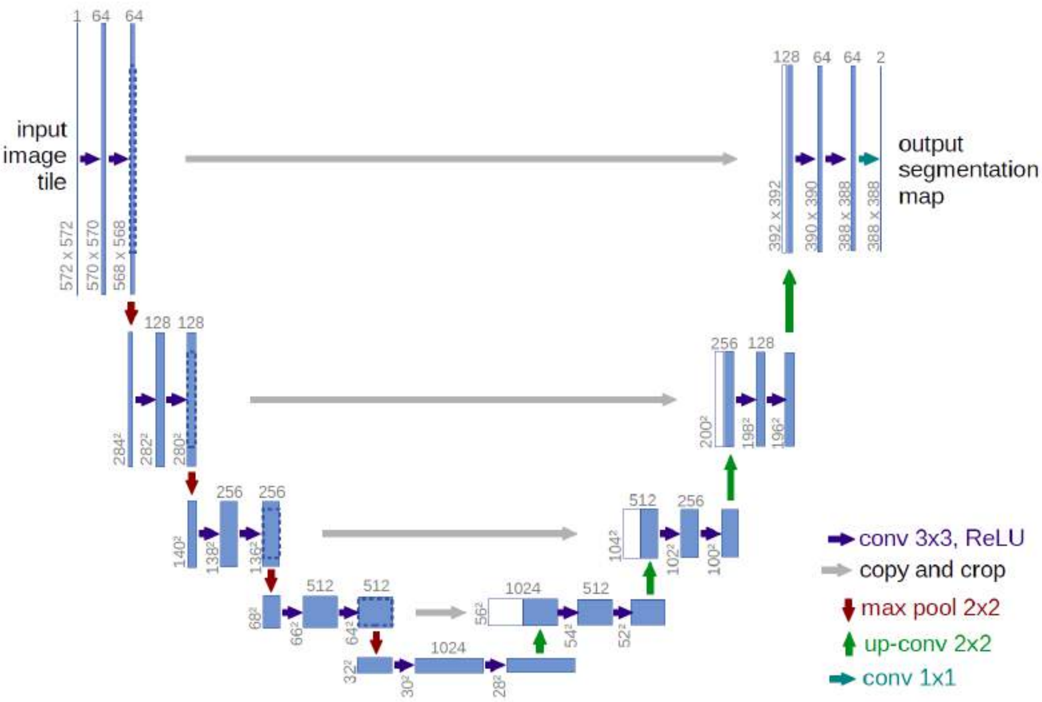


Figure 6.3: U-Net architecture utilizing skip connections from [41]

6.3 Preliminary Experiments

For our early tests, we used U-Net architecture [41] with the Cambridge-driving Labeled Video Database (CamVid) [43]. The training is done using Adam Optimizer [44] with learning rate 1e-4 minimizing pixel-wise Cross Entropy Loss. The implementation was done on the PyTorch [45] based fastai [46] framework. The architecture is shown in Figure 6.3. Some example images and labels are shown in Figure 6.4. The preliminary results are shown in Figure 6.5.

As seen from the results, the output segmentation is not sharp enough and sometimes cannot segment obstacles properly. The major issues which led to this are as follows:

- Since the CamVid dataset has only around 700 images, the network is not able to generalize. This also causes the network to inaccurately segment certain obstacles like a person on a motorcycle as in Figure 6.5.
- Further, since those images are from urban areas of foreign cities, the network is unable to sharply segment images from our institute's campus, which features less structured roads and other features.
- There are too many classes which makes the task more difficult for the network. This is because the underlying optimization problem is more difficult to solve.

Another issue, unrelated to the performance, is the high computational complexity of the model. The computational complexity hinders the real-time use of the model, since computational power on the robot will be limited. Thus, for our final implementation, we use ENet [47], which is computationally more efficient while preserving performance.

6.4 Final Implementation

E-Net architecture (in Figure 6.6 and Table 6.1) is implemented in PyTorch[45]. The code is available at `https://github.com/IvLabs/autonomous-delivery-robot/tree/master/sem_seg_pytorch`. The dataset used was Cityscapes [48], which provides around 25k images having coarse annotations for road scenes. Some examples are shown in Figure 6.7. In this final implementation, only the road is segmented from the image, while assuming anything other than the road to be an obstacle. Further, Dice Loss

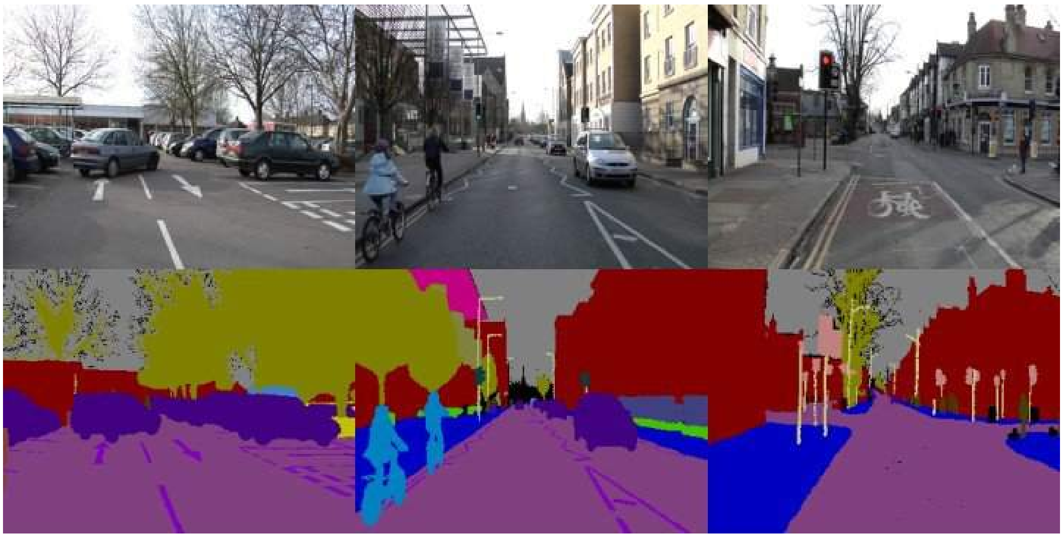


Figure 6.4: Sample images and pixel-level labels from CamVid [43]

[49] is used in combination with the Cross Entropy Loss to improve the segmentation output. Dice Loss is given by:

$$DL(p, \hat{p}) = 1 - \frac{2p\hat{p} + 1}{p + \hat{p} + 1} \tag{6.4.1}$$

where $p \in 0, 1$ and $0 \leq \hat{p} \leq 1$, p is the label and $\hat{p}$ is the prediction from the network.

Some of the features of this loss function are:

- The fraction $\frac{2p\hat{p}+1}{p+\hat{p}+1}$ signifies the amount of overlap between the label and the predicted segmentation. It is called the Dice Coefficient.
- Maximizing the Dice Coefficient will give maximum overlap. However, we are already minimizing the Cross Entropy Loss. Thus, we minimize $1 -$ Dice Coefficient, which is the same as maximizing the Dice Coefficient.
- Further, this loss function performs particularly well in case of unbalanced segment sizes i.e. very small obstacle segments may affect the optimization and it may get stuck in a local minima. Dice Loss helps to avoid this problem.
- However, the problem with Dice Loss is that the gradients are complicated compared to Cross Entropy gradients and have the tendency to explode (too small values of p and $\hat{p}$ may cause very large gradients), which makes the training unstable.

Thus, taking note of the advantages and disadvantages of Dice Loss, it is beneficial to use a combination of Dice Loss and Cross Entropy Loss. This combination is minimized simultaneously in our implementation using Adam optimizer with a learning rate of 1e-4.

