C O V E N T R Y
U N I V E R S I T Y

Faculty of Engineering, Environment and Computing

School of Computing, Electronics and Mathematics

Evaluation of Low-Cost Autonomous Driving Sensor kit

Author: Kankan Kumar Sarkar

SID: 8890977

Supervisor: Dr. Ming Yang (CU) and Dr. Prithvi Sekhar Pagala (KPIT)

Academic Year: 2019/20

Submitted in partial fulfilment of the requirements for the Degree of MTech in Automotive Electronics

Declaration of Originality

I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Statement of copyright

Statement of ethical engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)

Signed: Kankan Sarkar Date: 09/03/2020

Please complete all fields.

First Name:	Kankan
Last Name:	Sarkar
Student ID number	8890977
Ethics Application Number	P96683
1st Supervisor Name	Dr. Ming Yang
2nd Supervisor Name	Dr. Prithvi Sekhar Pagala

Table of Contents

1. Table of Figures

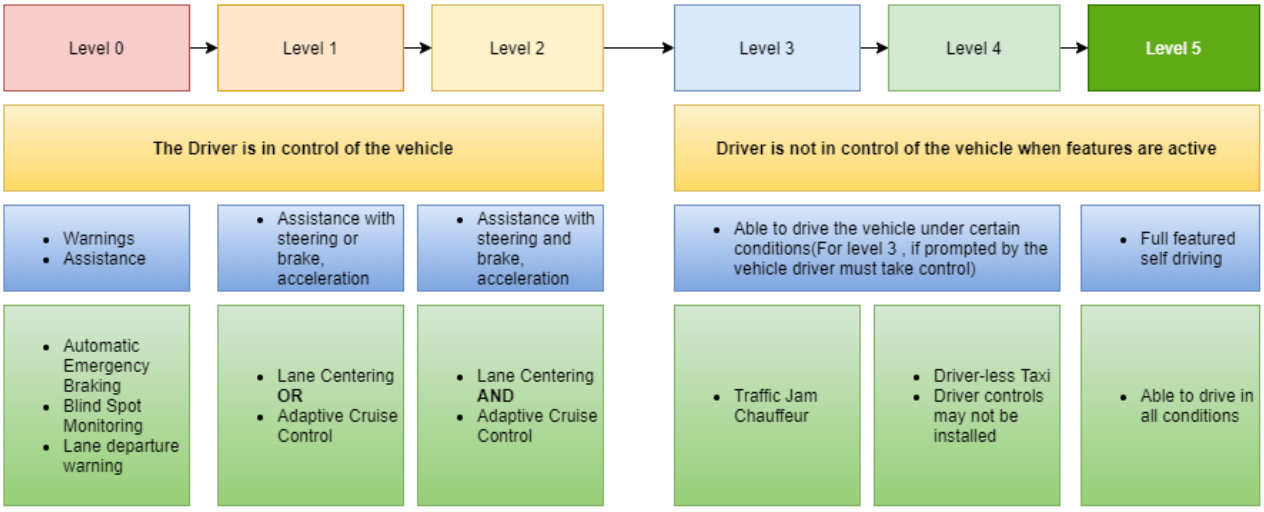
2. List of Tables

3. Introduction

Over last 10 years the automotive industry has seen monumental growth in electrification of vehicle and complementary rise in autonomous features like lane assist/lane keeping, automatic parking, cruise control , auto-steer, auto-summon, automatic emergency braking, side collision warning etc. Historically , autonomous driving concept dates to 1920's. The major milestones that contributed to development of autonomous driving technology are mentioned in below table.

Year	Technology Milestone
1925	First Demonstration of Remote-controlled Vehicle
1939	First Self Driving Car by GM based on electromagnetic wires embedded in road, Introduction of a humanoid named ELEKTRO
1948	Turtlebot, developed by Grey Walter who could find charging station autonomously
1968	Shakey by Stanford, a mobile robot equipped with machine vision and controlled by a computer
1977	Integration of camera by Japans Tsukuba to improve self-driving vehicle performance
1986	VAMORS by Mercedes achieved 96kph and advanced features like lane assist
1996	Project ARGO by University of Parma implemented a low-cost autonomous driving solution with only 2 cameras (stereoscopic vision) , Introduction of Honda ASIMO
1997	NASA Launched Pathfinder to Mars
2002	iRobot Launches Roomba , A room cleaner robot, able to self-navigate in indoor environment
2004	STANLEY from Stanford completed the DARPA autonomous driving challenge
2010	Google enters the race, the research team started experimentation with camera , lidar , radar , imu's.
2014	Introduction of Tesla autonomous driving(Hardware 1)
2015	Google logged 1million miles of self-driving without a accident(~13 collisions)
2016	Google self-driving vehicle meets accident due to computer fault; Tesla introduces autonomous driving hardware 2
2019	Tesla released hardware v3 with tesla processors

SAE J3016 (SAE, 2018) defines "Levels of Driving Automation" the summary of autonomous driving levels are explained in below.



Autonomous driving enables a vehicle to navigate without human intervention, ensuring maximum structural safety to the vehicle , its occupants and other vehicles , while complying with the road laws. Autonomous vehicles will bring more convenient and easier driving experiences for its occupants. Major advantages of Autonomous vehicles are listed below

- Decreased number of Accidents.
- Lessens Traffic Congestion.
- Stress free parking , Vehicle summon.
- Time-Saving Vehicle.
- Accessibility to transportation for everyone.

Autonomous vehicles sense their surroundings with sensors (RADAR, LiDAR, Camera, GPS, IMU etc.) and identifies navigation paths of a known/unknown environment. For any autonomous vehicle it is important to keep track of the motion (R, et al., 2014) , current location and the goal. The AV being a machine relies on sensors and vehicle information (such as lidar , radar , cameras, imu's , gps , vehicle network) for mapping a surrounding environment. A central computer keeps track of all the sensor information , processes the data and provides output as steering wheel , acceleration and brake signal. The project aims to develop a low-cost autonomous driving sensor kit for after-market retrofitting . The major driving factors of the project are

I. Cost : The proposed solution must be low cost without compromising the safety and reliability of the solution.
II. Processing time : High resolution data requires much more processing power and with processing power the cost also trends. So, one of the objectives of the project is to keep the balance between processing power and cost.
III. Accuracy : Accuracy is one of key factor on which the feature detectors are evaluated.

4. Project Background

Emerging cities across the globe faces traffic congestion every day. A snapshot taken from (tomtom, 2019) shows the traffic congestion index in the below table

Table 1 Traffic Index

Rank	City	Country	Congestion Level
1	Bengaluru	India	71%
2	Manila	Philippines	71%
3	Bogota	Colombia	68%
4	Mumbai	India	65%
5	Pune	India	59%

Referring the above table , top 3 wort traffic congested countries are India , Colombia and Philippines. Analysing the most sold vehicles in the top 3 countries(2019) , the below table is compiled

Table 2 Vehicle Popularity Index

Countries	India	Colombia	Philippines
First	Maruti Suzuki Swift	Renault Sandero	Toyota Vios
Second	Maruti Suzuki Alto	Renault Logan	Mitsubishi Xpander
Third	Toyota Innova	Chevrolet Onix	Toyota Rush

From the above data a primitive assumption can be drawn that most of the vehicle stuck in the traffic belongs to the above category. Analysing the vehicle specifications , it is concluded that most of the vehicles do not have Level 0-1 driving automation features.

If the above vehicles can be retrofitted with an autonomous driving kit which can take over in traffic , potentially freeing the driver of the wheels , we are looking at saving ~ 1 hour a day (tomtom, 2019). Moreover, implementation of autonomous driving will reduce the traffic accident and help managing traffic. Potential user groups would be

1. Individual vehicle owners.

2. Fleet owners.

3. Ride sharing services.

4. Autonomous utility researchers.

5. Government entities(Can monitor if someone's not following the traffic rule).

6. Insurance companies can use the data to investigate accidents.

With the motivation to reduce traffic congestion and improve productivity of the traffic management system the project aims to ensure passenger safety by enabling Level2-3 autonomous driving features . Similar implementation is already in the market by (Comma.ai, 2018) which uses only one camera to facilitate features like Adaptive Cruise Control and Automated Lane Centering on the highway or in stop-and-go traffic.

To achieve the project aim, appropriate sensor suite must be selected and a navigation algorithm to be implemented. Now there has been/ongoing extensive research regarding the appropriate sensor suite. Majorly sensors include LiDAR , RADAR, Ultrasonic Sensors , Cameras(RGB, Depth), GPS, INS, Vehicle information, Audio sensors etc. Selection of a sensor suite needs economic , computational , aesthetic and production consideration. On the other hand , Human Drivers do not have LiDAR/RADAR like sensor input , but they have below inputs (internal/external)

- Vision (Environment, location of other vehicles/road-side structures etc)
- Sound (Hazard warning—horn, Road type, moving vehicle speed etc)
- Others(GPS, Inertial sensors, ultrasonic sensors, vehicle phase, internet etc).

So theoretically , using only camera human like driving feature implementation is possible.

5. Project Objectives

The project goal is to assess feasibility of a low-cost autonomous driving kit for vehicles without Level 2-3 autonomy features like ACC ALC etc.

Project Objectives are as following

1.1 Objective 1(70H)

- Study similar existing solutions/technologies/ ongoing work /user needs. Sources include web pages , (Scholar, 2004) , (ieee, 1963) , (ResearchGate, 2008) , (arxiv, 1991).
- Study OEM implemented Level 2-3 autonomy features. Specially their sensor suite , overview of the data processing system.

The objective can be subdivided into three phases.

I. Autonomous vehicle market trend study based on major milestones(10H).
II. Research on similar existing solutions or ongoing research projects and their application areas(20H).
III. Study basics of sensor working principle, sensor fusion and standard mapping algorithms(40H).

1.2 Objective 2(30H)

Create a realistic simulation environment for implementation of standard mapping algorithms , create possible sensor suites based on background study , select an appropriate vehicle and integrate sensors in the vehicle. The objective 2 can be subdivided into the below sub phases

I. Compare and select available simulators intended for autonomous navigation research(10H).
II. Compare and select appropriate sensors to be integrated with the simulated vehicle.(Vehicle model customization is out of scope)(10H).
III. Test sensor integration , visualization tools and mapping/navigation stack(10H)

1.3 Objective 3(60H)

Implement standard mapping algorithm with combination of different sensor suite from Objective 2. The objective is further subdivided as below

I. Study and understand ROS programmer's model(20H).
II. 3D LiDAR(360) based Mapping Algorithm implementation(10H).
III. 2D LiDAR(360) based Mapping Algorithm implementation(10H).
IV. Camera(Front) based Mapping Algorithm implementation(10H).
V. Camera(Front) + LiDAR based Mapping Algorithm implementation(10H).

1.4 Objective 4(120H)

Development of a vision-based mapping algorithm in a dynamic environment, this is further divided into below sub objectives

I. Study Basics of Image processing in python(10H).
II. Study CNN as image classifier(35H).
III. Design and implement a suitable image classifier. Implementation include train/testing and optimization(35H).
IV. Design a path planner mask(20H).
V. Design and development of robust navigation unit(20H)

1.5 Objective 5(50H)

Comparison of Developed mapping algorithm vs standard mapping algorithm.

6. Overview of the report

The Project document is divided into below sub sections

I. Introduction
II. Project background , Objective and Literature research.
III. Project Design
IV. Project Implementation
V. Project outcomes as test result
VI. Conclusion and future scope.

7. Literature Review

Building map and environment perception is one of the fundamental challenges in autonomous navigation. Once the vehicle can localize in a prebuilt map , it is easy to navigate towards the goal. In robotics this is known as SLAM or Simultaneous Localization and Mapping solution. Since the inception of SLAM , there has been significant progress and in recent years it has reached a certain level of maturity . SLAM can be further subdivided into Offline SLAM and Online SLAM.

I. Offline SLAM : This approach is easy and frequently used for generating street maps. A mobile robot with equipped sensor-suite is manually traversed in an environment. Sensor data is logged and saved for offline processing. The sensor data is later processed, and grid map is generated for localization and path planning. (Frese, et al., n.d.) (Kuzmin, n.d.). Though it's the easiest and low-cost implementation of SLAM , this has significant challenges. The offline-SLAM cannot be subjected to dynamic environment and the SLAM process is slow.

II. Online SLAM: Online SLAM deals with real-time mobile-robot motion tracking and data analysis to take independent decisions (Kuzmin, n.d.). Online-SLAM is suitable for dynamic environment and it is capable to perform autonomous manoeuvre to destination (Pedrosa, et al., 2013). Though Online-SLAM feature real-time localization and navigation capability , online slam implementations are computationally expensive solution.

Among all proposed SLAM solutions over last 3 decades , the probabilistic SLAM approach (Thrun, et al., 1998) shows promising result . Probabilistic SLAM solutions Ex. EKF and RBPF were used extensively for LiDAR based and Visual SLAM. The basics of both algorithms are explained below

1.6 EKF or Extended Kalman Filter

The EKF-SLAM estimates the current pose of the robot instead of estimating the entire trajectory. The EKF-SLAM observes landmarks(objects) in the environment for perception (Smith, et al., 1987). The principle of the EKF-SLAM is explained below (Filbrandt, et al., 2019).

Assumption : The environment is 2D , robot control command is U, Observation =Z , Map of the environment is M and robot path = X

At any time, T

the robot control commands would be

$U_{1:T} = \{U_1, U_2 \ldots\ldots U_T\}$

The robot observations:

$Z_{1:T} = \{Z_1, Z_2 \ldots\ldots Z_T\}$

The robot path:

$X_{0:T} = \{X_1, X_2 \ldots\ldots X_T\}$

The State Space representation of the robot in the environment , where m represents the list of landmark positions, x, y are the robot position coordinates and θ is the orientation.

$X_t=(x, y, \theta, m_{1,x}, m_{1,y}, \ldots\ldots\ldots, m_{n,x}, m_{n,y})$

The above equation can be broken into pose matrix(μ) and covariance matrix(σ).

Pose matrix(μ)=$\begin{bmatrix} X \\ m \end{bmatrix}$ where X=(x ,y ,θ)

Covariance matrix(σ)=$\begin{bmatrix} \sigma_{XX} & \sigma_{Xm} \\ \sigma_{mX} & \sigma_{mm} \end{bmatrix}$

Here σ_{XX} relates to the covariance of the pose.

σ_{mm} is the uncertainty of the landmark and their correlation

σ_{mm} , σ_{Xm} is the correlation between the pose and the landmark

To the above pose and covariance matrix the EKF is applied , its an iterative process and iteration can be subdivided into below fundamental steps . Assumption in the environment there are 2 landmarks as m_1,m_2.

a) Prediction of State: Estimate a new position of the mobile robot given the control input. In this step only the pose vectors are updated in the pose matrix . In the covariance matrix the elements related to the robot's pose are updated (robots pose and the landmark observations). Predictions are expressed in below matrices (highlighted below)

I. Pose matrix(μ)=$\begin{bmatrix} X \\ m_1 \\ m_2 \end{bmatrix}$

II. Covariance matrix(σ)=$\begin{bmatrix} \sigma_{XX} & \sigma_{Xm_1} & \sigma_{Xm_2} \\ \sigma_{m_1X} & \sigma_{m_1m_1} & \sigma_{m_1m_2} \\ \sigma_{m_2X} & \sigma_{m_2m_1} & \sigma_{m_2m_2} \end{bmatrix}$

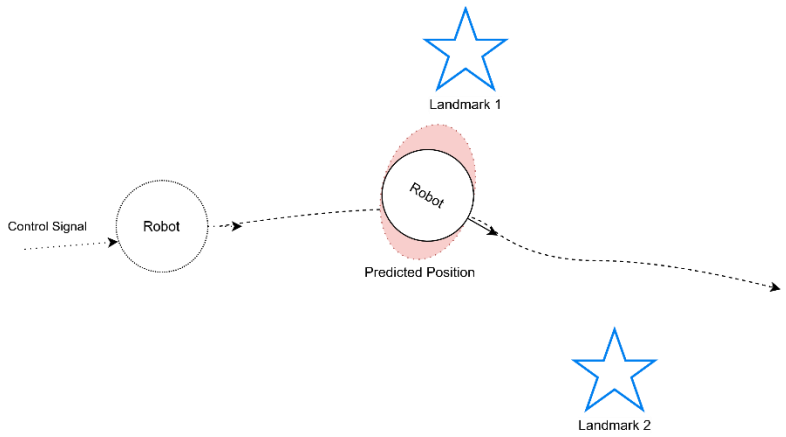


Figure 1 State Estimate

b) Measurement Prediction: Given the best estimate of the robot's position(as per previous step) predict the landmark position(highlighted below)

I. Pose matrix(μ)=$\begin{bmatrix} X \\ m_1 \\ m_2 \end{bmatrix}$

II. Covariance matrix(σ)=$\begin{bmatrix} \sigma_{XX} & \sigma_{Xm_1} & \sigma_{Xm_2} \\ \sigma_{m_1X} & \sigma_{m_1m_1} & \sigma_{m_1m_2} \\ \sigma_{m_2X} & \sigma_{m_2m_1} & \sigma_{m_2m_2} \end{bmatrix}$

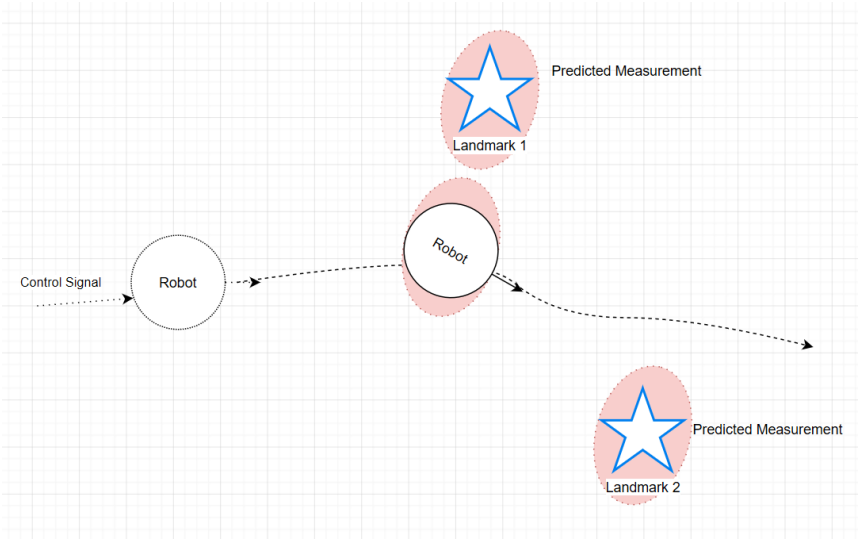


Figure 2 Measurement Estimate

c) Measurement : Measure the landmarks using sensor (LiDAR, Camera, Radar , Ultrasonic etc.)

d) Data Association: Measurement taken from sensors are associated to respective landmark.