The design choices of the ENet architecture are detailed in [47]. They are briefly summarized below:

- The use of **P**arametrized **Re**ctified **L**inear **U**nits (PReLU) [50] gives an additional learnable parameter to the non-linear nature of the network. This improves the performance over using standard ReLU non-linearity.
- In other architectures, decoder is generally an exact mirror of the encoder (w.r.t. architecture). Here, decoder is much smaller than the encoder. The idea is that encoder will process and filter the input, while the decoder simply upsamples and fine-tunes the encoder output.
- The use of factorizing filters or asymmetric convolutions [51] i.e. decomposition of $n \times n$ convolution into an $n \times 1$ convolution followed by a $1 \times n$ convolution reduces the computational and memory costs. For example, an asymmetric convolution with $n = 5$ is similar to a 3×3 convolution in terms of computational cost and memory requirements.

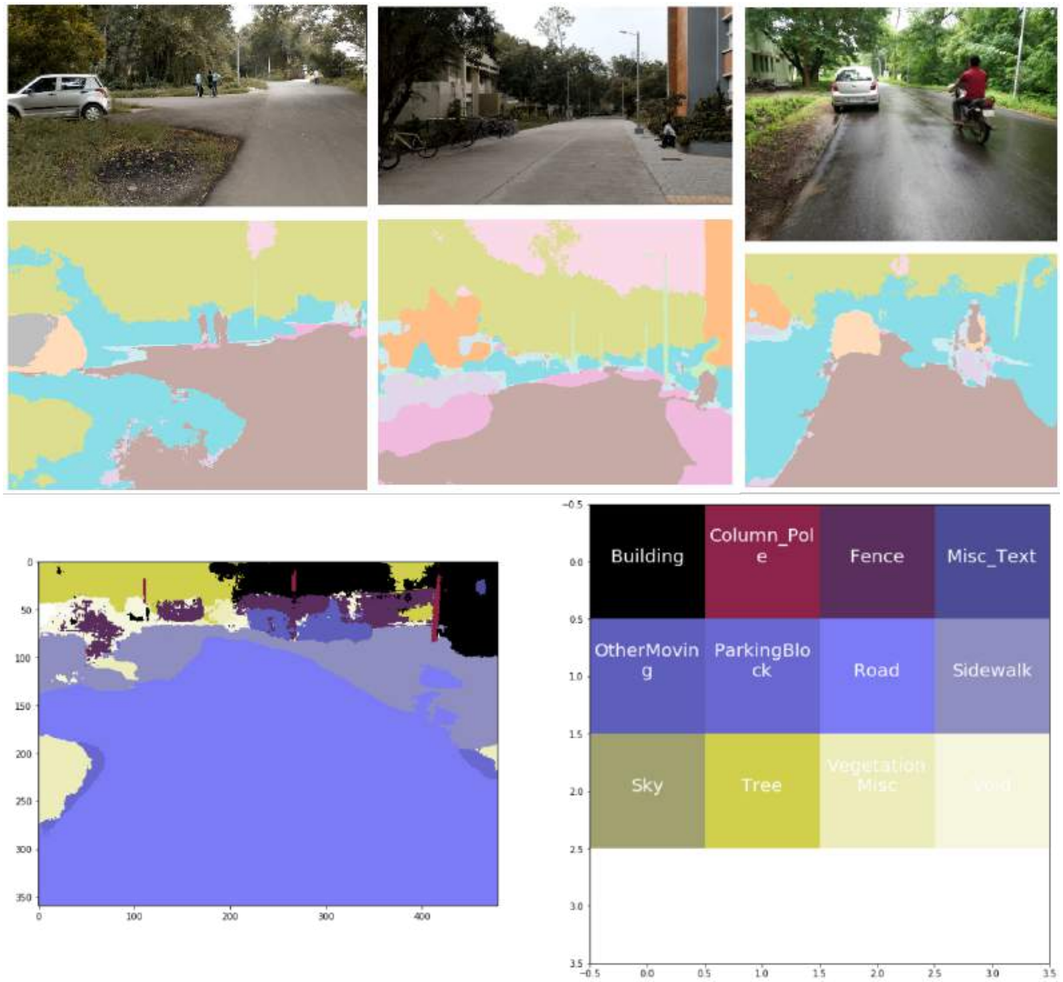


Figure 6.5: Results of U-Net trained with CamVid dataset on VNIT campus images with color map indicating relation between class and color.

- The use of dilated convolutions [52] gives the network a wide receptive field. Thus, dilated convolutions are used instead of normal convolutions.

The result of these design choices is the reduction in memory and computational requirements while ensuring good performance. The results on some images from the VNIT campus are shown in Figure 6.8. Note that these images are captured using a handheld smartphone camera.

6.4.1 Hardware Implementation

Since this semantic segmentation model is to be deployed on the robot, it is essential to use a small, portable computer. Further, it should have enough computational power to process the images in real-time while also running the other algorithms (global and local path planning, localization, etc.) in parallel. Several such low-cost single-board computers are available in the market like the Raspberry Pi, ODroid, etc. However, they all lack a graphic processing unit (GPU) which severely limits the performance of the model while additionally introducing latency in the entire system.

Thus, the choice of the onboard computer is limited to devices with a dedicated GPU. Nvidia has 2 products namely, Jetson TX1 and TX2 which have a dedicated GPU while maintaining a portable form factor. Thus, the Nvidia Jetson TX1 (since it's the cheaper option) is used as the onboard computer. The important hardware specifications of the Nvidia Jetson are as follows:

- Quad-Core ARM Cortex A57 CPU
- 256-Core Nvidia Maxwell GPU

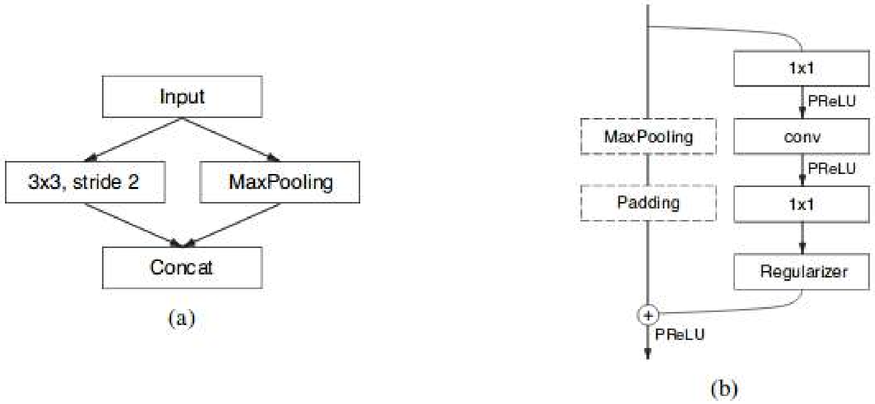


Figure 6.6: (a) ENet initial block. MaxPooling is performed using non-overlapping 2 × 2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. (b) ENet bottleneck module. conv is either a regular, dilated or deconvolution with 3 × 3 filters, or a 5 × 5 convolution decomposed into 2 asymmetric ones. Source: [47]

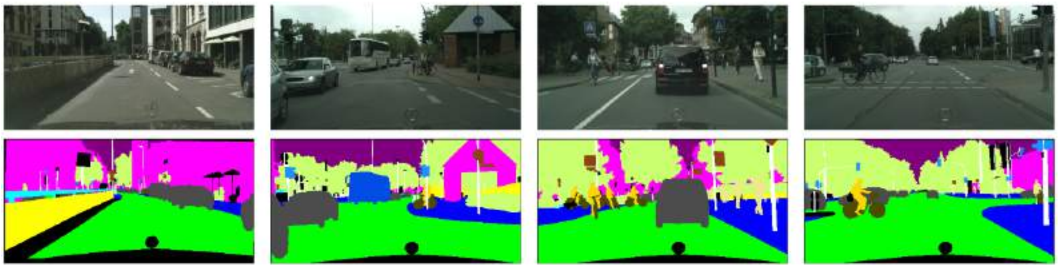


Figure 6.7: Sample images and pixel-level labels from Cityscapes [48]

- 4 GB LPDDR4 memory (RAM and GPU VRAM is shared)
- 16 GB eMMC storage (with SD Card expansion slot)
- < 10 W power requirement

This small yet powerful device has its disadvantages. Most libraries, frameworks and software support for Intel x86 CPUs is very good. However, that is not true for ARM processors, mostly because they are not as widely used. Thus, the framework choices became limited, and the model was implemented in PyTorch because support for the TX1 was available readily in older versions of PyTorch.