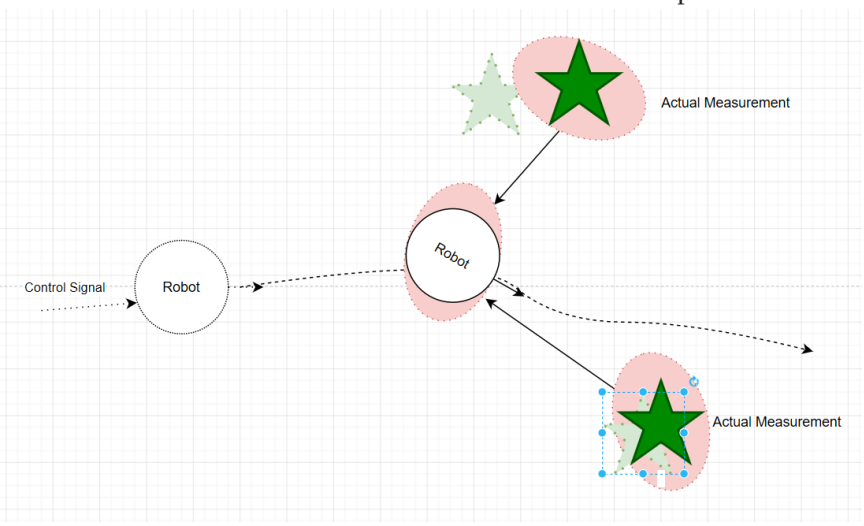


Figure 3 Data Association

e) Update : Correct the robot pose and landmark observation(highlighted below)

I. Pose matrix(μ)=$\begin{bmatrix} X \\ m_1 \\ m_2 \end{bmatrix}$

II. Covariance matrix(σ)=$\begin{bmatrix} \sigma_{XX} & \sigma_{Xm_1} & \sigma_{Xm_2} \\ \sigma_{m_1X} & \sigma_{m_1m_1} & \sigma_{m_1m_2} \\ \sigma_{m_2X} & \sigma_{m_2m_1} & \sigma_{m_2m_2} \end{bmatrix}$

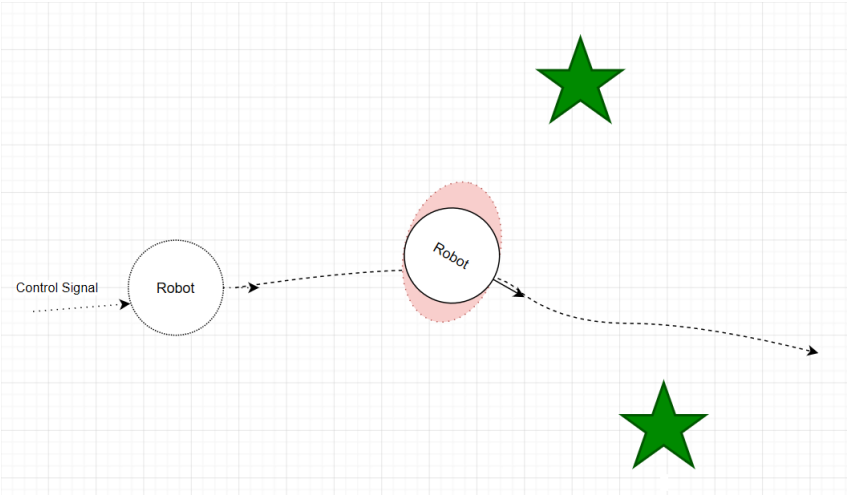


Figure 4 Current Error Calculation

EKF suffers from poor performance for non-linear system and its computationally not efficient , which leads to the next implementation RBPF.

1.7 RBPF or Rao-Blackwellzed Particle Filter

It's a particle filter-based SLAM algorithm proposed by (Murphy & Russell, 2000). Particle filtering in high dimensional state-spaces can be computation intensive as large number of samples are needed to represent the posterior . However, the state space variables can be decreased analytically by Rao-Blackwellisation (Boyen & Koller, 2013)

The joint posterior can be expressed as $p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1})$ where $x_{1:t}$ denotes the robot trajectory till time t, m is the map , $z_{1:t}$ is the robot observations so far(till time t) and $u_{1:t-1}$ is the odometry estimation so far(till time t). Assuming the odometry and sensor data are erroneous which can create error in map calculation over period. Hence sensor data correction is required during joint posterior calculation. Referring to (Murphy & Russell, 2000) and (Grisetti, et al., 2007) the joint posterior can also be expressed as below

$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid x_{1:t}, z_{1:t}).p(x_{1:t} z_{1:t}, u_{1:t-1})$ ---------- Equation 1

A motion model is necessary to generate new particles(x_t) from previous particle (x_{t-1}) by observing the odometry data. A importance weight is assigned to each particle value and lower weighted particles were excluded from map update process. By repeating the same process , using only small sets of particles RBPF tracks the robot pose and able to move around autonomously in an unknown environment.

(Montemerlo, et al., 2002)'s FastSLAM makes use of RBPF's to estimate robot's poses and tracks the landmarks using EKF. Based on RBPF, (Kou, et al., 2011) implements a light weight RBPF-based SLAM for indoor environment which uses sequential segmentation algorithm to increase the reliability of line segments extraction from raw scan data while simultaneously minimizing the required processing power compared to parent RBPF SLAM approach.

The above SLAM algorithms are generic and can be applied to different sensor suite for mapping and navigation. Preferred sensors for SLAM are 1. LiDAR 2. Visual(Camera based) often the sensors are equipped with IMU , GPS , ultrasonic sensors to give more robust SLAM result.

1.8 LiDAR SLAM

2D lidars are inexpensive sensors and extensively used in indoor autonomous robots for SLAM(Grid map approach). However , LiDAR's suffer from cumulative error , which makes the map data erroneous but modern SLAM implementations correct the sensor errors by probabilistic prediction approach. In 2007 , (Grisetti, et al., 2007) Gmapping was implemented based on RBPF which improves the positioning accuracy and reduces computational complexity by improving distribution and adaptive resampling. In 2011, Hector (Kohlbrecher, et al., 2011) slam used Gauss-Newton method for scan matching, it relies on high precision LiDAR instead of odometry. In 2016 , (Hess, et al., 2016) developed a SLAM , which is commonly known as Google Cartographer , reduces accumulative error(LiDAR) by applying loop closure to local and global map.

1.9 Visual SLAM

Visual SLAM is more computation intensive compared to its LiDAR counterpart. But visual slam provides the most accurate way to map the environment visually(colour , shape type etc.). Visual SLAM makes use of features extracted from images at different poses. (Davison, et al., 2007) proposed sparse feature tracker based on EKF. The observation uncertainty considered as a probability density function , which is obtained by recursive calculation and observation model. RBPF based SLAM (Sim, et al., 2005) implementations have high precision, but it needs more particle vectors which increases the computation demand. ORB-SLAM (Mur-Artal, et al., 2015) uses Bundle Adjustments in the backend for tracking , mapping , re-localization and loop closing. This makes the system more efficient and reliable . The SLAM uses ORB (Rublee, et al., n.d.) features to track frame by frame objects which is the key behind the Visual SLAM. As the name suggests the SLAM is immune to orientations and even can recover in case of tracking failure.

1.10 Sensor Fusion

Generally, the above discussed SLAM approaches are equipped with assistant sensors (Encoders , IMU , GPS) . The assistant sensor can be used as supervisory or redundant data source in case of main sensor temporary/primary failure. In (Xu, et al., 2018)'s paper demonstrates how EKF can be used to blend visual sensor data with LiDAR data and navigated using an improved tracking strategy even if the visual feed is temporarily not available. In the (Jiang, et al., 2019) 's paper SLAM is implemented vision fusion with low cost LiDAR and a 2.5D map approach is introduced which can be potentially used for semantics of a scene or object. (Lopez, et al., 2017) demonstrates SLAM based on LiDAR and IMU using EKF for sensor fusion.

1.11 CNN

Convolutional Neural Network is a class of DNN or Deep neural network and well suited for matrix data interpretation . A generic CNN architecture is shown explained (O'Shea & Nash, 2015), (LeCun, et al., 1998)

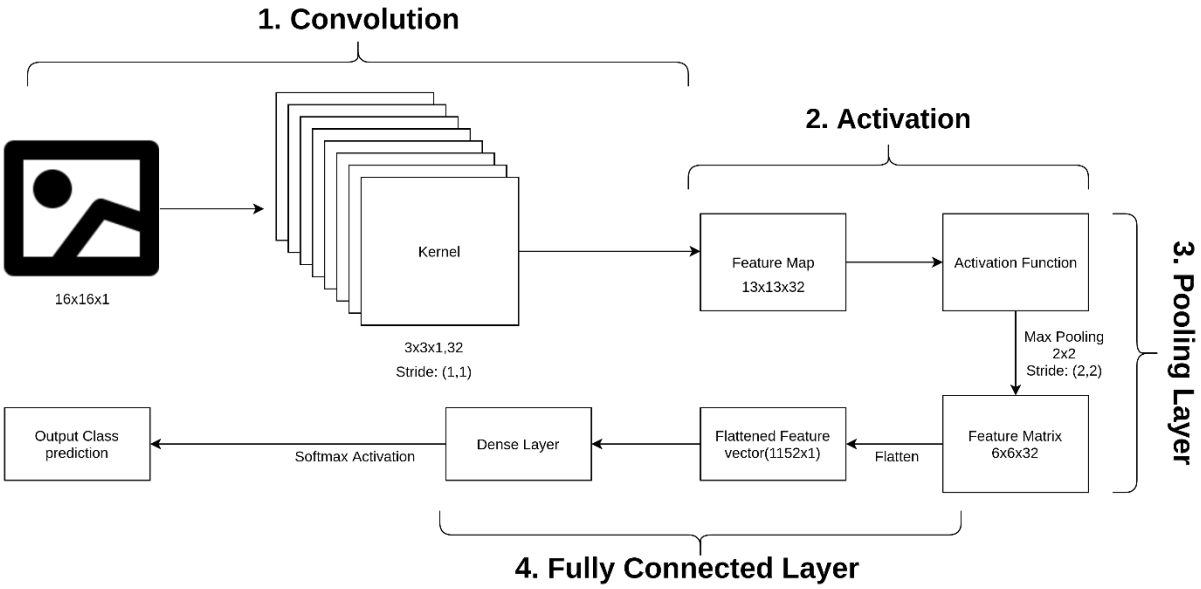


Figure 5 CNN Architecture

a) Convolution : In a nutshell convolution can be expressed by below mathematical expression

$$(I * K)(t) = \int_{-\infty}^{\infty} I(\tau) \cdot K(t - \tau)\, d\tau$$

I=Input series data (in the above case time series data),K=Kernel or Filter, t=Overall time, τ =instance of time t , as τ cannot be less than 0 and bound by overall time t the above equation can be further simplified as below

$$(I * K)(t) = \int_{0}^{t} I(\tau) \cdot K(t - \tau)\, d\tau$$

For an image I(x,y) this equation can be written as below where kernel/filter =K(i,j)

$$(I * K)(x, y) = \int_{0}^{x} \int_{0}^{y} I(x - i, y - j) \cdot K(i, j)\, didj$$

The above equation means , for a given image I(x,y) ,a filter/kernel K(i,j) is iterated through the whole image to generate a feature map (I*K). The feature map size is (Initial image size(w or h)-kernel/Filter size(w or h))+1. So, convolution is iteratively applying a filter/kernel(Ex. Edge detection) K(i,j) to an image I(x,y) with certain shift rate(stride) to generate a feature map.

In Figure 5 CNN Architecture , for an image of I(16x16) a filter/kernel K(3x3) is applied , one filter will be able to detect only one feature hence multiple filters/kernels (different feature descriptors) are applied to the same image to generate the feature map while preserving the spatial dimensions . Note: the image depth and filter depth must be same.

b) Activation : After convolution the feature map is passed through a non-linear activation function , generally leaky-relu and then it is passed to pooling layer.

c) Pooling layer : Pooling layer further compresses the feature map applying maximum value filter or Average Filter(Figure 6. Pooling Layer) This is performed based on sliding window approach(same as convolution).The final feature map size will be ((Initial image size(w or h)- kernel/Filter size(w or h))+1)/stride size.

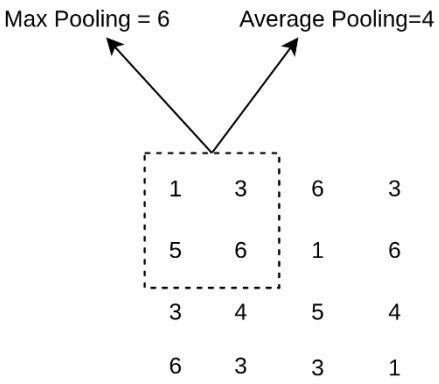


Figure 6 Pooling Layer

d) Fully connected layer: The previous three steps can be repeated until(even in asynchronous order) the feature vector size is not reduced to satisfactory size for further processing. The final layer flattens the feature vector and passes into dense layers and finally a soft-max activation function is applied to get the output class predictions.

e) Training of CNN : It is obvious that the convolution filters or kernels needs to be tuned to detect image features , this is performed during the training. The filter weights are updated/adjusted until the filters can detect the features of the training images. Once the filters/kernels can detect the respective features in the test images , the network is considered as trained.

1.11.1 Advantages of CNN

I. Sparse interaction between layers : Conventional Artificial Neural Networks need interconnection between each neuron which increases the number of parameters , hence more memory and processing(weight/bias update) is required during training however CNN's deep layers maintain sparse interaction between each other, also same feature information's are shared across different layers.

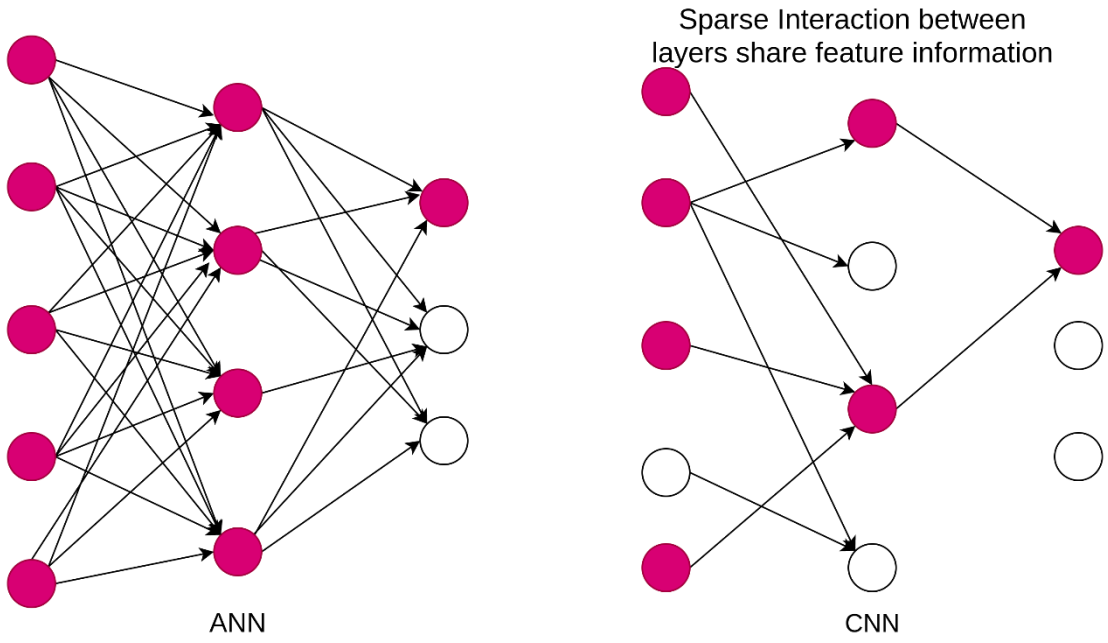
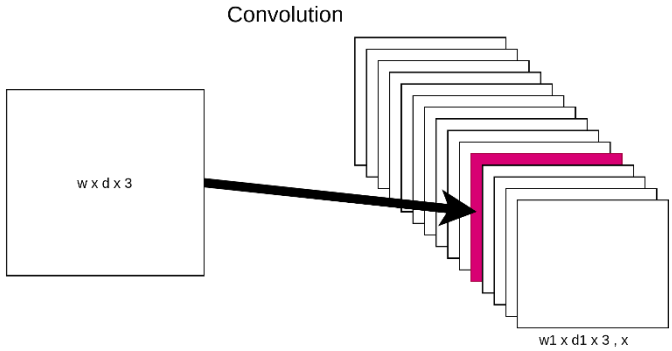


Figure 7 Sparse Interactions

II. Parameter Sharing : CNN shares parameters(filters/kernels) those contribute to same feature across different classes.



Each feature in the same depth layer is generated by the same filter that convolves the image. Same kernel can be reused to find similar features in different classes

Figure 8 Parameter Sharing

III. Equivariant Representation: CNN's are equivariant that is $f(g(x))=g(f(x))$, where function f is convolution operation and g is image translation operation . If we perform a translation first and then perform convolution operation it is same as if we convolute the image first and then

perform translation. Example Same parameters(filters/kernels) detecting edge can be reused in succeeding layers.

1.11.2 Improvement of CNN

Detection using CNN is computationally slow due to sliding window approach , in 2013 (Girshick, et al., 2013)'s implementation R-CNN or Region-Convolutional Neural Network addressed this issue . Which used segmented (Uijlings, et al., 2012)'s selective search-based region proposal to narrow down the detection region in a given image. R-CNN algorithm is shown in below illustration.

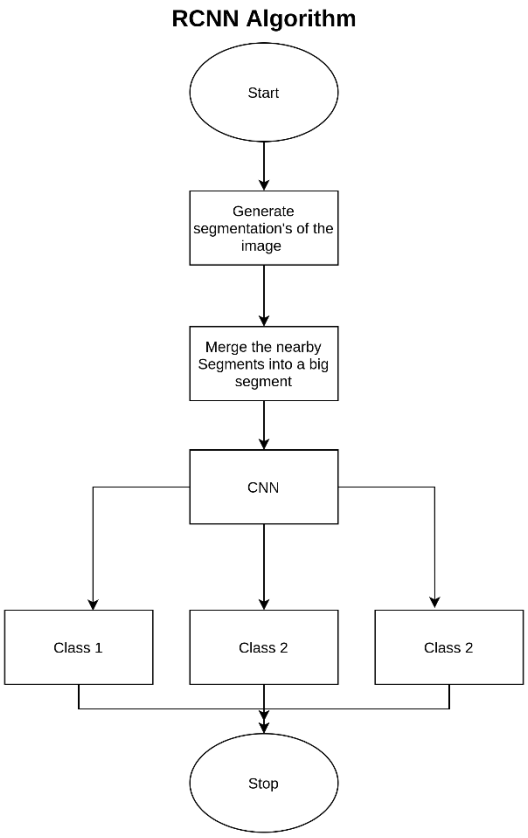


Figure 9 R-CNN Algorithm

Though R-CNN improved the object detection / training time , the region proposals were still large in number (~2000) , which needs processing. After 2 years (Girshick, 2015) came to recue with Fast R-CNN . Contrary to R-CNN , Fast R-CNN parses the convoluted feature map using a Region of Interest (RoI) pooling layer into a fixed size feature map. The fully connected layer then predicts the class following a SoftMax activation function.

1.12 Faster R-CNN

Due to the use of selective search approach , the Fast R-CNN was still computationally expensive. (Shaoqing, et al., 2015)'s Faster R-CNN solved the issue by implementing anchor boxes . The network is classified into two sub networks 1. Region proposal network , 2. Classifier . During training the both networks are simultaneously tuned together keeping the proposals fixed. Faster R-CNN scored 80%mAP against the PASCAL VOC dataset the process is shown in below image

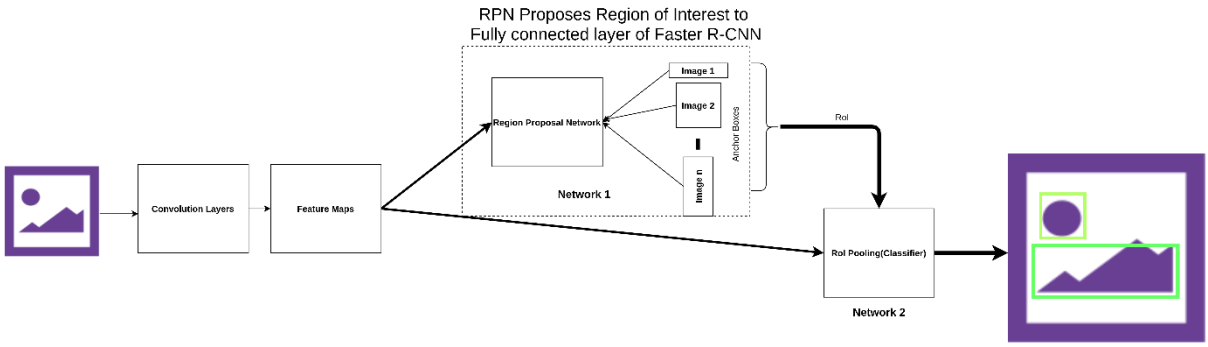


Figure 10 Faster R-CNN

1.13 Yolo

(Redmon, et al., 2015)'s Yolo or You only look once is the simplest of all the above CNN implementations whilst achieving the best of the performance. The Yolo Architecture and the algorithm is explained in below illustration.