Table 6.1: ENet Architecture [47]. Output sizes are given for an example input of 512×512

Name	Type	Output Size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
4×bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

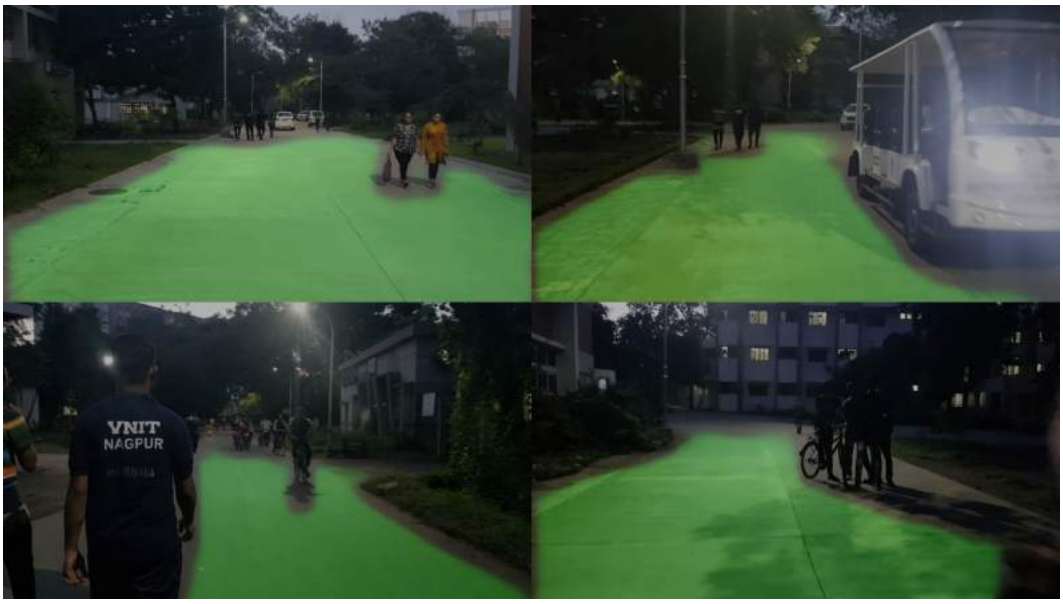


Figure 6.8: Results of ENet trained with Cityscapes dataset on VNIT campus images with green overlay representing the detected road pixels.

Chapter 7

360° Vision using Catadioptric Camera Setup

7.1 Overview

7.1.1 Need of 360° Vision in Autonomous Vehicle.

Building reliable vision capabilities for self-driving cars has been a major development hurdle. By combining a variety of sensors, however, developers have been able to create a detection system that can recognize a vehicle's environment even better than human eyesight.

From photos to videos, cameras are the most accurate way to create a visual representation of the world, especially when it comes to autonomous vehicles.

Autonomous vehicles have to rely on cameras placed on every side - front, rear, left and right to stitch together a 360-degree view of their environment. Some have a wide field of view as much as 120 degrees and a shorter range. Others focus on a more narrow view to provide long-range visuals.

7.1.2 Catadioptric System

Catadioptric systems are those which make use of both lenses and mirrors for image formation. This contrast's with catoptric systems which use only mirrors and dioptric systems which use only lenses.

Panoramic images can be created from conventional catadioptric cameras. Ideal omnidirectional catadioptric cameras can provide images covering the whole view space.

7.2 Literature Review

7.2.1 Cameras with a Single Lens

"Fish-eye" lenses provide a wide angle of view and can directly be used for panoramic imaging. A panoramic imaging system using a fisheye lens was described by Hall et al. in [53]. A different example of an imaging system using a wide-angle lens was presented in [54] where the panoramic camera was used to find targets in the scene. Fleck [55] and Base et al. [56] studied imaging models of fisheye lenses suitable for panoramic imaging.

7.2.2 Cameras with Single Mirror

In 1970, Charles [57] designed a mirror system for a single-lens reflex camera. Various approaches on how to get panoramic images using different types of mirrors were described by Hamit [58]. Gregus [59] proposed a special lens to get a cylindrical projection directly without any image transformation. Chahl and Srinivasaan [60] designed a convex mirror to optimize the quality of imaging. they derived the family of surfaces which preserve the linear relationship between the angle of incidence of light onto the surface and the angle of reflection into the conventional cameras.

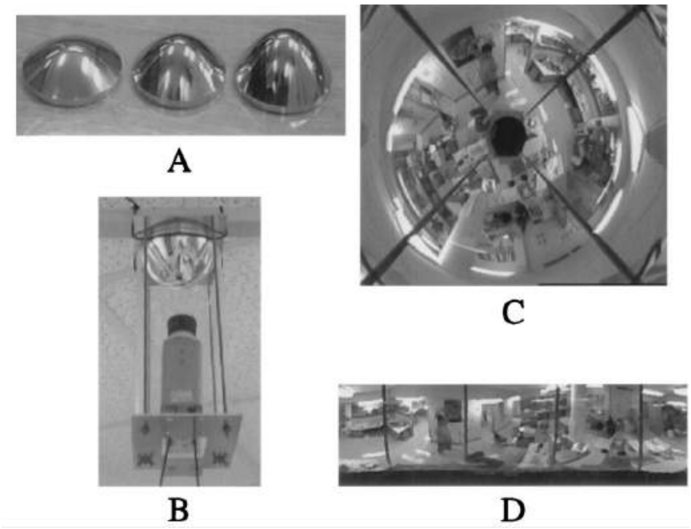


Figure 7.1: Experimental implementation of the global imaging system. A, Three surfaces produced by turning aluminum on a CNC lathe. The mirrored finish was achieved by polishing with a metal polish of various grades. B, Assembled device enclosed in a glass tube for rigidity and to protect against dust. Internal reflection did not appear to be a major problem. C, Image produced by the device. The field of view is approximately 240°. D, Image that resulted from unwarping the image, C. by Chahl et. al. in [60]

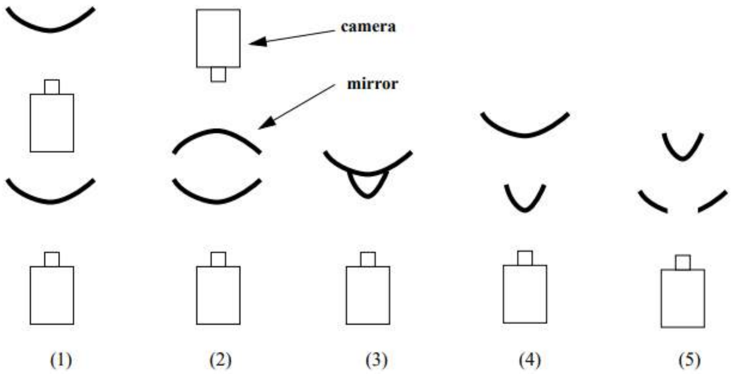


Figure 7.2: Side view of the five Catadioptric configurations examined by Mark Ollis et. al. [61] in (1) and (2) using two cameras. (3), (4) and (5) useing a single camera.

7.3 Proposed Omnivision (Catadioptric) Camera Setup

7.3.1 System Model

The proposed setup is given in Figure 7.4. The Pinhole camera model shown in the figure is used to focus and capture the reflected incoming light flux by an axially symmetric curved mirror at the bottom. The optical focus of the pinhole camera is defined as the primary (1^0) focus of the system and the converging point of the extended incoming light flux is estimated as secondary (2^0) focus of the system.

Figure 7.5 shows the performance of the proposed setup under the curved mirror typologies of three types of curvature conditions and the convention used for (1) Incoming Light Flux, (2) Extended Light Flux and (3) Converging Light Flux.