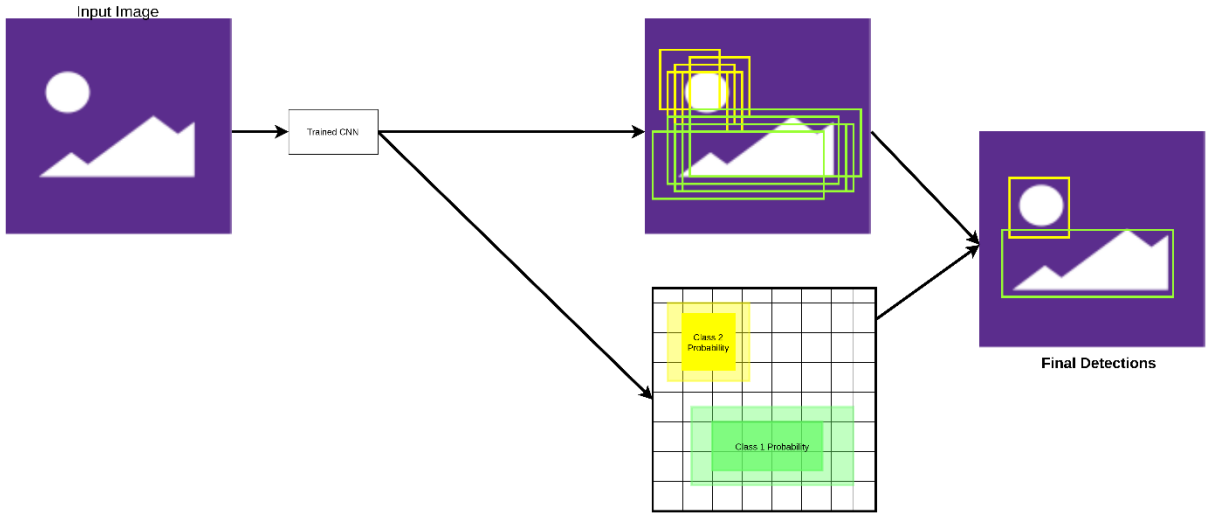


Figure 11 Yolo

1.14 Contributions of the Project

Though there has been significant progress in SLAM solutions, majorly the environment is considered static , which is not the case in real-world. This project proposes a low-cost sensor kit capable of mapping an dynamic environment using only Camera, IMU ,GPS , CAN(Not in the current scope) and self-navigate to a set goal(Not in current scope). By using a pretrained object(Traffic lights , Pedestrians, Vehicles , Road signs) classifier the environment perception will be achieved. The Output of the object classifier will be used to generate semantic information of an image frame, the semantic information will be later used to plan the robot motion in an unknown environment. Where IMU and GPS sensor data will be fused together using EKF for localization and navigation purpose(Not in current scope). (Zhong, et al., 2018) proposed a similar approach to improve SLAM solution using RGBD sensor and region proposal method , YOLOv2 model was used to extract semantic information which is channelled to the SLAM pipeline for mapping. This project's approach is similar but instead of using RGBD sensor , RGB sensor will be used with latest YOLOv3 implementation tuned to run in off the shelf SBC's in Realtime.

The unique contributions of the project are

1. Proposes a near real-time CNN(tiny-YoloV3) based mapping algorithm in while keeping the computation compatible to SBC(Ex. Nvidia TX2 Nano).
2. Integration of standard SLAM algorithms to Carla simulator. (Gmapping, R-TAB , Cartographer, Octograph, Hector mapping).
3. Sensor Fusion of GPS and INS for robust navigation in Carla Simulator.

Proposed concept architecture is shown in below image

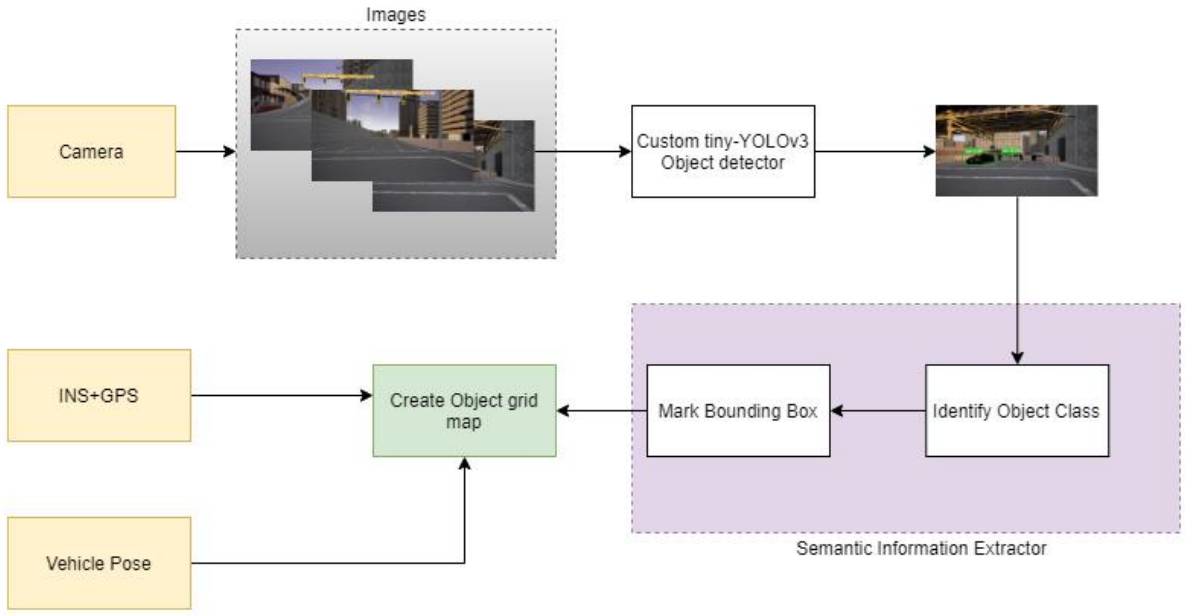


Figure 12 Project Concept

8. Methodology

Due to the time and budget constraint , it is not feasible to test and develop autonomous driving sensor suite in real-world environment . Hence a realistic simulator is used to perform experiments and draw conclusion on the project aim. After a comparative analysis of the available opensource drive simulator's , Carla Simulator is selected as the development environment. ROS or Robot Operating System is used for implementation of the standard SLAM solutions. The Standard SLAM algorithms were studied (Gmapping, Hector Mapping, Octograph , Cartographer , R-TAB) and implemented in Carla simulator supporting different sensor suite. ROS already includes prebuilt packages supporting sensor fusion , mapping and visualization which helped in reducing the integration and test time in Carla environment. After analysing the sensor suits used for standard SLAM implementations, (Tesla, 2019)'s autopilot implementation, (Comma.ai, 2018)'s product its concluded that Visual slam can feature Level2-3 Autonomous Driving. Hence, A DNN based Visual-SLAM algorithm is designed and implemented in the CARLA.

The project scope is subdivided into two parts

1. PS1 or Project Scope 1 or Implementation of standard SLAM algorithm.
2. PS2 or Project Scope 2 or Implementation of DNN based Visual-SLAM algorithm.

The nature of the project demands agile(Iterative) development where the development scope changes based on prototype test results. The PS1 development cycle is shown below.

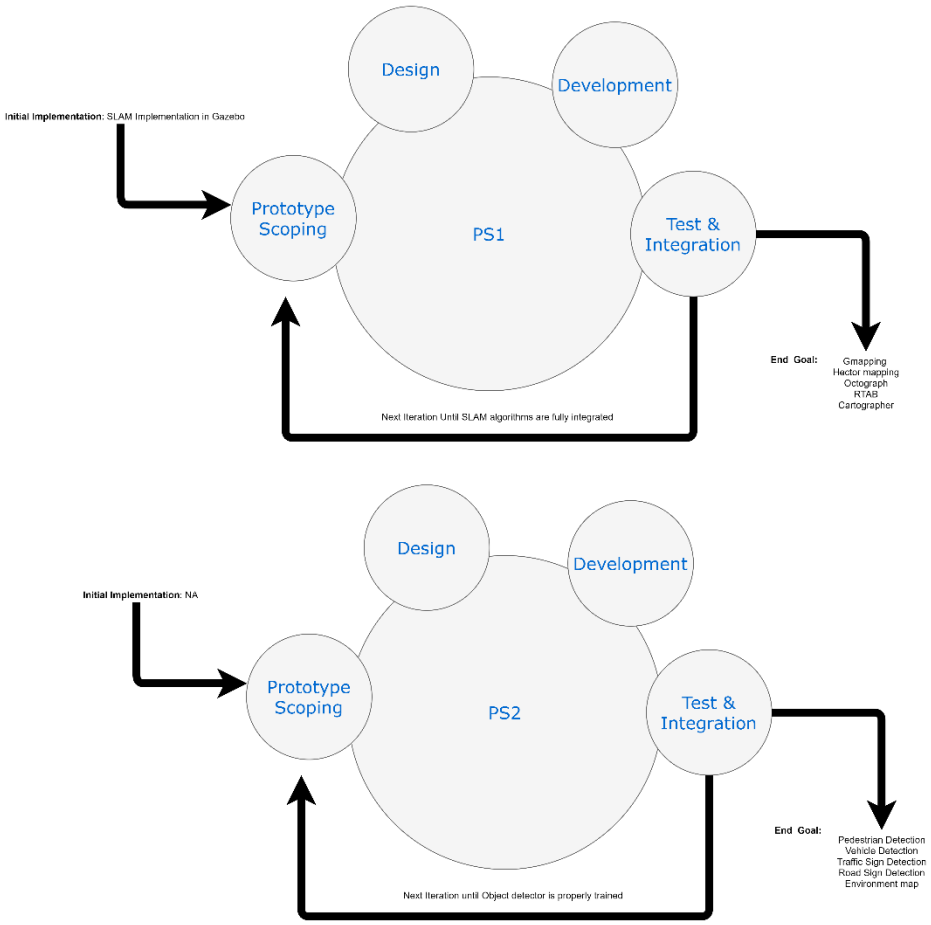


Figure 13 Development Cycle

Each iteration consists of 4 phases

I. Prototype Scoping : Each prototype needs certain feature modification/implementation on previous iteration result or requirement . The prototype scoping decides what features to be implemented in the current iteration. For example , During PS1 iteration 1 , a gazebo Gmapping implementation was used as a reference during prototype scoping of Gmapping implementation in CARLA simulator, where it was identified that the 3D LiDAR interface to be modified to implement Gmapping.

II. Design : Once the iteration scope is defined , the iteration architecture is designed , iteration architecture includes data input/output , data processing algorithm etc. For example , During PS1 iteration 1, the 3D LiDAR to 2D LiDAR scan conversion was necessary for Gmapping implementation in CARLA , where the **CarlaPCLtoLS** node was designed.

III. Development : This step of the iteration includes programming and unit testing , configuration tuning/modification.

IV. Test & Integration: The overall prototype is tested in the target environment and results are compared against the scope . For example, During PS1 iteration 1 , the Gmapping node was deployed in the Carla simulator and a grid map was expected . However , due to wrong "base_link" frame configuration the Gmapping node threw error, which resulted in test failure and needs revisiting the design phase.

9. Requirements

The major motivation behind the project was inspired by existing solutions like (Comma.ai, 2018) (Tesla, 2019) . The project requirements are gathered by document analysis and observing market trend. Project requirements sources are explained below(functional requirements are explained in the design section in detail).

I. Low Cost sensor kit : A Tesla Model 3 with OEM level 3 driving automation implementation costs approximately £42000 while most of the vehicle sold in 2019 (refer below charts) on average costs approximately £27754 , majority of these vehicles do not have Level2-3 autonomy features. Hence the sensor suite(which enables autonomous features) integration cost must compliment the vehicle price tag.

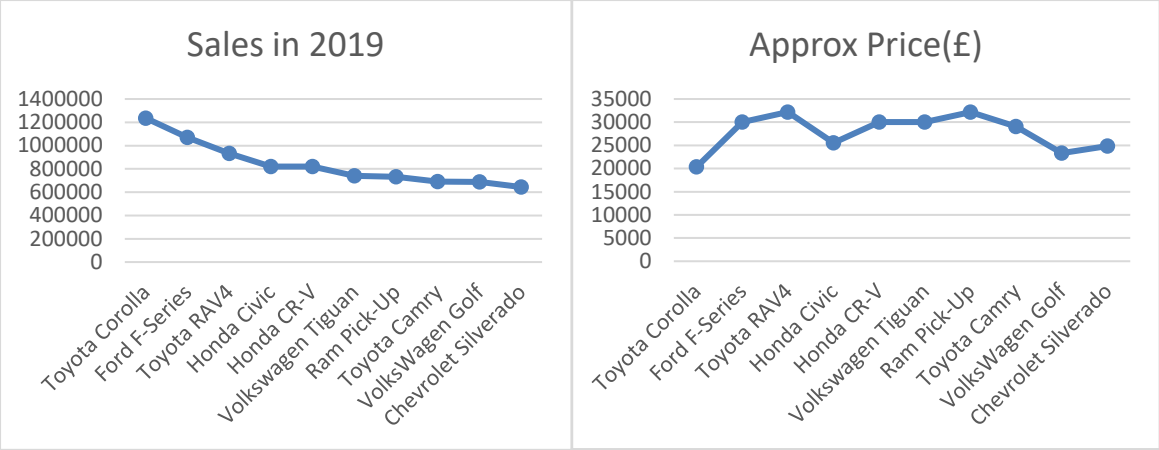


Table 4 Global 2019 Sales Quantity

II. Near Realtime processing: “What is the right hardware ?” is a straightforward question but the answer is not so simple. Because it depends on the scenario for example, in space exploration or defence application a highly priced sensor/processor which is runs state-of-the-art algorithm and provides 99% accuracy may not necessarily make it to the commercial autonomous vehicle market (Hermalyn, 2019). Hence a right balance of cost vs hardware(processor + sensors+ peripheral) must be maintained for sake of commercial applications.

III. Improvement of Current Visual SLAM: Majority of traditional visual-SLAM (Mur-Artal, et al., 2015) (Sim, et al., 2005) (Labbe & Michuad, 2013) algorithms do not provide semantic information , so the SLAM solution is not fully aware of the environment scene , moreover they don’t feature Realtime processing(30FPS). The project indents to improve the visual-SLAM approach by implementing a near real-time semantic information extractor.

10.Project Analysis

The overall project is divided into two phases as show in below illustration , implementation of phase 2 is not in the current assessment scope.

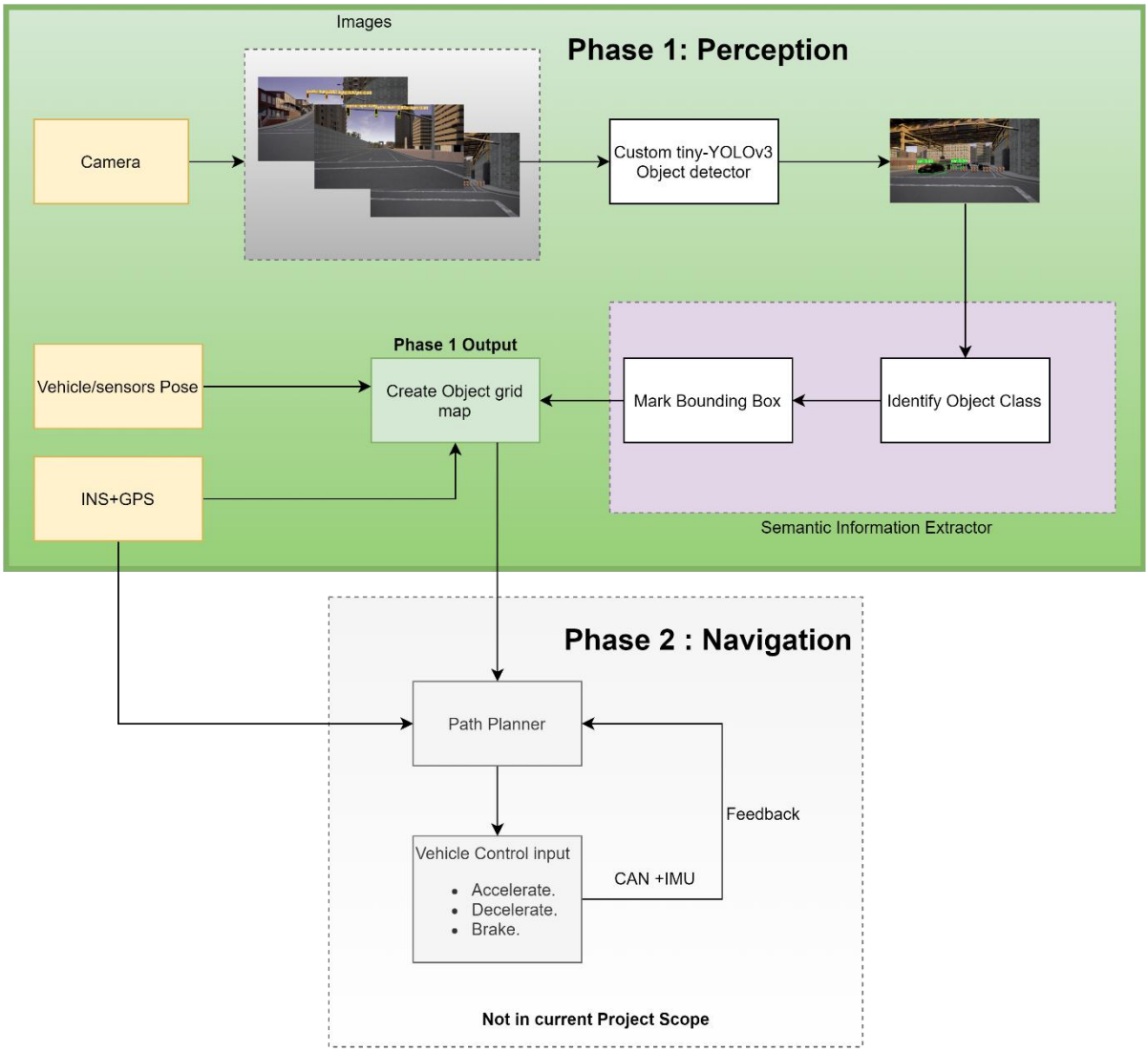


Figure 14 Project Phase Description

In the proposed Phase 1 , camera is used for simulated environment perception , motivated by (Tesla, 2019) ,(Comma.ai, 2018) ,(Mao, et al., 2019) and (Zhong, et al., 2018). The DNN based object detector gives prediction of objects(traffic lights, pedestrians , vehicles) location in each scene. Then semantic information extractor process the object classes and passes to SLAM pipeline. In the SLAM pipeline, the vehicle odometry and detected objects are fused into a single frame for further evaluation and comparison with standard mapping algorithm outputs.