7.3.2 General Cylindrical Projection of the Image

Considering the reflected flux makes and image on the plane shown in Figure 7.4 with respect to secondary focus, the curved mirror image to rectangular cylindrical projected image transformation equations is given by:

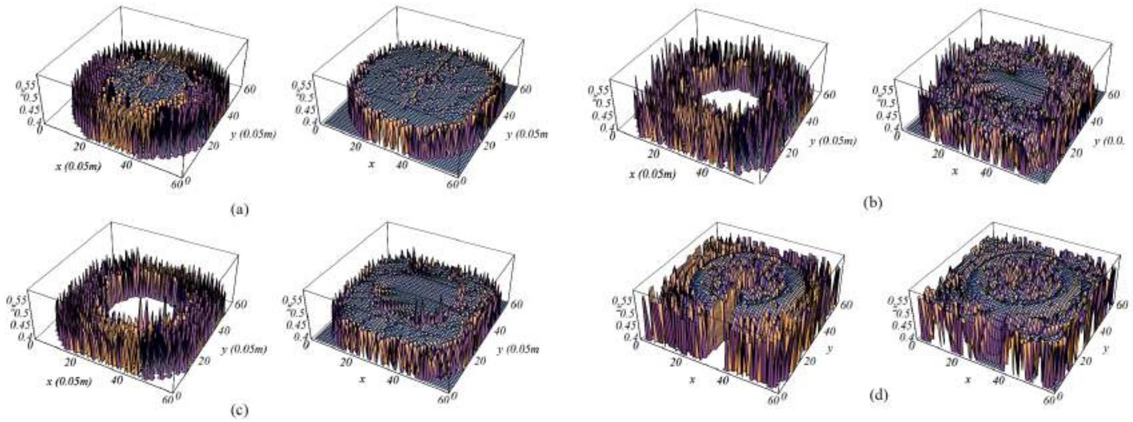


Figure 7.3: 3D view of reconstructed terrain using configuration 1. 3D points are integrated into a regularly spaced grid of height values.Raw terrain map & Interpolated terrain map, using (a) configuration 1, (b) configuration 1, (c) configuration 1,(d) configuration 4 & 5, by Mark Ollis et. al. [61]

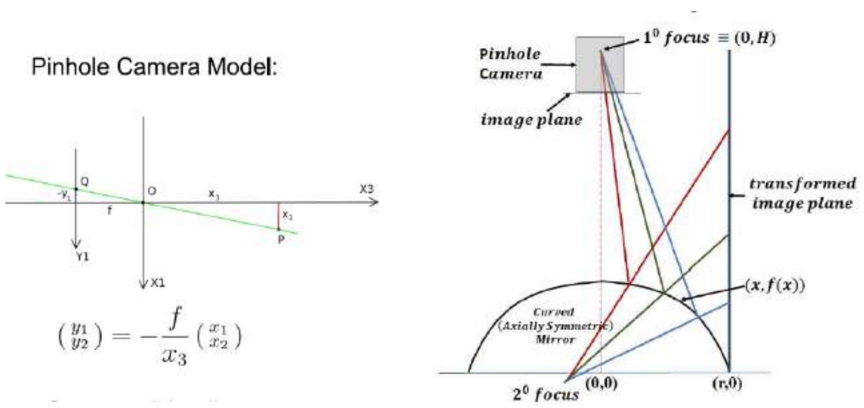


Figure 7.4: Pinhole Camera Model and Proposed Omnivision (Catadioptric) Camera Setup

$$f'(x) = -\tan\theta,\ \ \alpha - \theta = \Phi + \theta$$

$$i.e. \phi = \alpha - 2\theta, \tan\alpha = \frac{H - f(x)}{x}$$

$$\tan\phi = \frac{A + B}{1 - A.B},\ \ A = \frac{H - f(x)}{x},\ \ B = 2.\frac{f'(x)}{1 - f'(x)^2}$$

$$h = f(x) + (r - x).\frac{(1 - f'(x)^2).(H - f(x)) + 2.x.f'(x)}{x.(1 - f'(x)^2) - 2.f'(x).(H - f(x))} \tag{7.3.1}$$

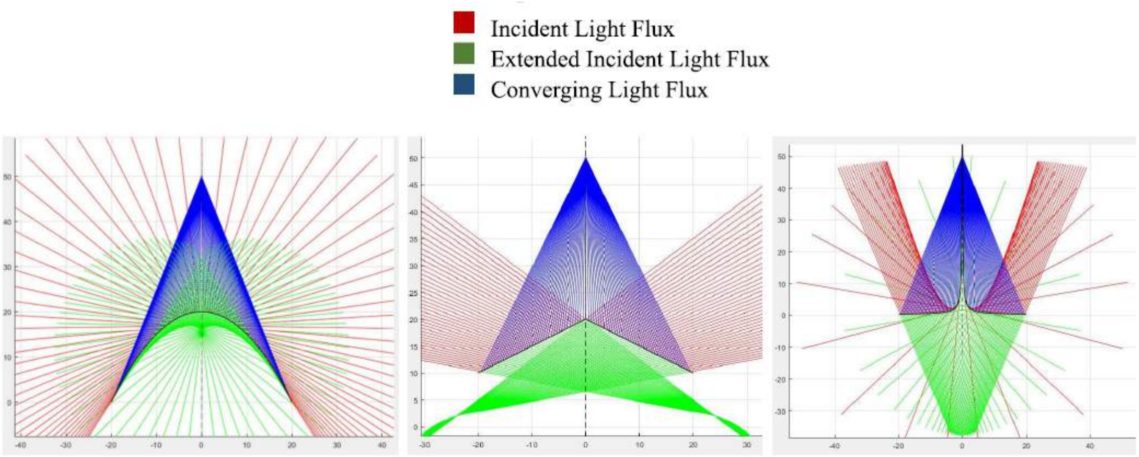
7.3.3 Suitable Shape for the curved Mirror

Based on the equation 7.3.1, the curved shape of the mirror is constrained due to certain conditions given by (a) The incoming flux lines should not intersect outside the mirror in order to have unique mapping on the rectangular frame, and (b) Extended flux lines must converge at the point inside the mirror to obtain the secondary focus of the system.

Hence by considering the above conditions;

According to condition (a),

for, $\alpha_1 > \alpha_2,\ \ \phi_1 > \phi_2$, i.e. $x_2 > x_1$



Convex up

Zero Curvature

Concave up

Figure 7.5: Light flux in different curved mirror typologies depending upon surface curvature.

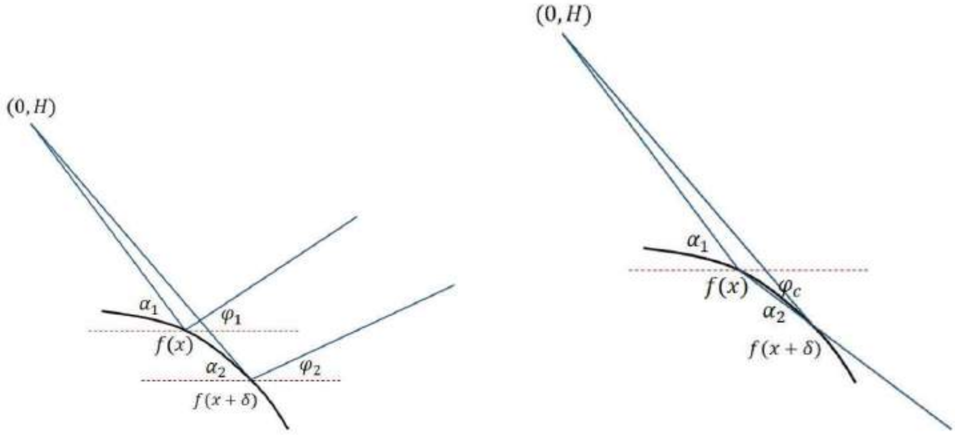


Figure 7.6: Suitable mirror shape derivation.

As shown in the Fig. 7.6, let ϕ_c be the critical angle of reflection,

$$\phi > \phi_c \therefore \tan\phi > \tan\phi_c, \quad \phi_c, \phi \in (-\pi/2, \pi/2)$$

$$\tan\phi_c = \frac{f(x) - f(x-\delta)}{\delta}$$

$$\frac{f(x) - f(x-\delta)}{\delta} = \tan\phi_{c1} \leq \tan\phi_1$$

$$\frac{f(x+\delta) + f(x)}{\delta} = \tan\phi_{c2} \leq \tan\phi_2$$

$$\text{but, } \tan\phi_1 > \tan\phi_2$$

$$\therefore \frac{f(x) - f(x-\delta)}{\delta} > \frac{f(x+\delta) + f(x)}{\delta}$$

for small δ.

$$f(x) > \frac{f(x+\delta) + f(x-\delta)}{2} f'(x_1) > f'(x_2) \quad \forall\, x_2 > x_1$$

if the function f is monotonically decreasing,

$$f'(x_2) \leq f'(x_1) \quad \forall\, x_2 > x_1 \quad f'(x_1)\,, f'(x_2) < 0$$

Hence, the curve function of the required shape of the mirror must be convex up.

7.4 Analysis Over Curved Surface Topologies

From the mathematical analysis discussed in the previous section, we developed the mathematical model for the system and compared the behavior of different geometrical shapes (sphere, cone, and paraboloid) under the set of parameters consisting of the height of the focus from the mirror center. The resultant plots obtained in MATLAB are given in the figure 7.7.

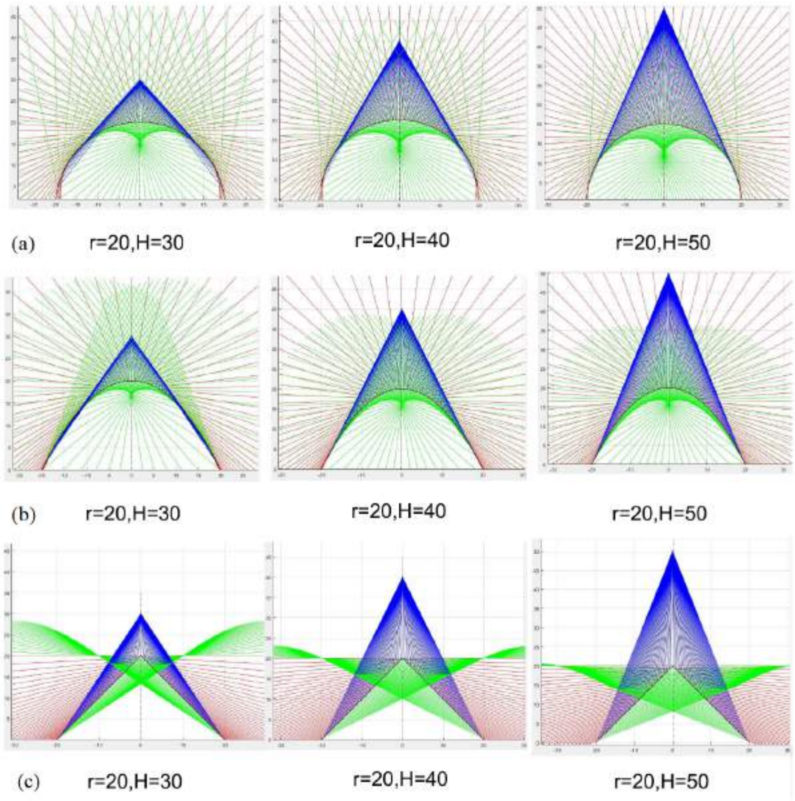


Figure 7.7: Light flux plots of (a) Spherical Mirror, (b) Paraboloid Mirror, (c) Conical Mirror with parameters mentioned.

As shown in the figure 7.7 the location secondary focus formed by Spherical and Paraboloid mirror is independent of the height H. But in the conical mirror secondary focus tends to go inside the mirror as height decreases. Also, the field of view in the conical mirror is ranged in the lower part parallel to the horizontal whereas in spherical and paraboloid it is located perpendicular to the horizontal reference.

Similarly, the conical mirrors with different slopes of the cross-section and height 'H' are studied.

7.5 Simulation Results

Based on an analysis of the different curved mirror surfaces and conical mirrors, we developed CAD design of the conical mirror with suitable dimensions and other objects in surrounding using SolidWorks software. The figure shows the experimental setup and the rendered images taken from it.

Further, Cylindrical Projection is taken using equations mentioned in Subsection 7.3.2. The final results are shown in Figure 7.10.

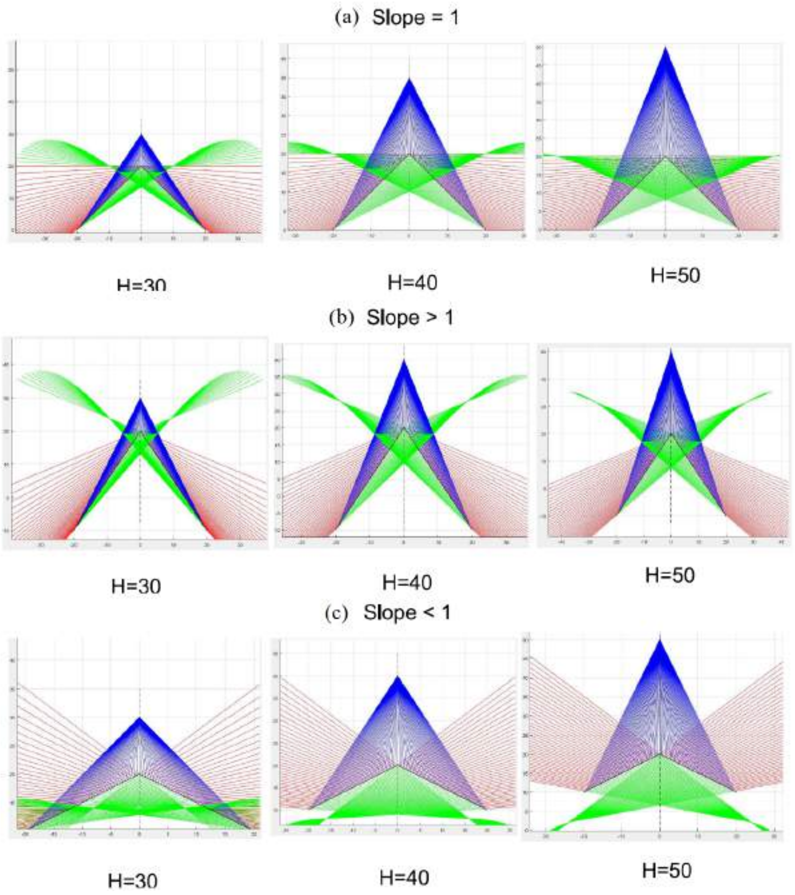


Figure 7.8: Light flux plots of Conical mirrors with cross section slope (a) = 1, (b) > 1, (c) <1.

7.6 Conclusion

The advantages of panoramic imaging with catadioptric setup are:

1. Increased area coverage with single (or two) cameras.

2. Simultaneous imaging of multiple targets.

3. Instantaneous full-horizon detection.

4. easier integration of various applications required for Autonomous Vehicles.

The proposed catadioptric system is economic over the present technologies providing the same information. The simplistic and static approach solves the problem more optimistically and economically. This idea has been discontinued for practical hardware implementation in this thesis since the manufacture of the conical reflector is too difficult for us to get a reliable reflector without distortions on the surface. Whereas, it will be very expensive if outsourced (or must be manufactured in bulk).