1.15 Use Case Model

Because the current scope does not facilitate autonomous navigation or motion planning , the Phase 1 outcomes directly do not relate to use case models. Hence , the overall project use case is shown below

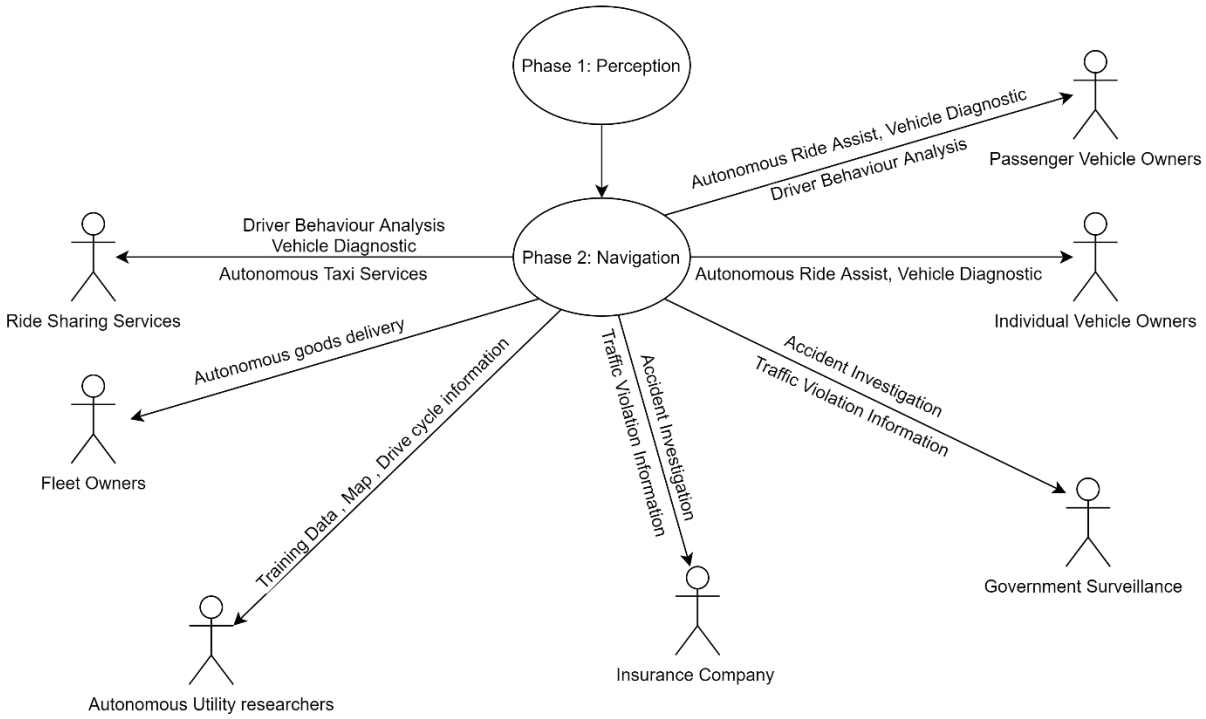


Figure 15 Use Case Model

1.16 Data Flow Model

Data flow model of Phase 1 : perception is shown below

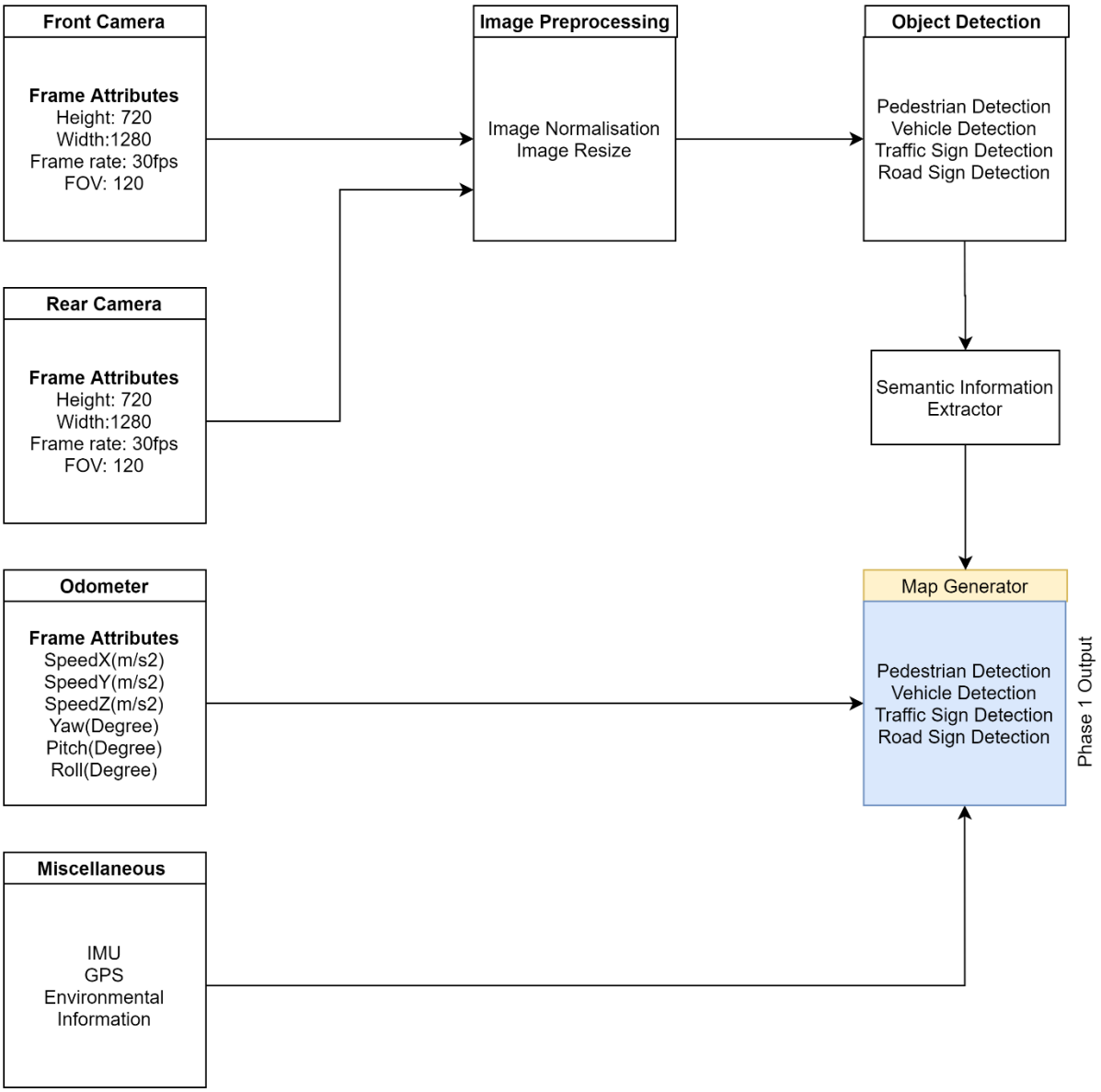


Figure 16 Data Flow Diagram

11.Design

1.17 Design Prerequisites

I. LiDAR: Light Detection and Ranging aka LiDAR is a remote sensing method that uses pulsed laser to measure absolute distance of a target. Basic working principle of LiDAR is explained in below example.

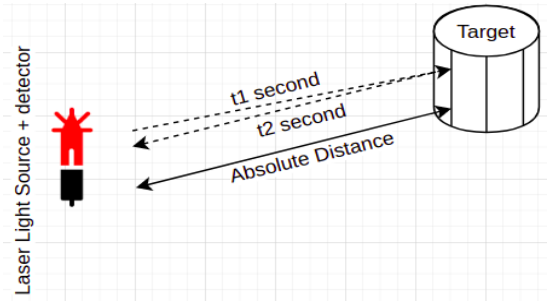


Figure 17 Lidar Principle

Assumption , the laser light source is static and speed of light in air=299704644.54m/sec

t1=Time taken by light to hit target object.

t2=Time taken by reflected light from target object to light detector.

Time of flight =(t1+t2) seconds

Hence , the absolute distance =(299704644.54 x (t1+t2)) / 2

Knowing the position of the sensor and using the perception of coordinate system the reflective surface can be calculated and represented by a point. By repeating the process many times, the lidar can build up a grid map or cloud point of the environment being mapped. Mainly LiDAR's used for autonomous navigation and mapping can be classified in two categories

a) 2D LiDAR: It is a pulse-based laser source and a detector which rotates and collects horizontal distance to targets to get data on X and Y axes (Mazzari, 2019) (Ackerman, 2016). Below is a pictorial representation of a 2d Lidar grid map. Below is an visualization of a 360deg 2D-LiDAR.

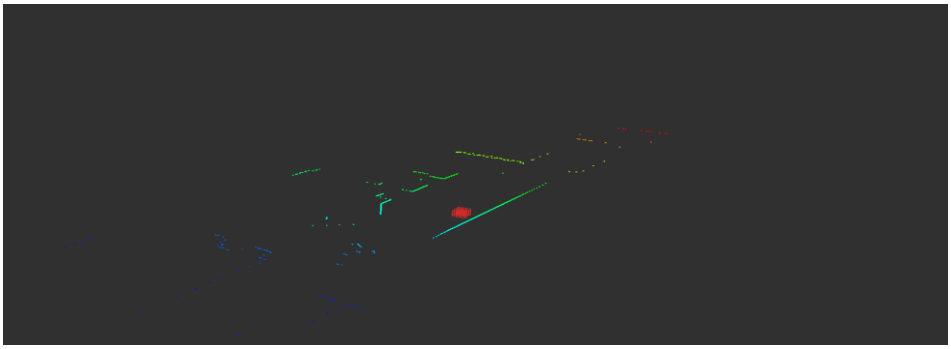


Figure 18 2D LiDAR Carla

Pros:

- Gives absolute distance data with high accuracy of surrounding environment as horizontal X and Y axes.
- Low cost compared to 3D LiDAR.
- Can see in the dark.

Cons:

- Prone to environmental debris, climate conditions(heavy rain, fog etc).
- Does not provide data about Z axis.

Model	TiM561	URG-04LX	RPLIDAR	SWEEP
Price($ – Variable)	2000	1115	398	249
Update Rate(Hz)	16363	10000	2000	500
Recommended Scan Rate(Hz)	15	10	5	2.5
Range Indoors(m)	10	5.6	5	50
Range Outdoors(m)	8	4.01	1	40
Range Accuracy	+-6cm	1% distance	0.2% of distance	1-2% of distance
Range Resolution(mm)	1	1	10	10
Field of view(deg)	270	240	360	360
Power(watt)	3W@10-28V	2.5W@5V	1.2W@5V	1W@5V
Weight(g)	250	160	170	120
Interface	ETHERNET/USB	RS232/USB	USB/UART	USB/UART
ROS Compatibility	YES	YES	YES	YES

Table 5 2D Lidar Pugh Matrix

b) 3D LiDAR: It is like 2D LiDAR , but several laser beams spread out on the vertical axe are shot to get data on X,Y&Z axes. Each laser beam will have an angle delta with the other beams. Below is an visualization of a 360deg 3D-LiDAR

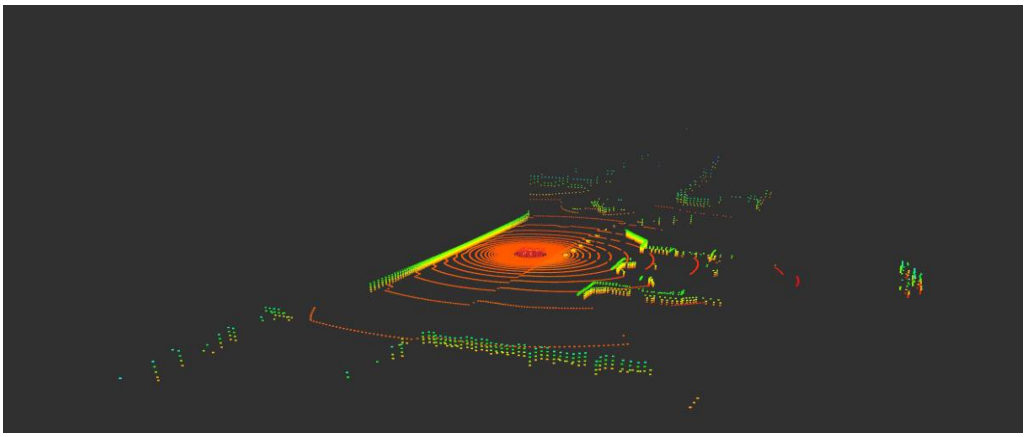


Figure 19 3D LiDAR Carla

Pros:

- 3D depth sense of the environment with high accuracy.
- Generally stand-alone sensor module and data can be collected quickly.
- Can see in the dark.

Cons:

- Ineffective in heavy rain , environment debris, fog/smog , transparent obstacles etc.
- High resolution 3D LiDAR's are costly, and this increases the final solutions price.

Model	Velodyne Puck Vlp-16	Velodyne VLP-32C	Velodyne VLP-32C	Valeo Scala
Price($ – Variable)	$	$$	$$$	NA
Number of Channels	16	32	64	NA
Range(m)	100	100	120	350m
Number of Cloud points	0.3million	1.39million	2.2million	NA
FOV(Vertical)	+15° to -15°	+10° to -30°	26.9°	3.2°
FOV(Horizontal)	360°	360°	360°	145°

Table 6 3D LiDAR Pugh Matrix

II. Camera : Images can represent visual representation of the world most accurately , especially when it comes to autonomous driving. Digital camera sensors acquire images surrounding the vehicle. Which is later processed to create an occupancy grid for autonomous route planning. Important properties of camera sensors are explained below

a. Width, height ,Pixel Density: If image resolution and the video quality is low ,the processing time will be faster but the accuracy to detect the number of objects will be low (Yadav, et al., 2017).
b. Field of View(FOV): FOV is described as the part of the environment visible through the camera at a position and orientation in space.
c. Frame rate : Frame rate is applicable to video stream; the number of frames being generated by the camera sensor is the frame rate. 30FPS is generally considered as real-time.
d. Data Output type: Cameras used for autonomous navigation output Serial data . Though parallel data lines could potentially increase the throughput and reduce latency but increases the infrastructure cost. Maximum speed for GMSL data transfer is 2.5Gbit/s with support for camera resolutions up to 1280 x 800 pixels, while CXP 2.0 delivers lightning-fast speeds of 12.5Gbit/s or 500% faster than GMSL (Lynn, 2019).

Cameras used in autonomous driving research can be classified into below types

- RGB Camera: RGB cameras are digital cameras which output images as 2d grid of pixel. RGB stands for Red , Green , Blue . Each attribute has a number from 0 to 255, so black, for example, is (0,0,0) and a pure bright red would be (255,0,0). Thousands to millions of pixels together represents a scene.
- RGBD Camera: A depth camera on the other hand, has pixels which have a different numerical value associated with them, that number being the distance from the camera, or "depth." Some depth cameras have both an RGB and a depth system, which can give pixels with all four values or RGBD (Intel, 2019).
- Stereo Camera : This type camera works similar as human vision. It uses two or more camera to perceive depth and 3D sense of the environment. Stereo cameras can be used as substitute of costly LiDAR sensors (Appiah & Bandaru, 2016)

Pros of Camera sensor

- Provides accurate visuals of the environmental.
- Low cost in terms of sensors and infrastructure.

Cons

- Cameras do not give absolute distance sense of the object.
- Cameras also perform poor in bad weather conditions(fog , rain , night-time, sandstorm etc.).

III. IMU: IMU's are important as cameras and LiDAR's when it comes to autonomous vehicles operating reliably. IMU or Inertial Measurement Unit is a sensor which keeps track of the vehicle in 6 DOF. Usually IMU's are stand-alone sensors without any knowledge of the external world , this feature makes IMU a core technology for both safety and sensor-fusion. A 6 DoF IMU consists of 3 Axis accelerometer and 3 Axis Gyroscope. Magnetometer is usually not used due to local magnetic interference in the vehicle. IMU's are also suitable for short duration trajectory planning when GPS navigation goes offline.

Pros of IMU sensor

- Low cost sensors with high accuracy.
- Can be easily integrated with LiDAR , Cameras for frame tracking.

Cons

- Suffers from accumulated drift error.

IV. GPS/GNSS: GPS/GNSS sensors provide autonomous geo-spatial positioning with global coverage. GPS outputs are location (latitude, longitude and elevation). GPS is widely used in automotive industry as a driver assistance feature for navigation. GPS is often coupled with IMU for better localization and navigation.

Pros of GPS sensor

- Worldwide coverage , Free to use.
- Low cost sensors.
- With proper coverage and receiver setup centimetre level accuracy is possible.

Cons

- Poor performance when satellite visibility is poor (inside tunnels , city streets surrounded by buildings etc.).

1.18 System Architecture

The Detailed system architecture is shown in below illustration

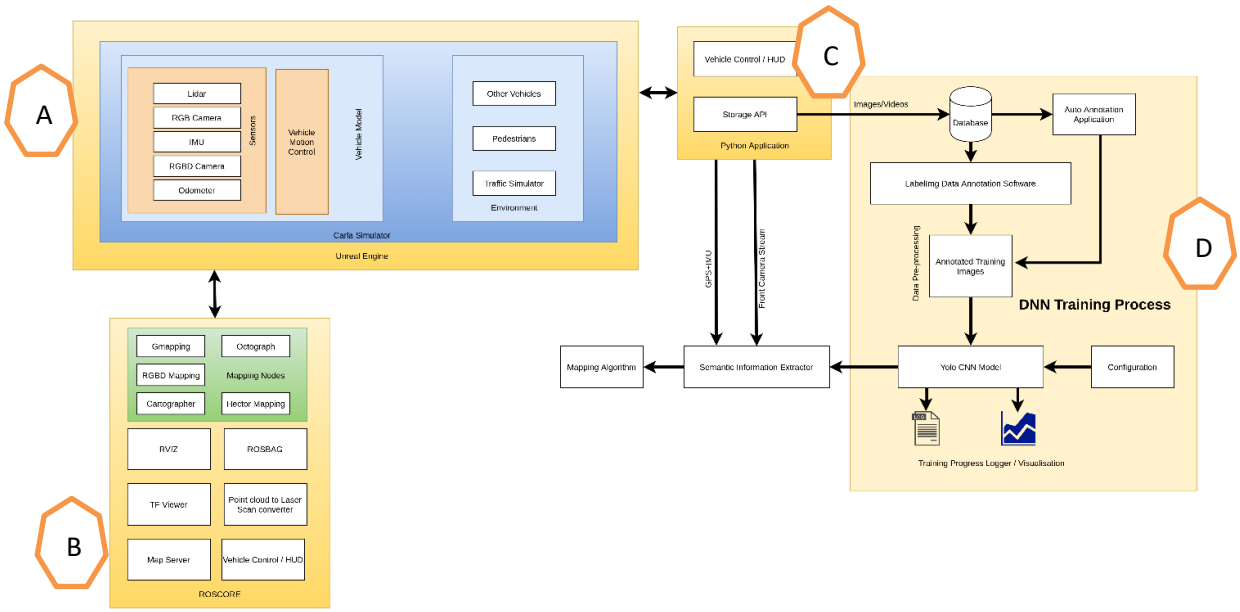


Figure 20 Overall System Architecture

Carla Simulator(A): CARLA has been developed from the ground up to support development, training, and validation of autonomous driving systems (Carla, 2019). The simulation platform supports flexible specifications of sensor suits and environmental conditions. Project specific Carla simulator configurations are mentioned as below

- 9 Cameras ,One Depth Camera
- One 360deg LiDAR @320000 points/sec , 28.8degree FOV , 32 channels
- One 6 axis IMU
- Vehicle Tesla Model3
- Server frame rate=5fps
- Vulkan API enabled.

PS: Below open source driving simulator comparison table shows that CARLA provides the best sensor suite to experiment with , hence CARLA simulator was selected .

Simulator Name	Engine	Supported Sensors	Preferred programming Language	ROS Support	OS Support
AirSim	Unreal Engine	IMU,Magnetometer,GPS, Barometer, LiDAR, Camera, Distance sensor	Python , C++	Yes	Ubuntu , Windows
Carla	Unreal Engine	IMU, GPS,LiDAR,RADAR, Camera, Depth Camera, Lane sensor, Semantic Segmentation	Python	Yes	Ubuntu , Windows

Table 7 Simulator Pugh Matrix

ROS(B): ROS provides a great learning platform for SLAM algorithms. One of the project objectives is to implement different sensor-based SLAM algorithms in CARLA and compare their results . CARLA simulator provides a great ROS bridge API through which the ROS nodes communicate with CARLA simulator. However, Carla simulator ROS bridge does not have native ROS SLAM nodes implemented , hence the below nodes were configured and implemented in CARLA-ROS bridge.

- PCL to Laser scan node (3D LiDAR to 2D LiDAR).
- Odometry node for RTAB mapping.
- Gmapping.
- Hector Mapping.
- Octograph.
- RTAB Mapping.
- Cartographer(WIP).
- Carla low speed automatic control node for RTAB.

Carla-Python Client(C): This could have been implemented inside ROS environment , however ROS melodic does not support Python3.6 and the ML environment is based on Python3.6. Moreover, Carla-ROS bridge has latency issues during teleoperate control. Hence to reduce system overhead , the Carla-Python data acquisition interface is purely built outside of the ROS environment. The Data acquisition interface is designed as per the experiment requirement. The Carla-Python subsystems are mentioned below

- HUD aka Heads up display is the HCI to the CARLA simulator , the main design is adopted from the standard implementation by CARLA developers , which is built on top of Pygame.
- Data I/O: The keyboard is used to control the vehicle motion ; A custom data recorder is implemented to record the images from front and rear cameras. The native image recorder skips frame while recording, which gets worse when the vehicle speed increases. To avoid this , RAM based buffer is implemented which holds the images in the primary memory.

DNN Training process(D): At present the training pipeline is not fully automated , the DNN based image classifier training needs operator intervention. Though there are automated annotation tools like CADET available (Brekke, et al., 2019)but CADET is not compatible with Carla 0.9.7. Hence for the project a YOLOV3-KITTI based automatic annotation is tool is designed and implemented. The first prototype was annotated using opensource annotation tool named as LabelIMG (Tzutalin, 2015). The automatic annotation tool was not built specifically for CARLA environment hence the detector model showed poor performance; hence few confidence-based rules were implemented to improve annotation (trial and error method).

12.Implementation

The project scope is subdivided into two parts

1) PS1 or Implementation of Standard SLAM Algorithms.
2) PS2 or Implementation of DNN based Visual-SLAM Algorithm.

The common tool is CARLA simulator. Which runs on top of Unreal Engine container. The current version of CARLA supports Unreal Engine 4.22 , which is built from source with GPU support to take advantage of hardware accelerated simulation (Carla Installation, 2019)

Challenges met during Unreal Engine build/launch

I. Takes time(approx 4hours) to compile from source for the first time.
II. If GPU support enabled , the vulkan drivers must be installed or else after 4 hours of compilation the launcher will crash.
III. "Engine crash handling finished; re-raising signal 11 for the default handler. Good bye. Segmentation fault (core dumped)**.** The issue was identified to be Nvidia Vulkan driver related issue by analysing error log. This can be avoided by forcing to opengl only mode ,however the UE4 Editor and UE4 Engine rendering performance drastically reduces and the simulation becomes slow. The issue was resolved by reloading vulkan sdk by "vulkaninfo" command in the shell before launching the UE4 Editor or UE4 Engine.
IV. Unresponsive Unreal Engine was observed due to CPU throttling during shader building (Texture compilation). After loading the Carla project first time the models , textures must be compiled , which takes sometime hence the UE4 editor may become unresponsive for couple of minutes during the texture compilation phase.

Once Unreal Engine is built CARLA source is compiled against the Unreal Engine 4.22 binaries. There are two option to run Carla Simulator

Using Unreal Editor: Carla project can be loaded in UE4-Editor and then played in the editor itself.

Figure 21 Unreal Editor

Using Compiled Carla Simulator: Carla project can be compiled to launch without needing to launch the Unreal Engine Editor. This method is much effective as the Unreal Engine 4 editor sub-modules need not to be loaded in order to perform experiments. For most of the experiments the stand-alone simulator is used.

Figure 22 Standalone Carla Launcher

Few important points to remember while performing experiments in CARLA are

- Carla works as a client and server model. By default, the listening ports are 2000, 2001 and 2002 . If one is running CARLA simulator in an external system , make sure to add the ports to firewalls outbound connection.
- If the Client(Python 3.6) throws error while importing Carla module , then append the Carla*.egg file path to python system. Also, the *.egg files are different for python2.7 (used for ROS bridge) and python 3.6 (stand-alone client) , so before launching any python client the respective *.egg file must be appended in the path. Use of virtual environment is encouraged. The egg files can be found under the CARLA dist/**/PythonAPI/ folder.
- While exiting if the CARLA client does not remove the actors from the Unreal Engine server instance, the actors remain in the server. Launching the client again will spawn more clients in the server instance and the server performance will become slow. Hence its recommended to destroy all the actor models from the server before exiting the client.
- Multiple clients can be launched at the same time.

1.19 Implementation of standard SLAM Algorithms

ROS-Carla bridge from (ros-bridge, 2019) provides a good starting point to explore CARLA simulator using existing ROS nodes and tools. The ROS client architecture is shown below. Five Slam algorithms were implemented and tested in the ROS environment using different sensor suite. The different SLAM nodes implementation is explained below

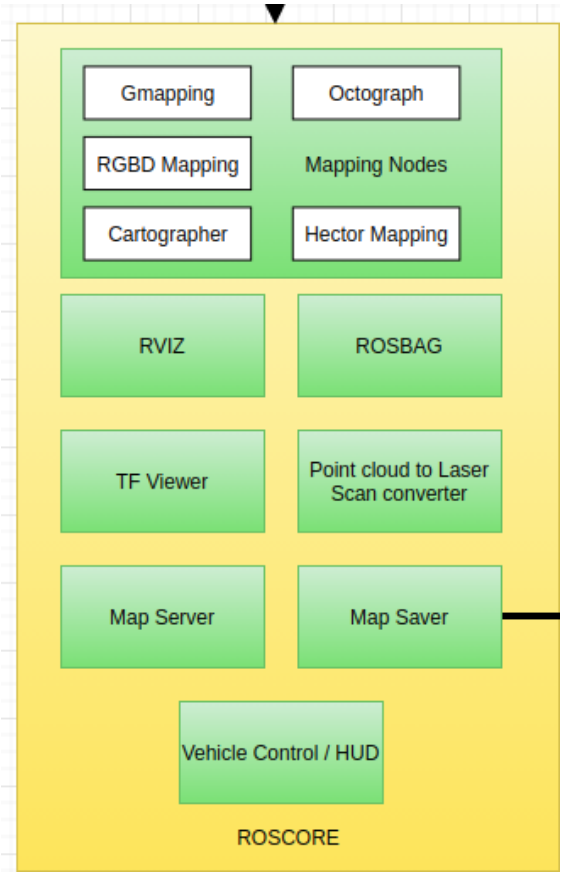


Figure 23 ROS Nodes

Essential Nodes/Tools

a) **Carla-ROS client:** This is a custom python2.7 CARLA client implementation(based on carla-ros bridge), the client has configurable sensor suite. Sensors include

- 3D lidar , 32channel , 32000 points, vertical 27deg view, horizontal 360* view.
- 1 Rear camera(RGB) , frame size 1280x720, 120* FOV.
- 1 Front camera(RGB) , frame size 1280x720, 120* FOV.
- 1 View camera(RGB) , frame size 1280x720, 90* FOV.
- 1 depth camera(RGBD) , frame size 1280x720, 120* FOV.
- 1 GPS sensor.
- 1 Odometer.
- 1 IMU , 6 axes.

Sensors.json file can be edited to configure the sensor suite for experiment. The client has a view Camera which is as a HUD unit which acts as the teleoperator view to the server simulation, shown in below image

Figure 24 Carla Simulator HUD

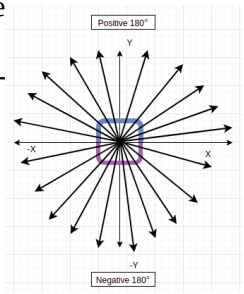
The node can be launched using roslaunch command

I. Modification : The default client is modified to facilitate below features
II. Vehicle filter: If set this set, then it will spawn a specific vehicle for experimentation consistency.
III. Vehicle Actors: The ROS parameter if set NumberofVehicles will spawn given number(integer) of vehicles in the CARLA server.
IV. Pedestrian Actors: The ROS parameter if set NumberofPedestrians will spawn given number(integer) of pedestrians in the CARLA server.
V. Weather configuration: The ROS parameter if set weather(tuple of Cloudiness, Precipitation , SunAltitudeAngle) will control the simulator weather accordingly. Though the weather is not changed during the project , this will be useful for future development and testing.
VI. Spawn location: It's possible to spawn always at a same location using the CARLA client if spawn_point is set in the launch config.
VII. 2D LiDAR: The point cloud to laser scan node is integrated in the client so when the client runs it will automatically enable the 2D LiDAR.
VIII. RVIZ: There are two client launcher configurations to launch the client with RVIZ and all sensor suit activated or without RVIZ , during most of the test the without Rviz client is launched as it is much memory saving than with RVIZ.

b) **CarlaPCLtoLS**: This is a custom node built on open-source project (Bovbel, 2019) point cloud to laser scan(3d LiDAR to 2d LiDAR) . The node is written in cpp and the launch file is written in xml. By default, CARLA does not support 2d LiDAR , hence the above node was implemented

specifically to support SLAM nodes. The below parameters were tuned specific to CARLA simulator for 360degree coverage ,

- angle_max=3.14 (180-degree coverage)
- angle_min=-3.14 (-180-degree coverage)
- angle_increment=0.0087(less this value more laser scan points , more precision but more processing demand).
- The number of laser scan points are calculated as (angle_max-angle_min/angle_increment)



c) **Rviz**: Rviz is used throughout the project for data visualization and analysis. By default, the RVIZ config appears as below. The below configuration features 8 camera surrounding 360deg , a 2D LiDAR sensor and a 3D LiDAR sensor

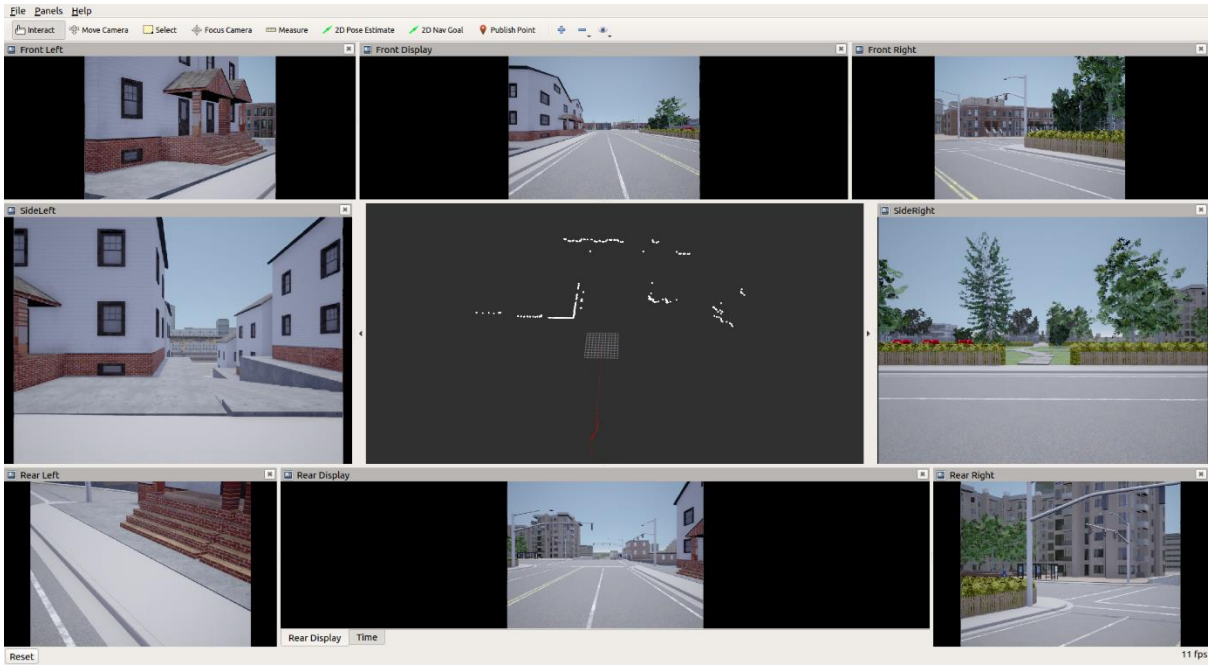


Figure 25 Rviz

Rviz is configured as per different slam configuration as different sensors were used.

d) **Rtabmapviz**: Rtabmapviz is only used for Rtab mapping, as the visualization tool gives more flexibility of RTAB mapping parameter tuning.

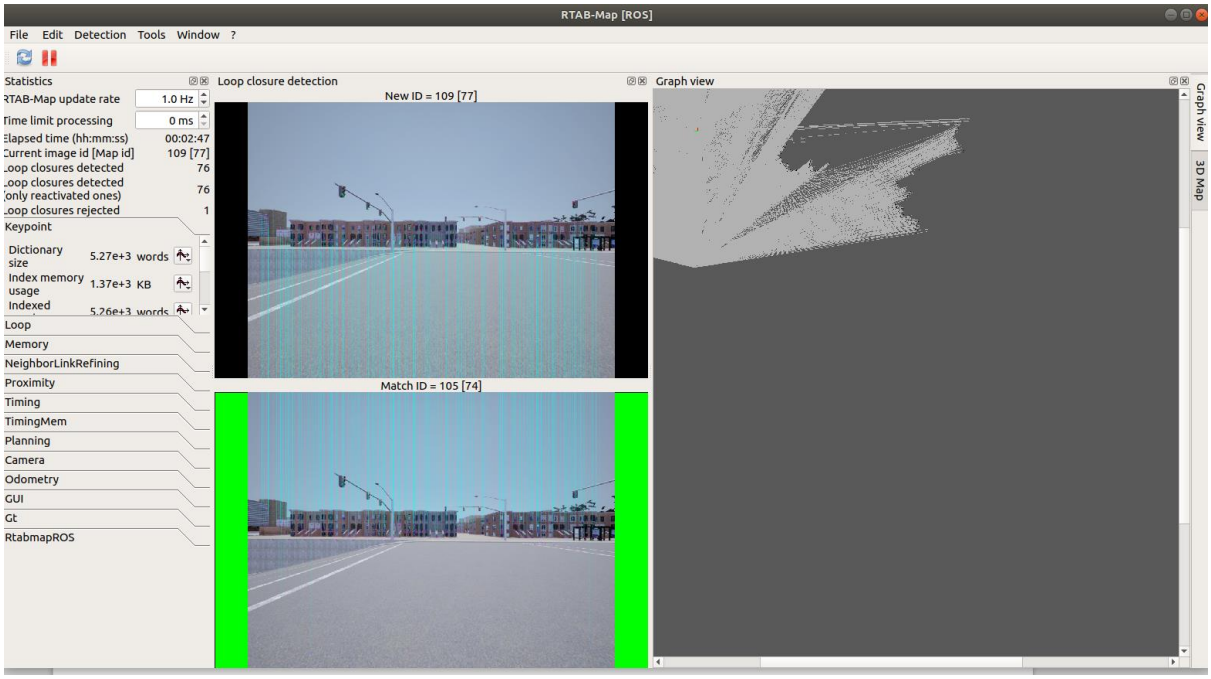


Figure 26 Rtabmapviz

e) **Custom Odometry**: Most of the ROS slam packages publish map data over "map" frame . Co-incidently the same frame is used by Carla ROS bridge. Hence a custom odometry node is implemented to publish odometry as per the SLAM node requirement. (Node status Obsolete as all slams have built-in odom calculation).During the implementation of custom odometry node , the following challenges were encountered

I. **Time synchronization** : The new frame structure looks like below , where "base_link" mimics the "map" frame and "odom" frame is the odometry frame. The new transform works properly , however when the slam nodes were running the slam node will throw warning regarding time synchronization issue. The node subscribes to a "/carla/ego_vehicle/odometry" for the odometry information , found out I was assigning latest time in the "odom" header , so when the slam nodes look at sensor time frame and the odom timeframe they do not match . Which was quickly resolved by reusing the native odometry time frame.

II. **Transform Mismatch** : Carla-ROS bridge was publishing frames as below and the custom odom publisher was not a part of the Carla-ROS bridge TF frame. The issue is not fully resolved. A workaround solution was to modify the carla-ros bridge source code and remap the entire tf tree as per slam frame requirement.

III. **Processing time**: At the first implementation , the odom calculation was manually done observing the LiDAR frame displacement , which resulted in increased response time from the odom frame , because by the time a frame is being processed already multiple frames were being built up in the subscribed message queue. This was fixed by removing any manual calculation and reassigning CARLA-ROS bridge odometry result.

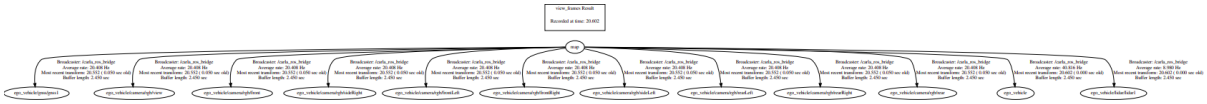


Figure 27 Carla Full Tf Tree

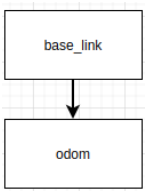


Figure 28 Custom odom tf tree

f) **Custom speed control**: This is a custom-built node that publishes low speed vehicle control command . This is specifically implemented for RTAB mapping and Cartographer.

Issues:

g) **Vehicle speed overshoot**: The first implementation used on/off control which used a fixed step throttle control due to which the vehicle speed used to overshoot sometimes. The issue was resolved by implementing proportional throttle control.

h) **MISC**

view_frames: This is a standard implementation of ROS tf bundle , this tool saves tf tree as pdf for further analysis . This tool is used for comparing existing gazebo-based slam algorithm tf frames vs my CARLA-ROS based slam tf frames.

tf_echo: this is also a standard implementation of ROS tf bundle. This tool prints transform data in the terminal , this was useful to check tf data validity(if values are 0.0 and vehicle is moving , something's gone wrong).

rosbag : standard implementation in ROS melodic , used for saving simulation data and replaying later.

rqt_graph: This is also a standard implementation of ROS tf bundle , this tool visualizes ros frames in real-time , very useful to view and analyse tf tree(parent-child relation) and publish rate.

rqt_plot: this is a graphing tool built into ROS , this tool is used to visualize control commands and imu data.

rostopic : used to check message topics, publish message , view message content etc. Other tools used for workspace management were catkin_make, catkin_create_pkg, roscd, rosfind,rosparam.

1.19.1 2D Gmapping SLAM

The Gmapping node is based on standard ROS gmapping node (Gerkey, 2019).The base configuration is modified as below to support gmapping in Carla Simulator

The node relies on below sensor interface

I. 2d laser scan from CarlaPCtoLS node
II. base_frame that is “map”

The map output frame is map2, which is subscribed in RVIZ to get map while slam is running. The calculated odometry by Gmapping node is published in odom frame. The Gmapping transform tree is shown below

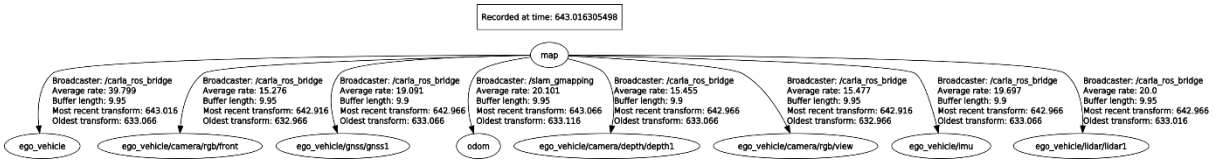


Figure 29 Gmapping tf tree

Status of Node Implementation : Implemented and working.

1.19.1.1 Challenges

I. 2D LiDAR: As mentioned earlier , the Gmapping expects a 2D LiDAR data for grid mapping, but CARLA simulator does not have 2D LiDAR implementation, hence the CarlaPCtoLS node was implemented to facilitate 2D laser scan data for Gmapping implementation in Carla Simulator.
II. TF Frames: At the starting of the project , it was challenging to understand the slam algorithms TF requirement , TF tree hierarchy . To clarify tf frames and their usage in slam , the gmapping implementation in turtlebot-Gazebo simulator is analysed and mimicked in CARLA environment.
III. Incorrect base_frame: Odometry is calculated based on reference(in this case 2d LiDAR) frame , at the first implementation a wrong frame was referred due to which after sometime the odometry will thow error and gmapping enventually does not generate any map(Empty grid map). After analysing the tf frames and their relation with the base frame “map” , this was resolved by passing the 2D LiDAR sensor tf frame as odom frame to track. Which resulted in correct map.