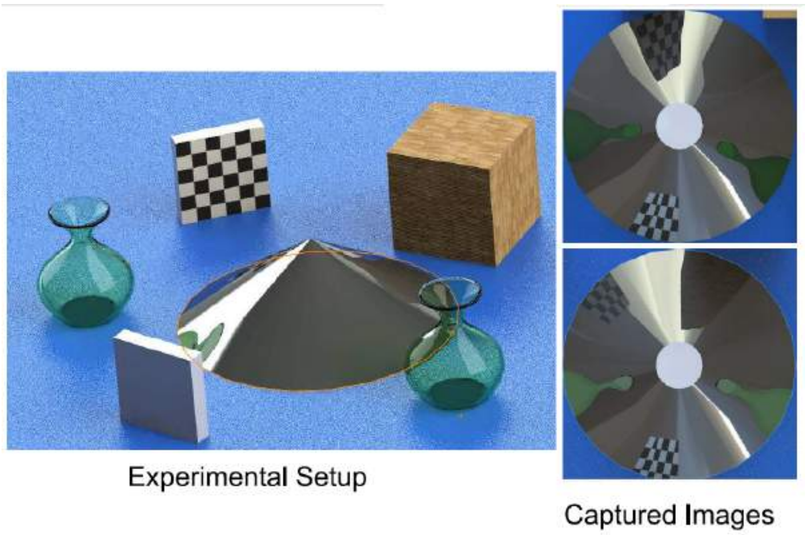


Figure 7.9: Experimental Setup and Captured Images from Conical Mirror.

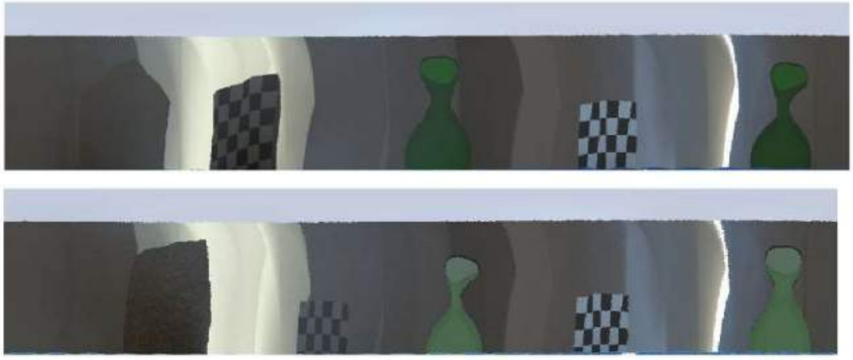


Figure 7.10: Cylindrical Projection of images given in Figure 7.9

Chapter 8

Camera Lidar System Calibration

8.1 Overview

8.1.1 Camera Calibration

Camera Calibration also known as Camera Resectioning is the process of estimating internal parameters (also known as intrinsics) and external parameters (also known as extrinsics) of a camera. Internal parameter comprises of focal length, principal point, skew (if present), aspect ratio and lens distortions. External parameters comprises of 6 degrees of freedom associated with the camera in any arbitary coordinate system. In order to derive 3D (metric) information from a camera, calibration is an essential step. It allows photogrammetric measurements from images, distortion correction, 3D reconstruction, etc.

8.1.2 LIDAR Calibration

LIDAR stands for LIght Detection and Ranging. It is a device which provides accurate range measurements. It could be used in tasks such as localization, odometry, etc. In this project we are using a 2D LIDAR named YDLIDAR. It generates a planar point cloud of the scene. In order to fuse the information of LIDAR point cloud with the camera images, it is essential to register the devices together. By register, we mean to compute the transformation matrix between the Camera and a LIDAR.

8.2 Pin Hole Camera Model

In order to mathematically model digital cameras used in this project, it is important to gain understanding of a simple pinhole camera. A pinhole camera, unlike other cameras, do not have any lenses or mirrors, but just an aperture for light rays to enter and a film to capture the light. It is shown in Fig. 8.1.

The image obtained from pin-hole camera is inverted and is projection of 3D world object onto a camera film. The size of the projected image is dependent on the distance of object from pin-hole as well as the distance of pin-hole from the camera film (also referred as focal length).

The relationship between the 3D object and its 2D image for a pin-hole camera from Fig. 8.1 can be expressed as :

$$\frac{y}{-f} = \frac{Y}{Z} \text{ or } y = -f * \frac{Y}{Z} \tag{8.2.1}$$

Similarly,

$$\frac{x}{-f} = \frac{X}{Z} \text{ or } x = -f * \frac{X}{Z} \tag{8.2.2}$$

Negative sign in the equations indicates that the images are inverted.

8.2.1 Extending pin-hole camera model to digital cameras

The pin-hole model can be extended to cameras by making few changes : -

- The origin of image lies on the top left corner, so we must shift the projected coordinates with (c_x, c_y), which represents the projection of camera center on the image plane plane

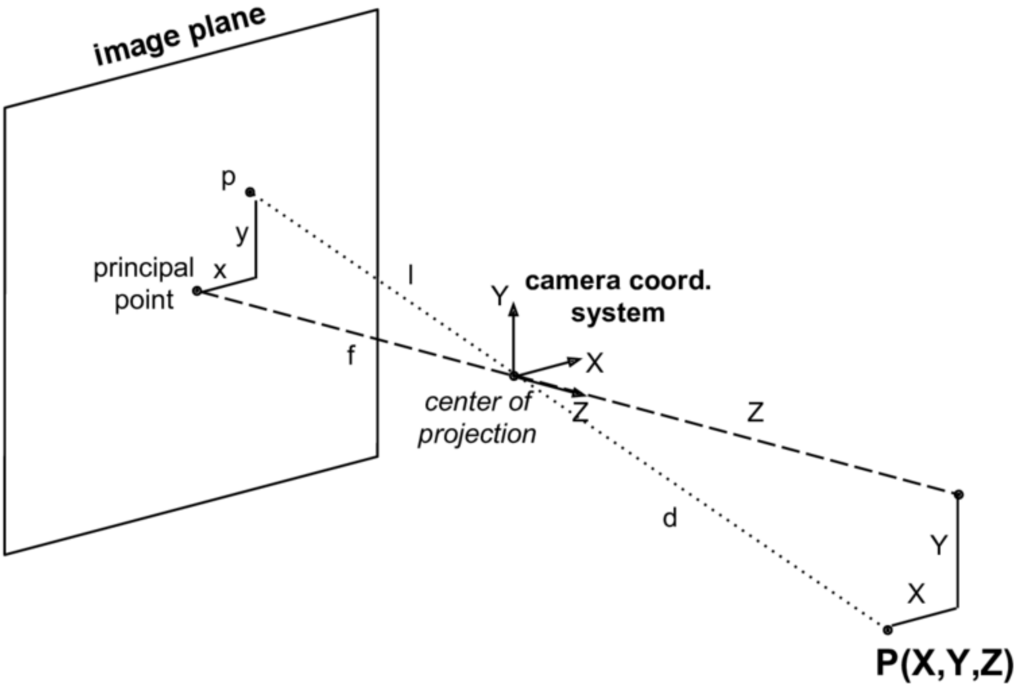


Figure 8.1: Pinhole camera

- In order to keep the equations 2.2.1 and 2.2.2 positive, we have assumed a virtual screen in front of the camera, thus eliminating the negative signs.
- The image recorded is not continuous but discretized by the image sensor into numerous pixels. Each pixel should ideally be square. However, it deviates from a square, in terms of aspect ratio and skew.
- After accounting all the parameters, the new transformation from camera coordinate system to image in homogeneous coordinates can be written as follows :

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{8.2.3}$$

  In the above equation 2.2.3, λ is equal to Z, which is the distance of the world point from the principal point.

- The X, Y, Z in the equation 2.2.3, is expressed in camera coordinate system, in order to generalize it to any arbitrary Cartesian system we may introduce a Transformation Matrix T, from arbitrary coordinate system to camera coordinate system, thereby modifying equation 2.2.3 as :

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{8.2.4}$$

  The first three columns of transformation matrix constitutes a rotation matrix and the last column is the translation vector.

- The pinhole model deviates as we move farther away from the image center, this effect can be accounted using polynomial model of distortion. We can also incorporate tangential distortions if necessary.