1.19.1.2 Issues

I. High memory usage : The Carla simulator environment is large compared to usual gazebo simulated environment , which demands high computation during mapping. The issue worsens as the vehicle speed increases and map size gets bigger. Usually the map will be, the issue is resolved by reducing the scan points in the lidar and moving the vehicle relatively slowly during the mapping process.

1.19.2 Hector mapping

As gmapping and hector mapping belong to same slam(2d) , it was easy to integrate hector mapping after gmapping implementation (Kohlbrecher, 2019). It relies on below sensor suite

I. 2d laser scan from CarlaPCtoLS node

Hector mapping important configuration parameters

I. map_size(8096) : the size of the square map , static and cannot be changed during runtime.
II. map_resolution(0.050m): the length of a grid cell edge in meters.
III. map_update_distance_thresh(0.2m): After moving this much distance the mapping algorithm will update the grid map.
IV. map_update_angle_thresh(0.06rad): After rotating set angular change the map will be updated.
V. scan_subscriber_queue_size(100): Depends on 2D LiDAR output rate , faster the LiDAR higher the subscriber queue buffer or the scan data will be rejected.

Status of Node Implementation : Implemented and working. Hector mapping tf frames are shown below

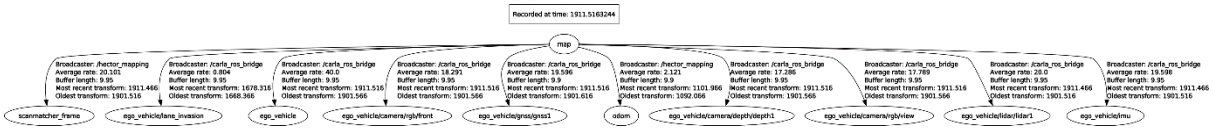


Figure 30 Hector map Tf tree

1.19.2.1 Challenges

I. **Constant map size**: The map size in hector mapping is constant and predefined contrary to gmapping. The mapping starts from the middle of the given map size . After few trial and error with map_size parameter an acceptable mapsize was achieved for carla simulator environment.
II. Map Resolution: With higher map resolution the size of the map becomes larger and in turn requires more processing bandwidth.

1.19.2.2 Issues

I. Overlapping maps: When the vehicle is increased the maps, boundary were overlapping with one another. A workaround solution was to limit the vehicle speed during mapping.

1.19.3 Octomap

Octomap by (Wurm, et al., 2012) implements 3d Occupancy(voxel) grid using 3D LiDAR sensor, it is a standard ROS supported slam implementation. The sensor suite used for octomap are

I. 3d laser scan from /carla/ego_vehicle/lidar/lidar1/point_cloud topic
II. Base_frame (in this case “map”)
III. resolution: 0.1
IV. sensor_model/max_range: 100.0

By tweaking the depth parameter, the map resolution can be improved , but this demands more processing , low mapping speed, increased size of occupancy grid map. The map output is published over /octomap_full as voxel grid occupancy map. Which can be later used for localization.

Status of Node Implementation : Implemented and working.

1.19.3.1 Challenges

I. Octomap creates 3d point cloud map which cannot be extracted map_server tool . It needs a native tool octomap_server to save voxel cloud map.

1.19.4 RTAB Mapping

RTAB or Real-time appearance-based mapping is a RGBD SLAM approach based on global loop detector with real-time constraints. It can create 2d/3d map of an environment. Which can be used for navigation and localization. For this project the standard RTAB mapping implementation is tweaked for CARLA simulator. Rtabmapviz is used for this slam instead of RVIZ as the Rtabmapviz provides as detailed view of the slam process and eases the paramter tuning in real-time. The sensor suite used for RTAB mapping are

I. RGB Camera topic → /carla/ego_vehicle/camera/rgb/front/image_color
II. Depth Camera → /carla/ego_vehicle/camera/depth/depth1/image_depth
III. 3d LiDAR → /carla/ego_vehicle/lidar/lidar1/point_cloud
IV. 2d LiDAR → /camera/scan
V. visual odometry calculation → true(the native odometry showed error hence the visual odometry is enabled).
VI. base_link-→ map
VII. approx_synch-→ true (As timestamps of the input topics are not synchronized)
VIII. map publish frame map2

Status of Node Implementation : Implemented and working. RTAB map tf tree is shown below.

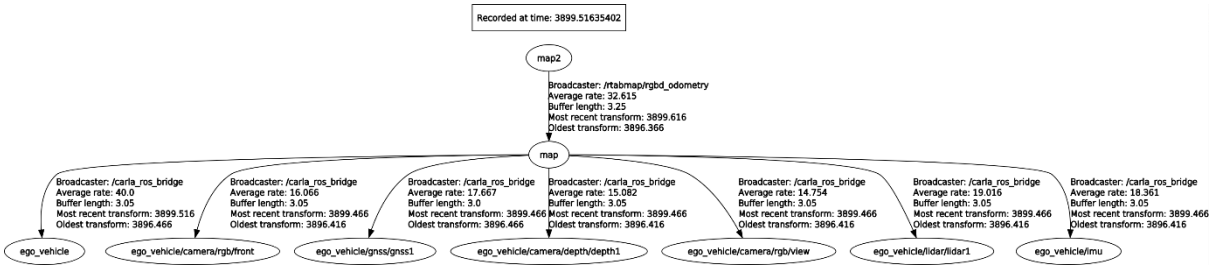


Figure 31 RTAB Map tf tree

1.19.4.1 Challenges

During RTAB mapping below challenges were encountered

I. No Loop Closure: The Loop closure was being rejected due to high vehicle speed mapping. When the vehicle speed was reduced , the loop closures were being flagged properly.If the Carla_ROS bridge odometry is provided for SLAM then the RTAB map does not detect loop closure , if visual odometry is enabled then loop closure works just fine.
II. Map publishing: By default, the map was being published in under frame “map” which was coincidently the base_link of the Carla-ROS bridge. At first it was flagged as odom frame fault , hence a stand-alone odometry publisher was implemented to test the slam . Which did not help , after properly analysing the tf tree and the RTAB map tf requirement the map publish frame was renamed to map2 and the RTAB map generated the map properly.

1.19.4.2 Issues

I. The RTAB mapping does not work in Carla when the vehicle speed is higher than 5kmph. A workaround solution was to implement a low speed vehicle speed limiter(3kmph) for mapping.

1.19.5 Cartographer

Cartographer provides Realtime SLAM implementation in 2D/3D LiDAR sensor configuration. It uses sensor fusion to correct gravity/speed variables during slam by using IMU/GPS sensors. The sensor suit/configuration used for Cartographer is explained below.

I. sensor 1 3D lidar , 1 6DoF IMU, 1 gps sensor .
II. map_frame= map2(by default map is used as base_link by carla).
III. tracking_frame= base link of the vehicle being tracked , which is “map”.
IV. provide_odom_frame= if odom is provided or calculated by the slam. In the implementation it is set to true.
V. published_frame= odom_frame is if odom is provided , in the implementation it is ego_vehicle.
VI. odom_frame= if provide_odom_frame is set to true the frame is provided here. In our case the carla/ego_vehicle/odometry.
VII. use_nav_sat=true (gps is provided for sensor fusion).
VIII. num_point_clouds=1 (1 LiDAR channel)

Status of Node Implementation : Non-Functional node.

1.19.5.1 Issues

I. Cartographer implementation in CARLA environment is not successful due to mismatch in the transform tree and sensor data synchronization. Further development and test is required to implement cartographer.

1.20 DNN based Visual-SLAM implementation

The DNN based Visual-SLAM or PS2 is divided into two phases

- Implementation of Object Detector
- GPS and IMU sensor fusion

1.20.1 Carla Object Detector

The Carla Object detector implementation flowchart is shown below

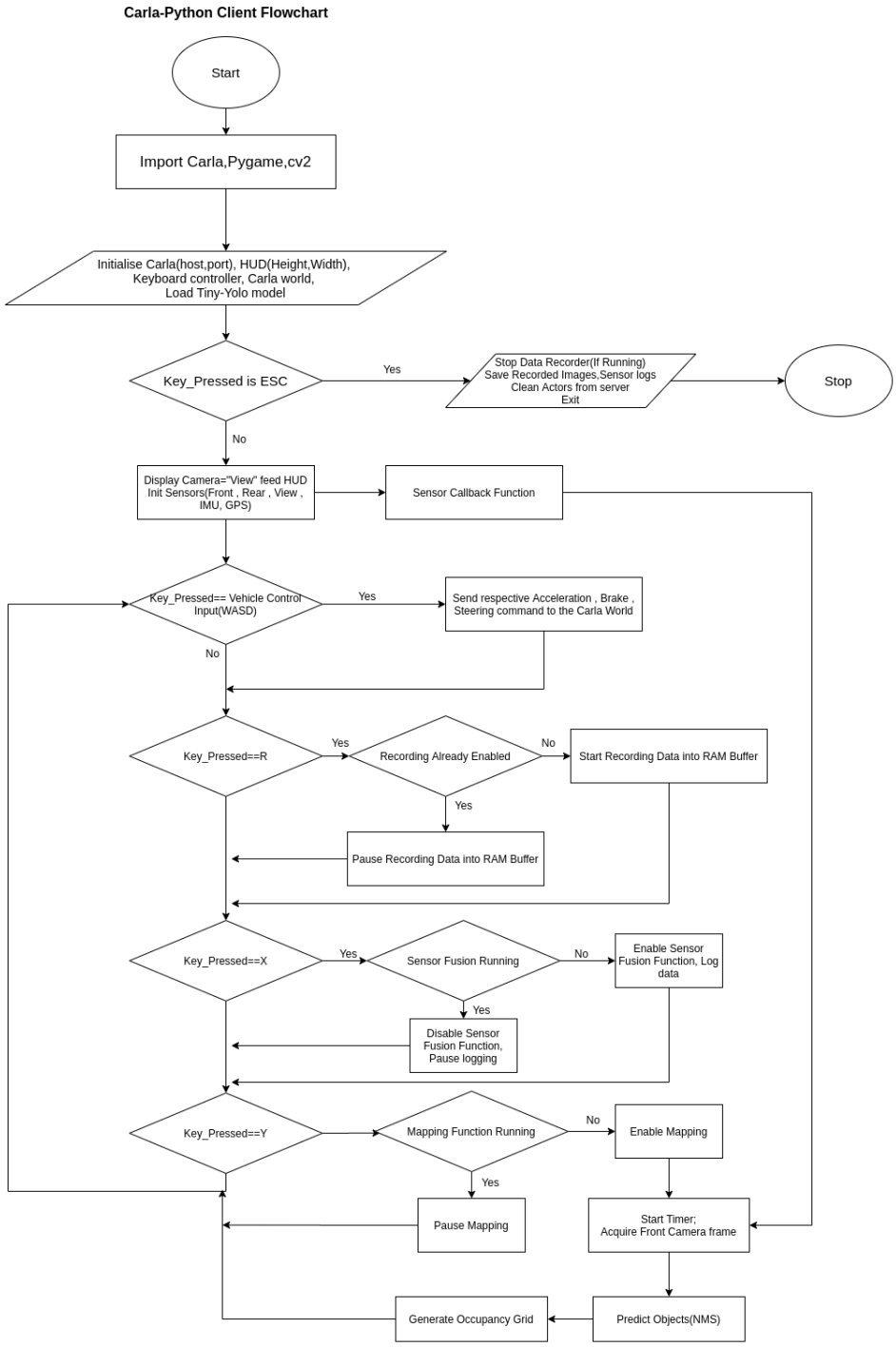


Figure 32 Carla-Python Client Flowchart

1.20.1.1 Carla Yolo training

To annotate the data an automated tool is developed based on Yolov3 trained on coco dataset. Which is used to label ~5000 training images . The flowchart is shown below.

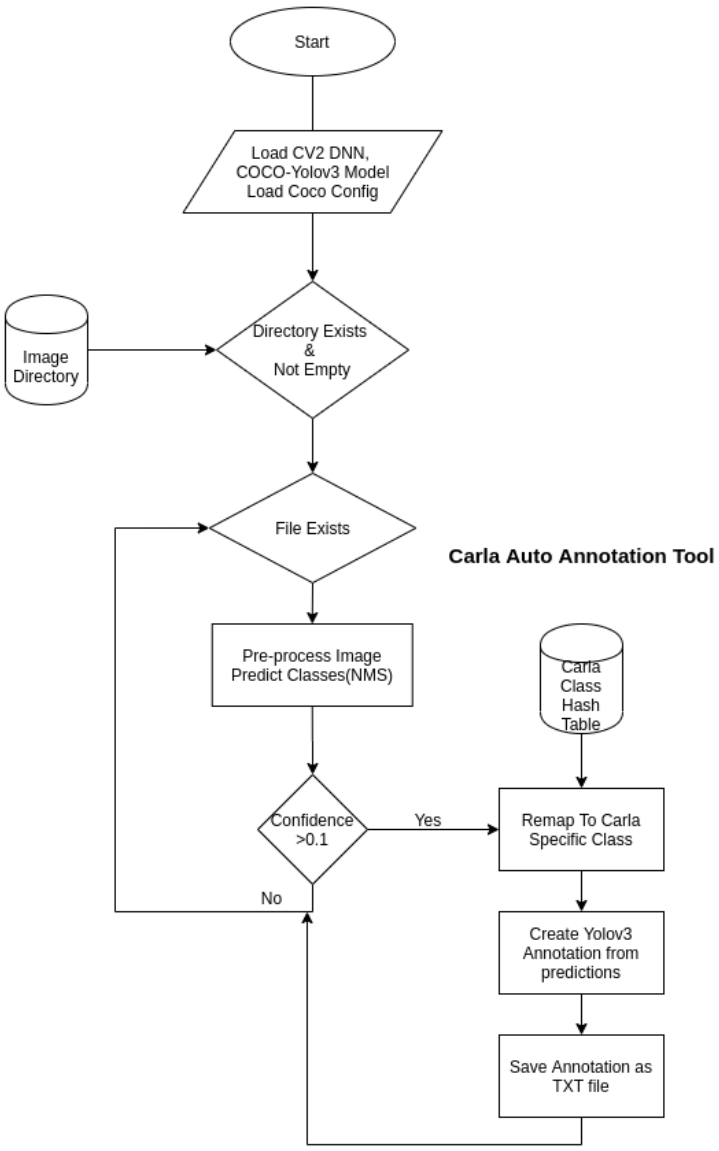


Figure 33 Carla Annotation Tool

Yolov3 training flow chart is shown below

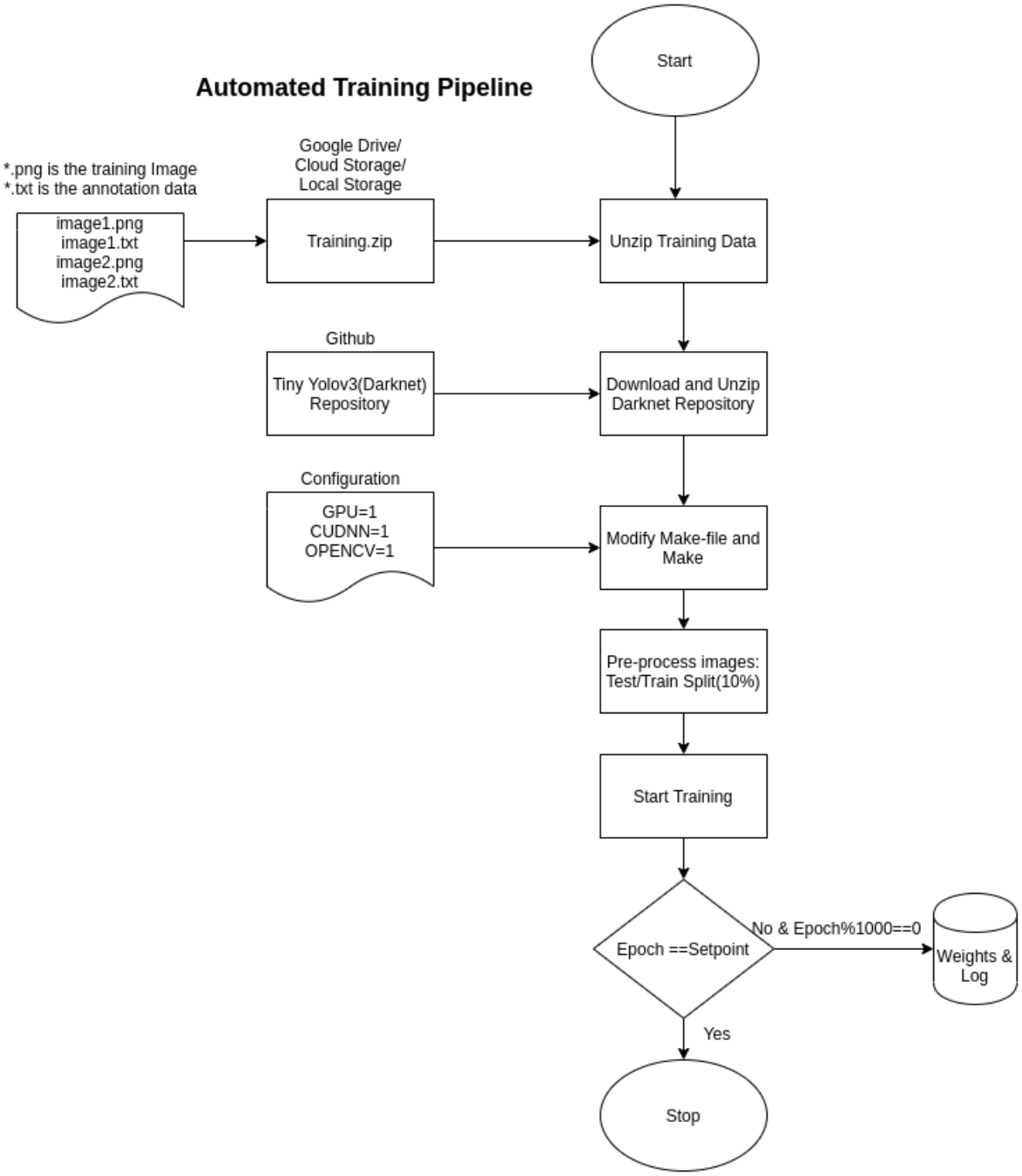


Figure 34 Carla Training Flowchart

1.20.2 GPS & IMU Sensor Fusion

In real-world (urban city streets, underground tunnels , forest etc) GPS do not provide good coverage and the frequency of GPS data output frequency is also limited, hence only GPS cannot be used for reliable localization and navigation solution. However, if the GPS sensor is assisted with high frequency precision IMU sensor , then the combined solution can tackle almost all cons of both the sensors working individually. In this phase of the project a GPS (Location sensor) and a accelerometer sensor data is fused together to achieve better localization in CARLA simulator. The latest release of CARLA simulator(0.9.7) supports noise injection in the IMU unit . The noise margin is used to test the filter performance .

The Filter algorithm is based on Section 4.1 , Extended Kalman Filter. The filter algorithm is explained below.

Step 1 :State prediction , $(x_k) = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix}$, motion equation at k+1 , $x_{k+1} = A.\ x_k + B\ .\ u$

Step 2: Measurement , Measurement matrix H is initialized to cover the position and acceleration vectors

Step3: Measurement Nosie Covariance , the measurement will have certain error based on sensor's standard deviation , which is considered during the measurement by the noise covariance matrix R

Step 4: Process noise covariance calculation , the measurement will have certain deviation based on environmental effect , electrical noise etc. The process noise covariance matrix Q accounts the overall process error.

Step 5: Update estimate , the location sensor(GPS) data is used to cross validate the accelerometer-based prediction and measurement error covariance is updated for the next iteration.

1.20.3 Challenges

I. Data Storage: While Saving images using native image_save function the simulator skips frame which results inconsistent sensor data while the image is being saved. The issue was resolved by implementing a temporary buffer in the ram to store the images being recorded. When the simulator is stopped , the buffer is cleared after the images are saved in the secondary storage.