8.3 Zhang's Method of Camera Calibration

In order to calibrate our camera, we have used Zhang's method of camera calibration. This method requires us to capture images of an asymmetric checker board in different orientations and positions. The output of this method is a set of camera parameters:

- The intrinsic parameters like:
  - principal point (c_x, c_y)
  - focal length in pixels (f_x, f_y)
  - skew s (if present)
  - distortion coefficients
- The extrinsic parameters like:
  - The orientations of checkerboard w.r.t camera-coordinate system in different images
  - The positions of checkerboard w.r.t camera-coordinate system in different images

This information will be used in further processing.

8.4 Camera LIDAR Extrinsic Calibration

In order to extrinsically calibrate LIDAR, we have implemented the algorithm presented in ??. This method requires us to capture images as well as the point clouds of the scene in which checker-board is placed. The orientation of checkerboard is changed after every capture. Once the capture is completed, we manually segment the points from every point-cloud of LIDAR that belongs to checker-board. Then we calibrate the camera from the images which were captured using Zhang's method described in Section 2.3. For every pose of checkerboard we compute the normal N in camera cordinate system using the equation:

$$N = -R_3(R_3^T.t) \tag{8.4.1}$$

Here, R_3 represents the third column of rotation matrix from equation 2.2.4 and t represents the translation vector. The equation of the checkerboard plane can then be written as

$$N.x = \|N\|^2 \tag{8.4.2}$$

Here, x denotes the points lying on the plane, in camera system. Now lets assume that the transformation between the points in camera and lidar is represented as : -

$$P_l = \phi P_c + \delta \text{ or } P_c = \phi^{-1}(P_l - \delta) \tag{8.4.3}$$

Here, P_l represents the point in Lidar system and P_c in Camera System. ϕ is the rotation matrix and δ is translation vector between them. Substituting equation 2.4.3 in 2.4.2, we get

$$N.\phi^{-1}(P_l - \delta) = \|N\|^2 \tag{8.4.4}$$

Equation 2.4.4 can further be simplified as :

$$N.HP_l = \|N\|^2 \tag{8.4.5}$$

Where,

$$H = \phi^{-1} \begin{bmatrix} 1 & 0 & -\delta_x \\ 1 & 0 & -\delta_y \\ 1 & 0 & -\delta_z \end{bmatrix} \tag{8.4.6}$$

We can solve for H with every pose with multiple LIDAR points linearly using least squares.

After determining $H = [H_1, H_2, H_3]$, ϕ and δ can be computed as

$$\phi = [H_1, -H_1 \times H_2, H_2]^T \tag{8.4.7}$$

$$\delta = -\phi.H_3 \tag{8.4.8}$$

The solution obtained can then be refined by non-linear optimization using Levenberg-Marquardt algorithm.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this thesis, the construction of the prototype and the tasks of perception, mapping, localization and planning for the Autonomous Delivery Robot are demonstrated. We have implemented and tested the various modules required in an autonomous robot operating in open environments. The issues faced in design and operation of the robot in outdoor environments have been discussed in detail in this thesis. We aimed to create a completely autonomous robotic system, robust enough to operate smoothly on roads with proper localization and planning. This would solve the problem of robots being manually controlled by a human operator and help in carrying out tasks for humans to make their lives easier.

Although the purpose of the robot in this thesis is aligned towards a delivery robot, the pipeline explained in the thesis can be used for any robots operating autonomously in an outdoor environment. For example, the presented system can also play a significant role in medical assistance applications such as in the current COVID-19 pandemic situation, autonomous food and medicine delivery as well as sample collection in critical areas.

9.2 Future Work

This thesis demonstrates an end-to-end pipeline for an autonomous robot operating in an outdoor environment. The current prototype is centered on the software portion and theory implementation rather than the actual functionality of the delivery vehicle. Therefore, in the future, the hardware will be more focused on delivery vehicle aspects such as effective human-machine interface as well as the dashboard, distant monitoring system and sound fail-safe management, long run time, and the larger payload capacity. However, highly complex situations were avoided while testing. The outdoor environment is highly unpredictable due to dynamic obstacles such as pedestrians, animals, vehicles, etc. Further research could be carried out to develop highly optimized and robust planning algorithms to overcome the mentioned issues in an efficient manner. The robot could be made more robust to operate in different weather conditions like rain, snow, etc. and on uneven road terrains. The future work can include developing efficient algorithms or choose appropriate sensors to tackle the issues faced in localization of the robot in outdoor environment as explained in Chapter 4. Instead of using a single channel Lidar for localization, a multi-channel Lidar could be used to provide a well defined 3D point cloud. The map can be refined using additional layers as mentioned in the literature review to construct a precise 3D map of the environment containing very useful information. The autonomous robot and the underlying algorithms explained in this thesis can be used as a base platform to conduct further research in each independent module.

Acknowledgement

First and foremost, we express our sincere gratitude to our guide, Dr. Ashwin Dhabale, Assistant Professor, Department of Electrical Engineering, for his continuous support and guidance throughout the course of the project.

We would also like to thank Dr. Shital Chiddarwar of Mechanical Engineering Department and the Professor-in-charge of IvLabs - the Robotics Club of VNIT for providing space to work in IvLabs and for GPU access for training our models.

Bibliography

[1] M. Vasic and A. Billard. “Safety issues in human-robot interactions”. In: *2013 IEEE International Conference on Robotics and Automation.* 2013, pp. 197–204.

[2] M. Edwards. “Robots in industry: An overview”. In: *Applied Ergonomics* 15.1 (1984), pp. 45–53. ISSN: 0003-6870. DOI: https://doi.org/10.1016/S0003-6870(84)90121-2. URL: http://www.sciencedirect.com/science/article/pii/S0003687084901212.

[3] C. D. Kidd and C. Breazeal. “Robots at home: Understanding long-term human-robot interaction”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2008, pp. 3230–3235.

[4] Navid Panchi et al. “Deep Learning-Based Stair Segmentation and Behavioral Cloning for Autonomous Stair Climbing”. In: *International Journal of Semantic Computing* 13.04 (2019), pp. 497–512. DOI: 10.1142/S1793351X1940021X. eprint: https://doi.org/10.1142/S1793351X1940021X. URL: https://doi.org/10.1142/S1793351X1940021X.

[5] U. Patil et al. “Deep Learning Based Stair Detection and Statistical Image Filtering for Autonomous Stair Climbing”. In: *2019 Third IEEE International Conference on Robotic Computing (IRC).* 2019, pp. 159–166.

[6] H. Durrant-Whyte and T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110.

[7] Kumar Chellapilla. *Rethinking Maps for Self-Driving.* URL: https://medium.com/lyftlevel5/https-medium-com-lyftlevel5-rethinking-maps-for-self-driving-a147c24758d6.

[8] Kris Efland and Holger Rapp. *Semantic Maps for Autonomous Vehicles.* URL: https://medium.com/lyftlevel5/semantic-maps-for-autonomous-vehicles-470830ee28b6.

[9] OpenStreetMap contributors. *OpenStreetMap.* https://www.openstreetmap.org/copyright.

[10] I. R. Nourbakhsh R. Siegwart and D. Scaramuzza. “Introduction to Autonomous Mobile Robots”. In: Second Edition, MIT Press. URL: https://mitpress.mit.edu/books/introduction-autonomous-mobile-robots-second-edition.

[11] L. Liao et al. “Bayesian Filtering for Location Estimation”. In: *IEEE Pervasive Computing* 2.03 (July 2003), pp. 24–33. ISSN: 1558-2590. DOI: 10.1109/MPRV.2003.1228524.

[12] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. URL: https://doi.org/10.1115/1.3662552.

[13] L. Chen, H. Hu, and K. McDonald-Maier. “EKF Based Mobile Robot Localization”. In: *2012 Third International Conference on Emerging Security Technologies.* Sept. 2012, pp. 149–154. DOI: 10.1109/EST.2012.19.

[14] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey. “Circumventing dynamic modeling: evaluation of the error-state Kalman filter applied to mobile robot localization”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C).* Vol. 2. May 1999, 1656–1663 vol.2. DOI: 10.1109/ROBOT.1999.772597.