II. Tiny-Yolov3 integration: The detector takes ~17 ms to detect objects in a current frame , during the detection the thread pauses and waits for the detector to finish

III. GPU memory constraint : The host PC has 4gb of GPU memory which resulted in very slow training with a batch size of 2. “GPU:CUDA Error: out of memory” will be usually thrown when the batch size is >=3, so at a time only 2 images can be processed. To reduce the training time , free to use Colab GPU was used , however it can only train till 10000 iteration(12 hours of free session). The final model was trained in GCloud with Tesla p4 GPU which can process upto 4 images at once and it took ~ 1.2 days to finish 30000 iterations.

IV. Yolov3 not predicting: Initially yolov3 was not predicting any classes. After couple of iterations , it was concluded that the model is not trained with adequate amount of data. Hence a dataset of 5500 images was prepared distributed among 4 classes. After training with the dataset, the detector worked properly.

13.Testing

The standard SLAM implementations tested based on the map quality, CPU usage , Memory usage(Resource) , vehicle speed.

1.21 Gmapping

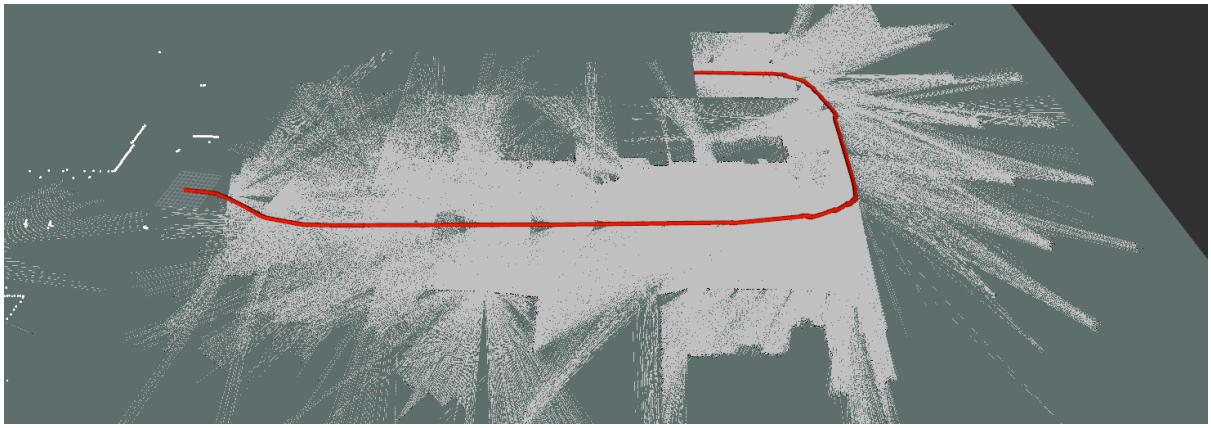


Figure 35 Gmapping Route

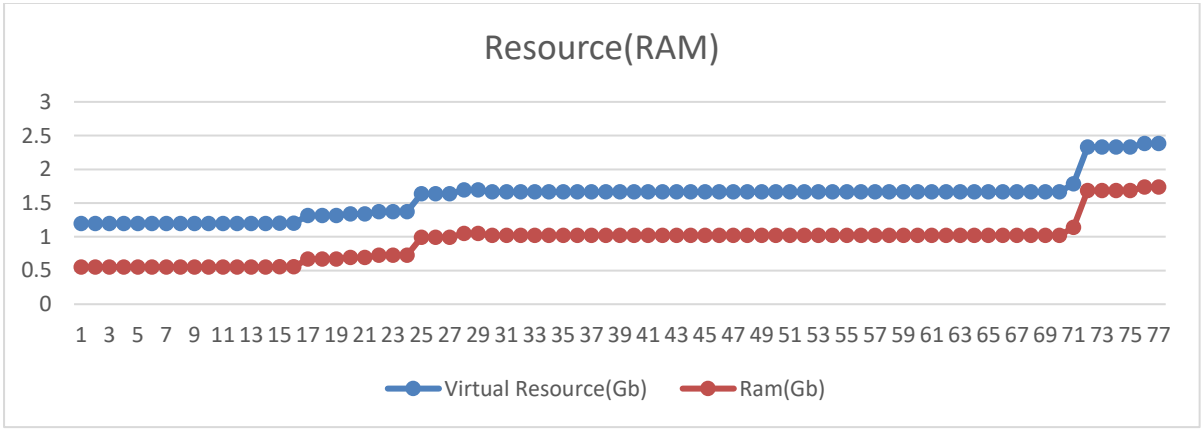


Figure 36 Gmapping Memory Usage

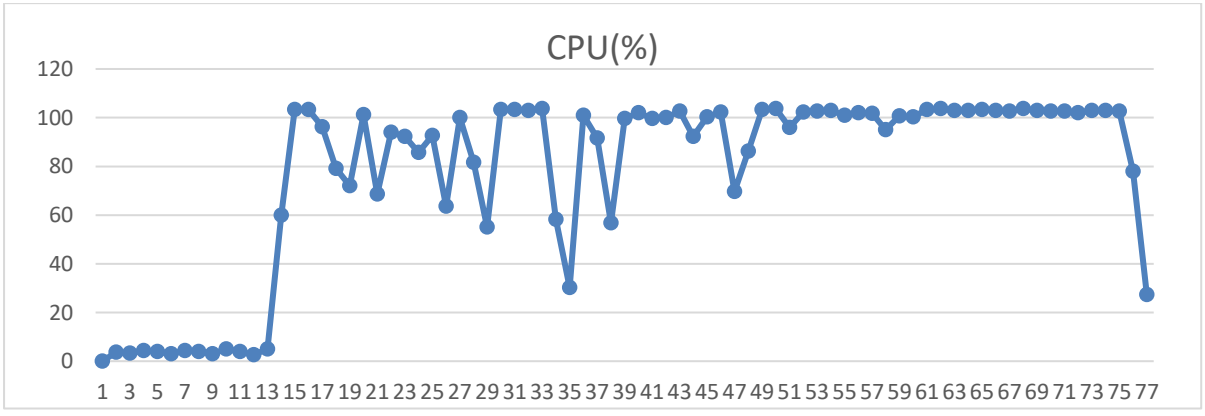


Figure 37 Gmapping CPU Usage

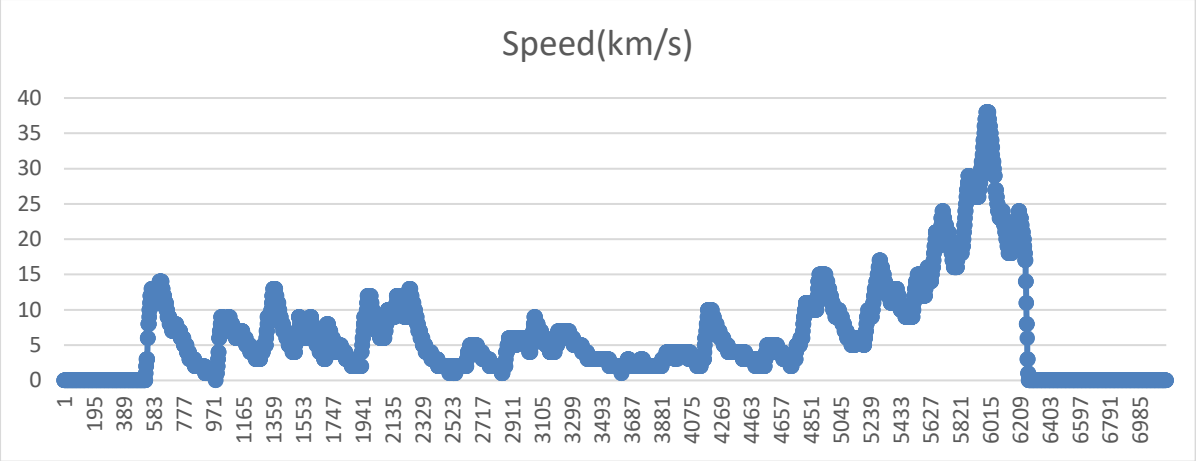


Figure 38 Gmapping Vehicle Speed Trend

1.22 Hector map