[15] A. Giannitrapani et al. “Comparison of EKF and UKF for Spacecraft Localization via Angle Measurements”. In: *IEEE Transactions on Aerospace and Electronic Systems* 47.1 (Jan. 2011), pp. 75–84. ISSN: 2371-9877. DOI: 10.1109/TAES.2011.5705660.

[16] Greg Welch and Gary Bishop. "Welch, Bishop , An Introduction to the Kalman Filter 2 1 The Discrete Kalman Filter In 1960". In: 1994.

[17] Lawrence Schwartz and Edwin Stear. "A Computational Comparison of Several Nonlinear Filters". In: *Automatic Control, IEEE Transactions on* 13 (Mar. 1968), pp. 83–86. DOI: 10.1109/TAC.1968.1098800.

[18] "Nonlinear Kalman filtering". In: *Optimal State Estimation.* John Wiley and Sons, Ltd, 2006. Chap. 13, pp. 393–431. ISBN: 9780470045343. DOI: 10.1002/0470045345.ch13. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470045345.ch13. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470045345.ch13.

[19] Paul Dawkins. *Linear Approximations.* URL: https://tutorial.math.lamar.edu/Classes/CalcI/LinearApproximations.aspx.

[20] Jonathan Kelly. *State Estimation and Localization for Self-Driving Cars.* Coursera Lecture Video. 2019. URL: https://www.coursera.org/learn/state-estimation-localization-self-driving-cars/lecture/OCrZc/lesson-5-limitations-of-the-ekf.

[21] Jonathan Kelly. *State Estimation and Localization for Self-Driving Cars.* Coursera Lecture Video. 2019. URL: https://www.coursera.org/learn/state-estimation-localization-self-driving-cars/lecture/2imn3/lesson-2-multisensor-fusion-for-state-estimation.

[22] GIS Geography. *Trilateration vs Triangulation – How GPS Receivers Work.* URL: https://gisgeography.com/trilateration-triangulation-gps/.

[23] Jonathan Kelly. *State Estimation and Localization for Self-Driving Cars.* Coursera Lecture Video. 2019. URL: https://www.coursera.org/learn/state-estimation-localization-self-driving-cars/lecture/TBMU9/lesson-2-the-inertial-measurement-unit-imu.

[24] Lucas Vieira. *File:3D Gyroscope.png.* 2006. URL: https://commons.wikimedia.org/wiki/File:3D_Gyroscope.png.

[25] Patrick McGarey. *Visual Odometry (VO).* cs.toronto.edu. 2016. URL: http://www.cs.toronto.edu/~urtasun/courses/CSC2541/03_odometry.pdf.

[26] Jonathan Kelly. *State Estimation and Localization for Self-Driving Cars.* Coursera. 2019. URL: https://www.coursera.org/learn/state-estimation-localization-self-driving-cars/home/welcome.

[27] Steven Waslander. *Motion Planning for Self-Driving Cars.* Coursera. 2019. URL: https://www.coursera.org/learn/motion-planning-self-driving-cars/lecture/EPw6F/lesson-1-creating-a-road-network-graph.

[28] David J. Malan and Brian Yu. *CS50's Introduction to Artificial Intelligence with Python.* EDX. 2019. URL: https://courses.edx.org/courses/course-v1:HarvardX+CS50AI+1T2020/courseware.

[29] Geoff Boeing. "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks". In: *Computers, Environment and Urban Systems* 65 (2017), pp. 126–139. ISSN: 0198-9715. DOI: https://doi.org/10.1016/j.compenvurbsys.2017.05.004. URL: http://www.sciencedirect.com/science/article/pii/S0198971516303970.

[30] Swarnendu Ghosh et al. "Understanding Deep Learning Techniques for Image Segmentation". In: *CoRR* abs/1907.06119 (2019). arXiv: 1907.06119. URL: http://arxiv.org/abs/1907.06119.

[31] Alberto Garcia-Garcia et al. "A Review on Deep Learning Techniques Applied to Semantic Segmentation". In: *CoRR* abs/1704.06857 (2017). arXiv: 1704.06857. URL: http://arxiv.org/abs/1704.06857.

[32] Ali Borji et al. "Salient Object Detection: A Survey". In: *CoRR* abs/1411.5878 (2014). arXiv: 1411.5878. URL: http://arxiv.org/abs/1411.5878.

[33] Jifeng Dai, Kaiming He, and Jian Sun. "Instance-Aware Semantic Segmentation via Multi-Task Network Cascades". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[34] Ekaterina H Spriggs, Fernando De La Torre, and Martial Hebert. "Temporal segmentation and activity classification from first-person sensing". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. 2009, pp. 17–24.

[35] Bo Peng, Lei Zhang, and David Zhang. "Automatic image segmentation by dynamic region merging". In: *IEEE Transactions on image processing* 20.12 (2011), pp. 3592–3605.

[36] Hong-Ying Yang et al. "Color texture segmentation based on image pixel classification". In: *Engineering Applications of Artificial Intelligence* 25.8 (2012), pp. 1656–1669. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2012.09.010. URL: http://www.sciencedirect.com/science/article/pii/S0952197612002412.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[38] E. Shelhamer, J. Long, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (Apr. 2017), pp. 640–651. DOI: 10.1109/TPAMI.2016.2572683.

[39] Xiaolong Liu, Zhidong Deng, and Yuhan Yang. "Recent progress in semantic image segmentation". In: *CoRR* abs/1809.10198 (2018). arXiv: 1809.10198. URL: http://arxiv.org/abs/1809.10198.

[40] Jonathan Masci et al. "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction". In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 52–59. ISBN: 978-3-642-21735-7.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[42] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[43] Gabriel J. Brostow et al. "Segmentation and Recognition Using Structure from Motion Point Clouds". In: *ECCV (1)*. 2008, pp. 44–57.

[44] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[45] Adam Paszke et al. "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*. 2017.

[46] Jeremy Howard et al. *fastai*. https://github.com/fastai/fastai. 2018.

[47] Adam Paszke et al. "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation". In: *CoRR* abs/1606.02147 (2016). arXiv: 1606.02147. URL: http://arxiv.org/abs/1606.02147.

[48] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[49] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *CoRR* abs/1606.04797 (2016). arXiv: 1606.04797. URL: http://arxiv.org/abs/1606.04797.

[50] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852.

[51] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[52] Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *arXiv preprint arXiv:1511.07122* (2015).

[53] Z. Hall and E. L. Cao. “Omnidirectional viewing using a fish eye lens.” In: *SPIE Optics, Illumination, and Image Sensing for Machine Vision* (Oct. 1986), 728:250–256.

[54] S. J. Oh and E. L. Hall. “Calibration of an omnidirectional vision navigation system using an industrial robot.” In: *Optical Engineering* (Sept. 1989), 28(9):955–962.

[55] M. M. Fleck. “Perspective projection: The wrong imaging model”. In: *Research report* (1995), pp. 95–01.

[56] A. Basu and S. Licardie. “Alternative models for fish-eye lenses”. In: *Pattern Recognition Letters* (1995), 16(4):433–441.

[57] F. Hamit. “New video and still cameras provide a global roaming viewpoint”. In: *Advance Imaging* (Mar. 1997), pp. 50–52.

[58] M. V.-P. Greguss. “Centric minded imaging in space research.” In: *International Workshop on Robotics in Alpe-Adria-Danube Region* (June 1998), pp. 121–126.

[59] P. Greguss. “Panoramic imaging block for three-dimensional space”. In: *U.S. Patent:* (Jan. 1996), pp. 4, 566, 763.

[60] J. S. Chahl and M. V. Srinivassan. “Reflective surfaces for panoramic imaging.” In: *Applied Optics* (Nov. 1997), 36(31):8275–8285.

[61] Herman Herman Mark Ollis and Sanjiv Singh. “Analysis and Design of Panoramic Stereo Vision Using Equi-Angular Pixel Cameras”. In: *The Robotics Institute Carnegie Mellon University* (Jan. 1999).