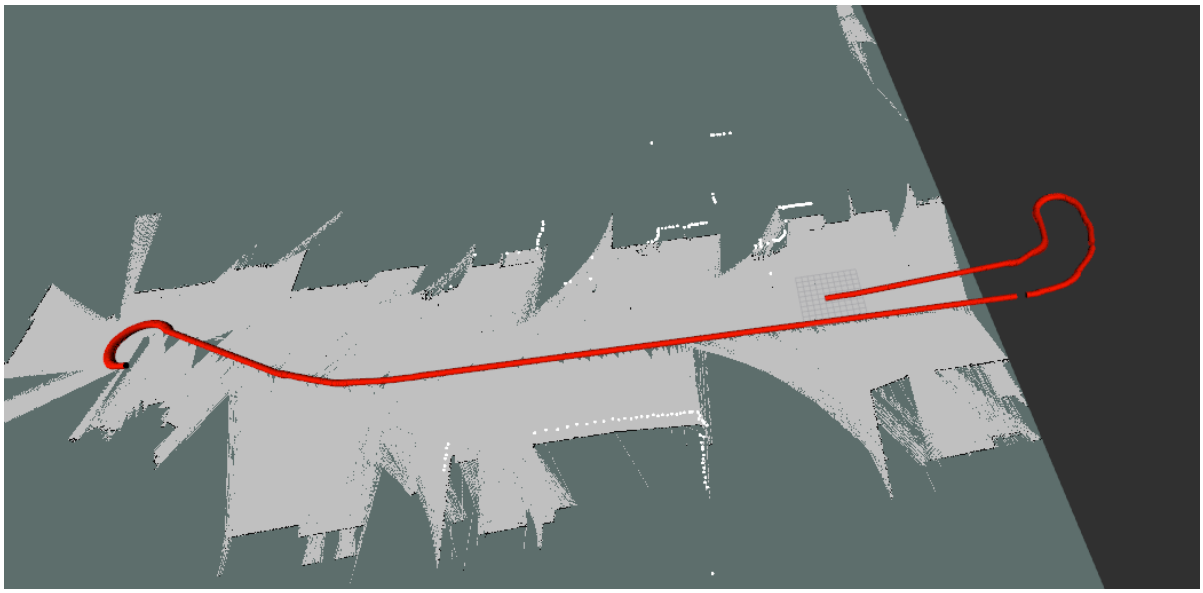


Figure 39 Hector map route

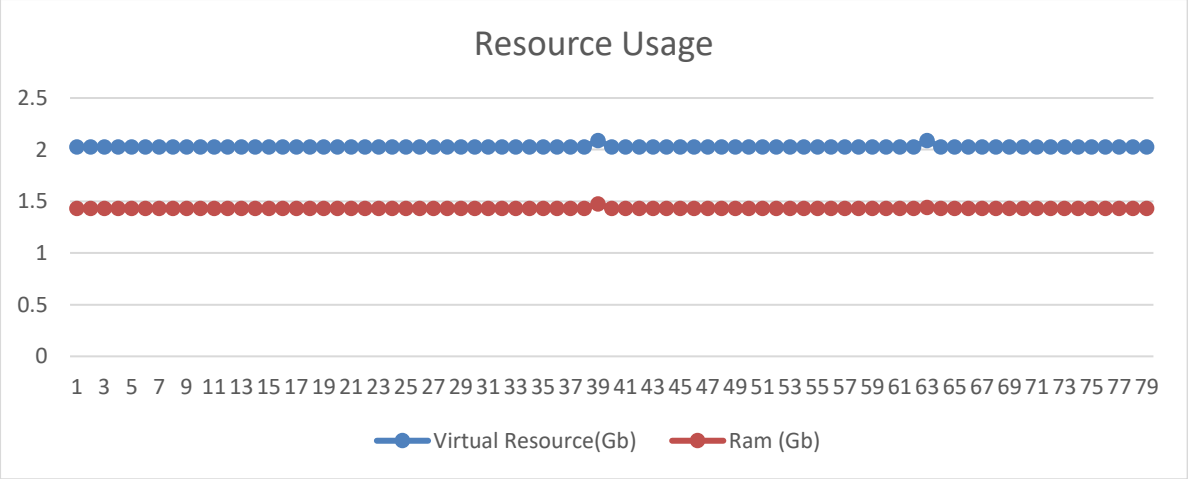


Figure 40 Hectormap Memory Usage

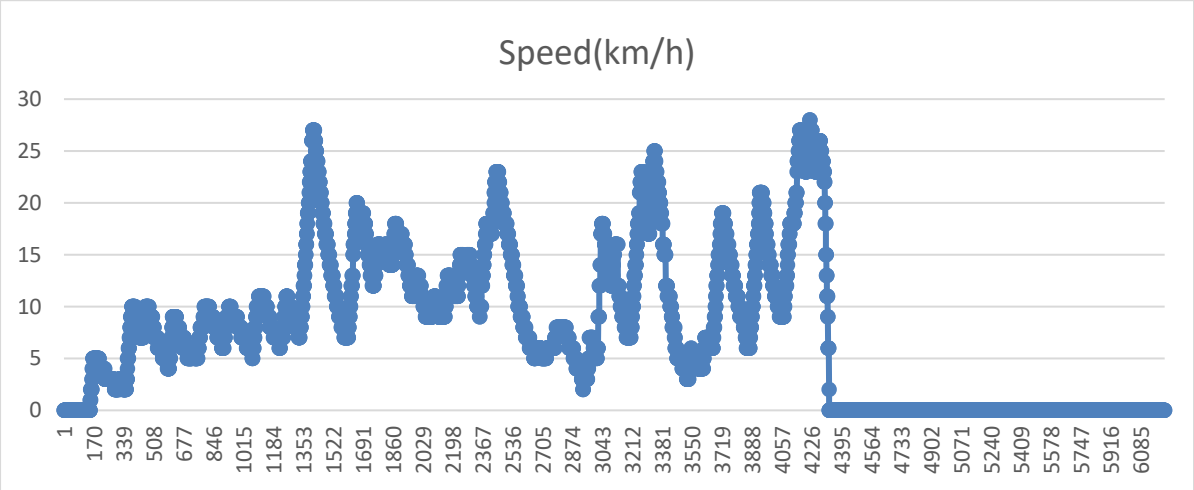


Figure 41 Hector map Vehicle Speed trend

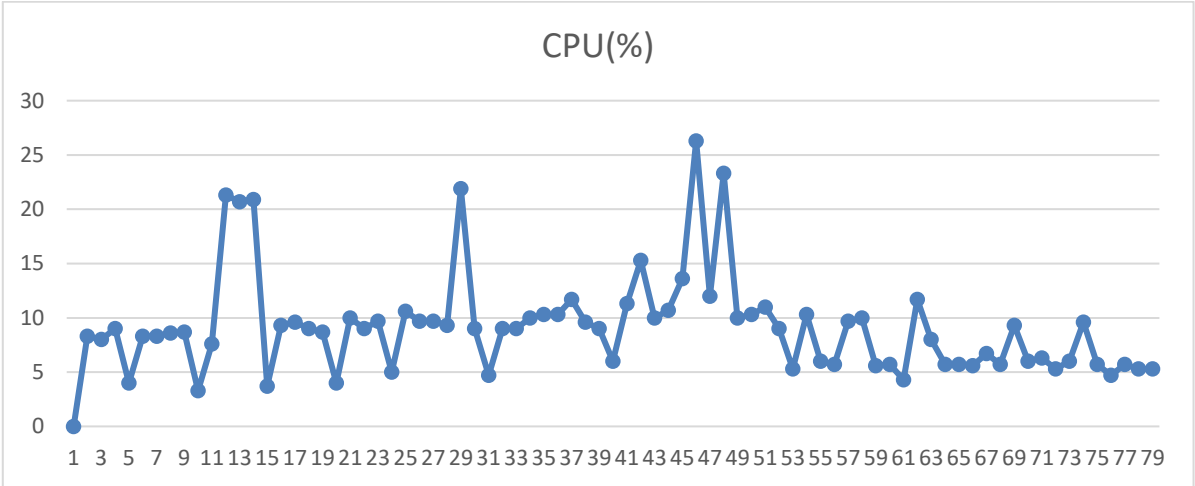


Figure 42 Hectormap CPU Usage

1.23 Octomap

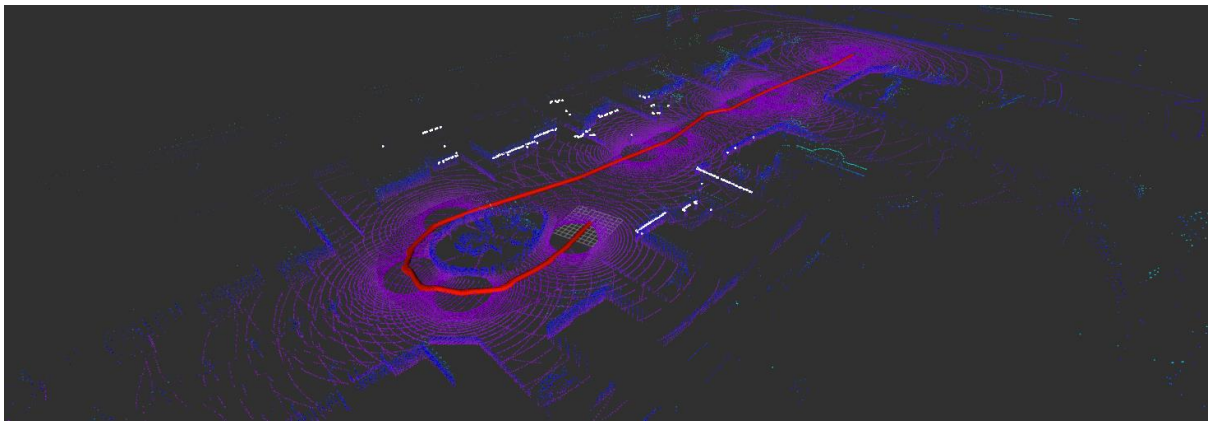


Figure 43 Octomap Voxel View 1

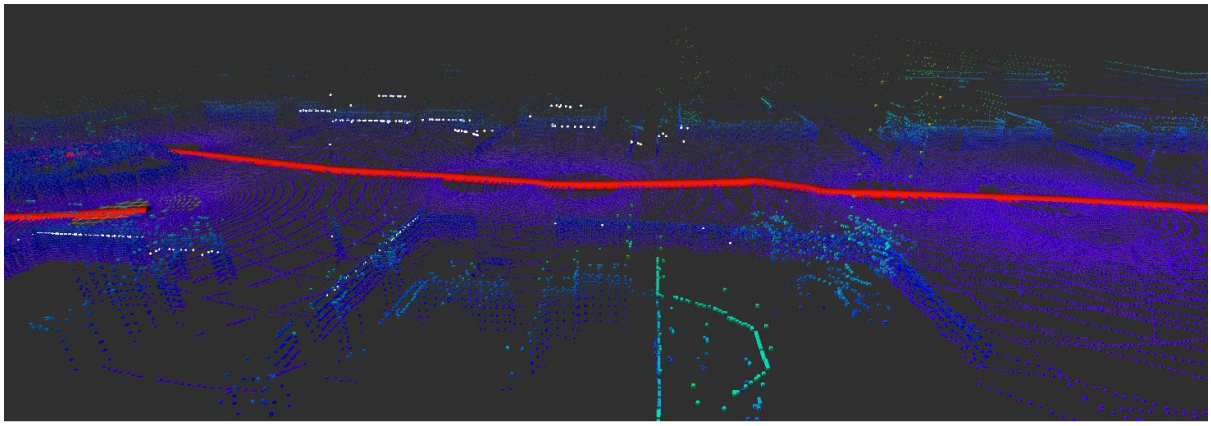


Figure 44 Octomap Voxel View 2

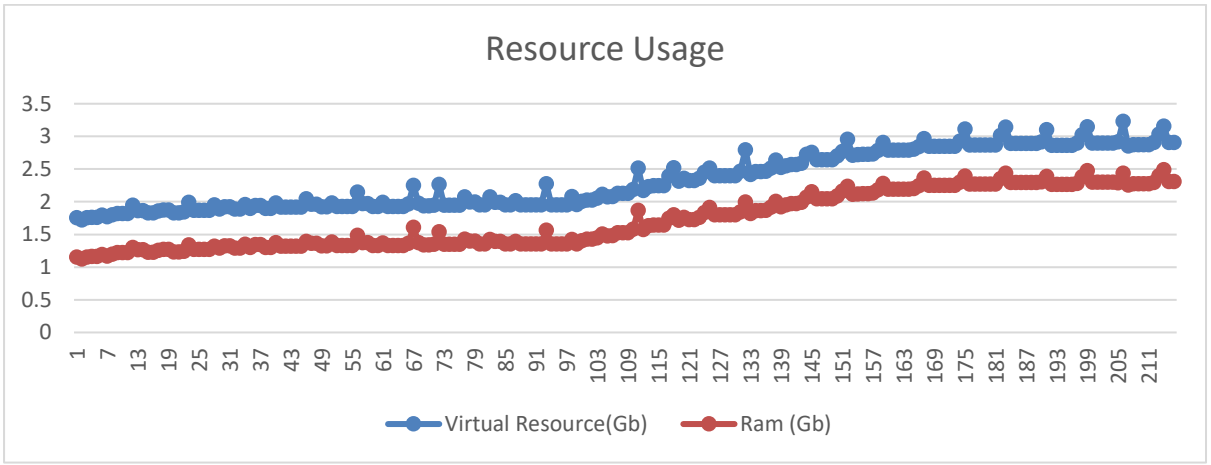


Figure 45 Octomap Memory Usage

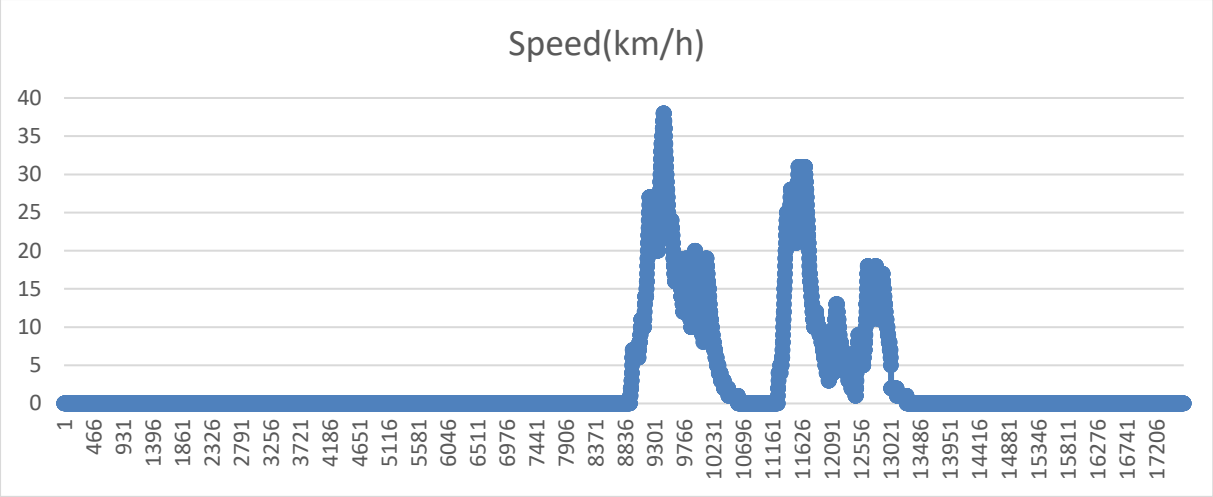


Figure 46 Octomap Vehicle Speed Trend

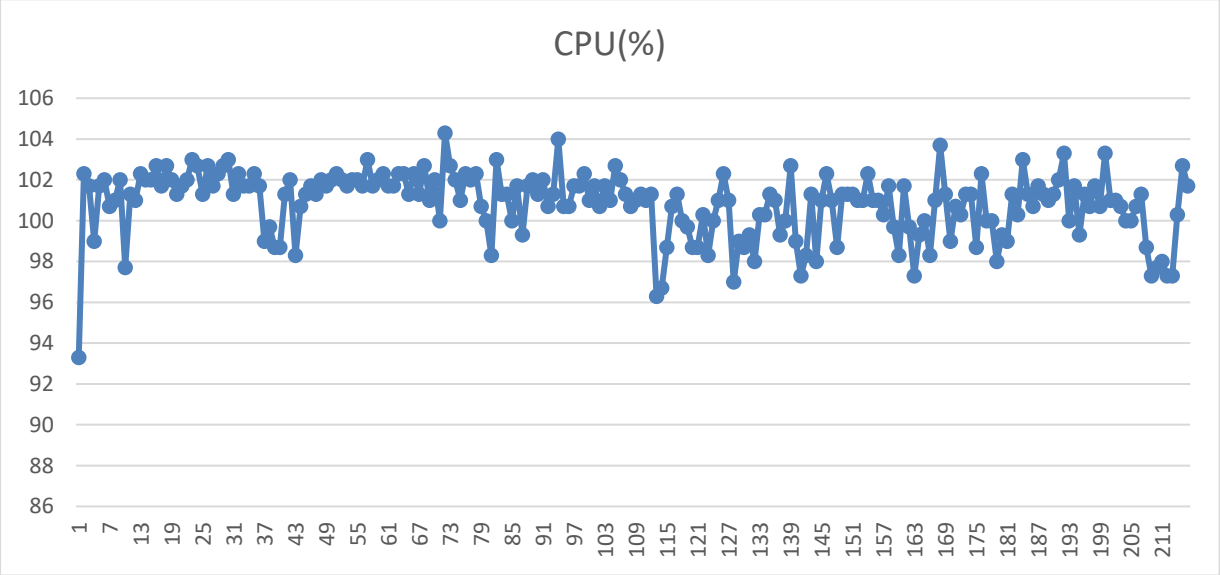


Figure 47 Octomap CPU Usage

1.24 RTAB

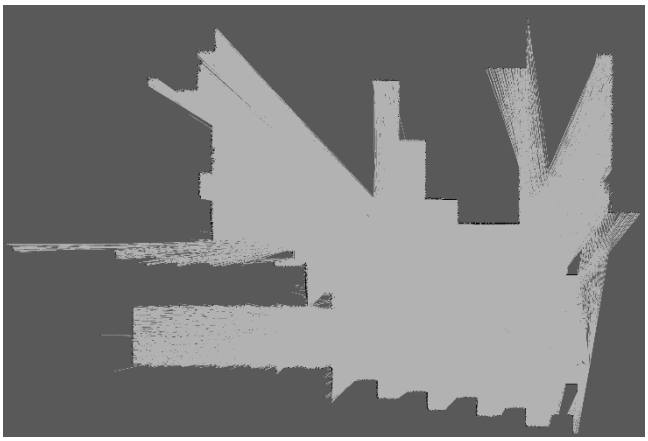


Figure 48 Rtab 2D map

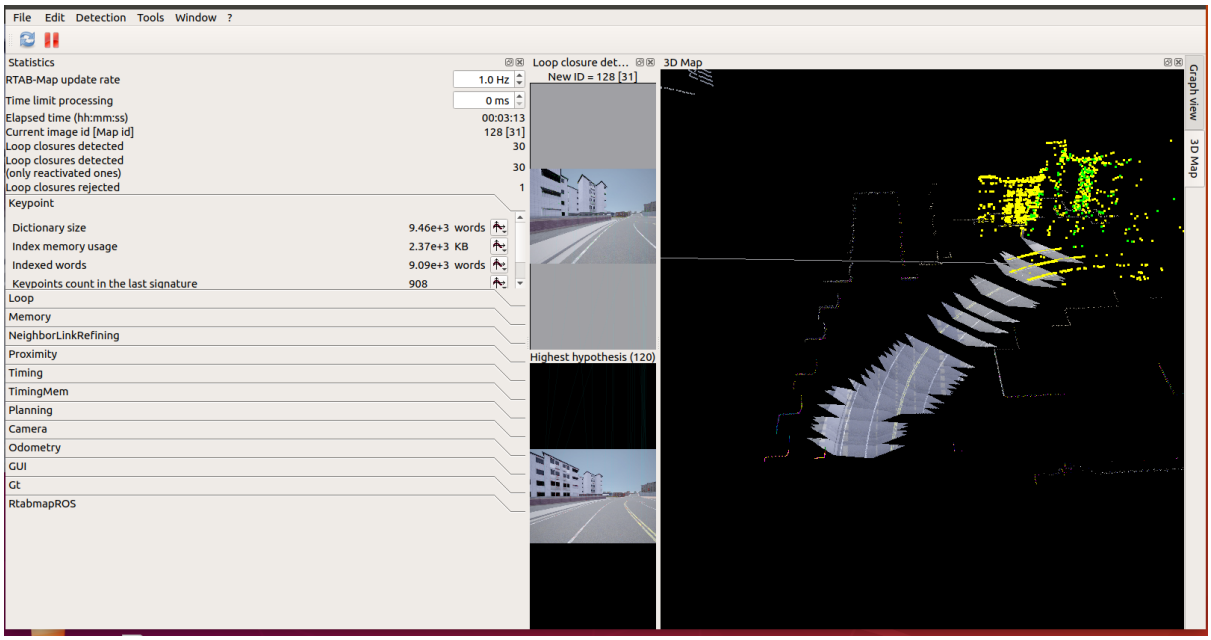


Figure 49 Rtab 3D Map

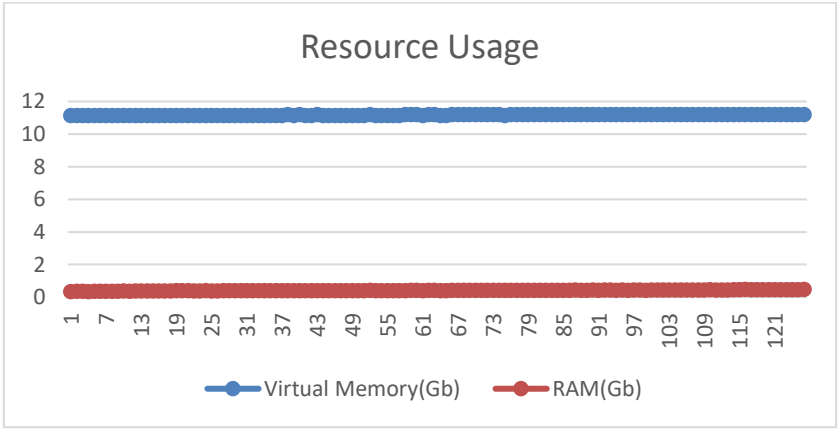


Figure 50 Rtab Resource Usage

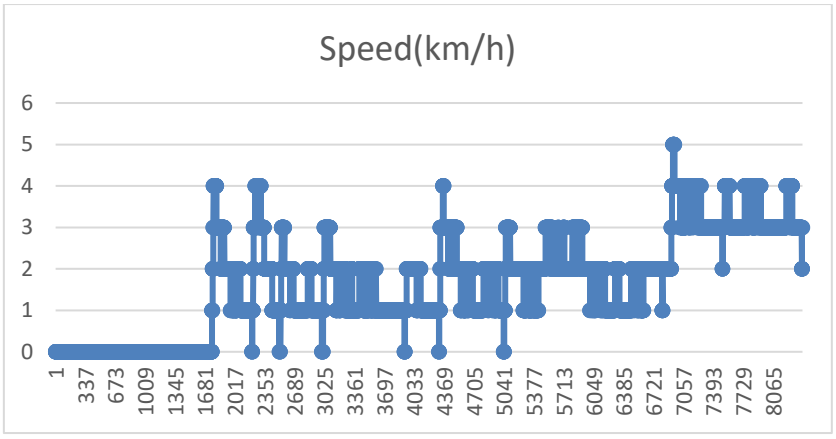


Figure 51 Vehicle Speed During RTAB Map

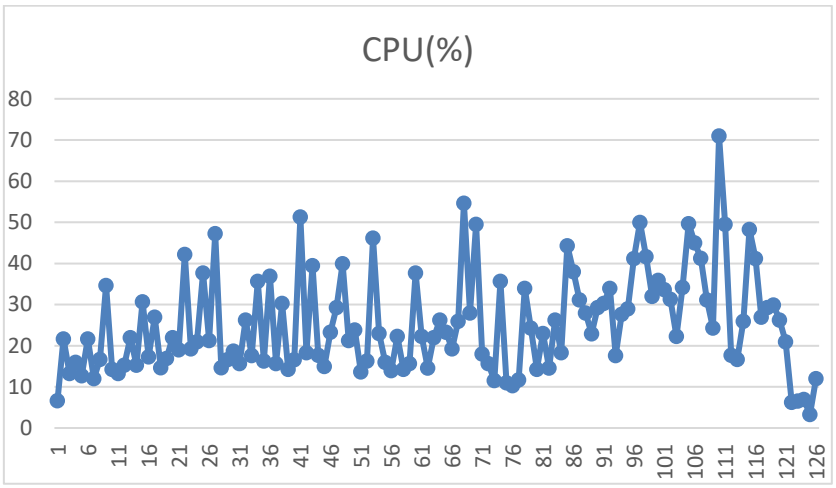


Figure 52 RTAB SLAM CPU Usage

1.25 DNN Based Visual SLAM Test Result

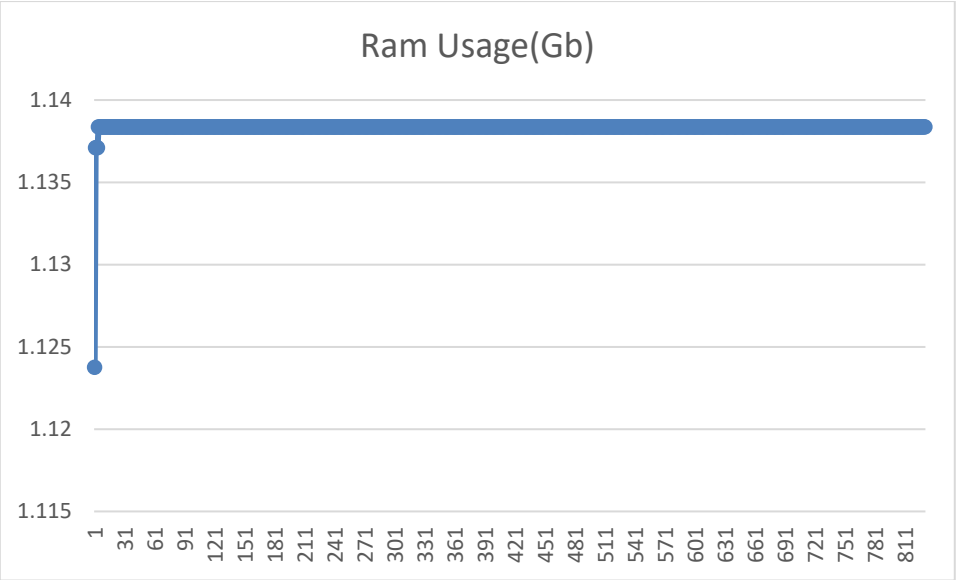


Figure 53 DNN SLAM RAM Usage

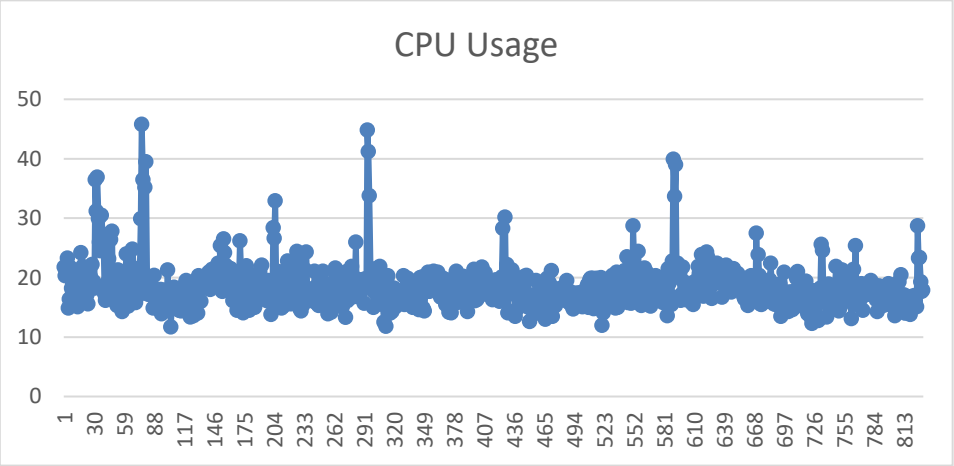


Figure 54 DNN SLAM CPU Usage

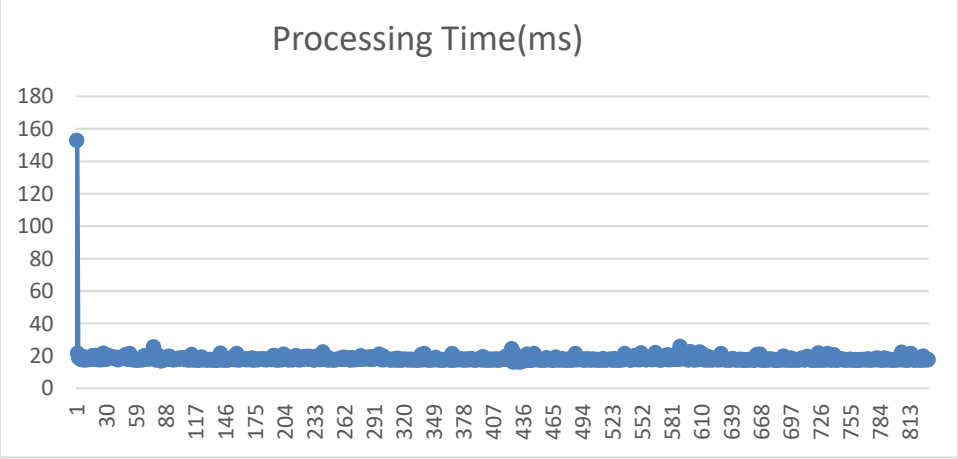


Figure 55 DNN SLAM Frame Processing Time

1.25.1 Object Detection and Semantic Information Output

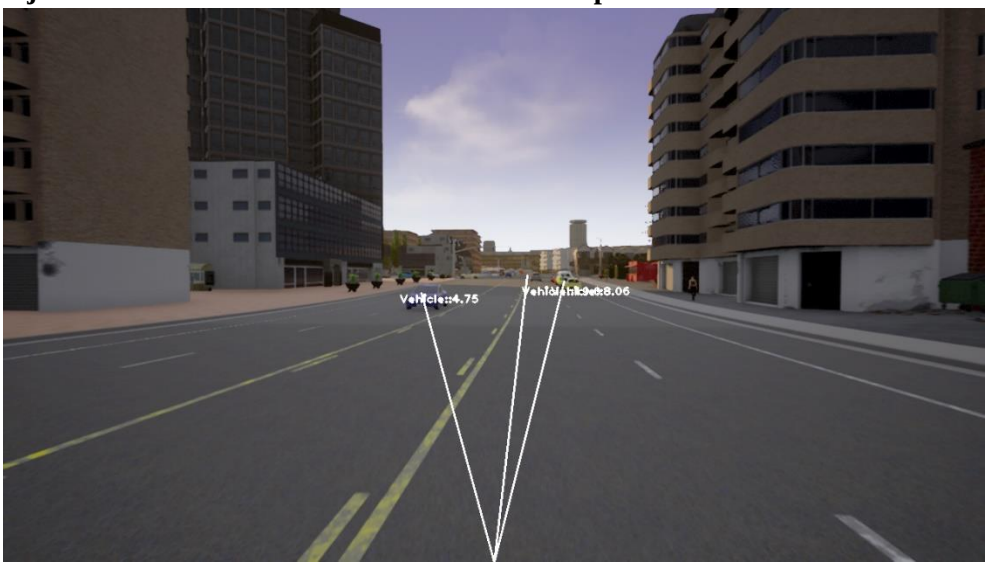


Figure 56 Detection Result 1

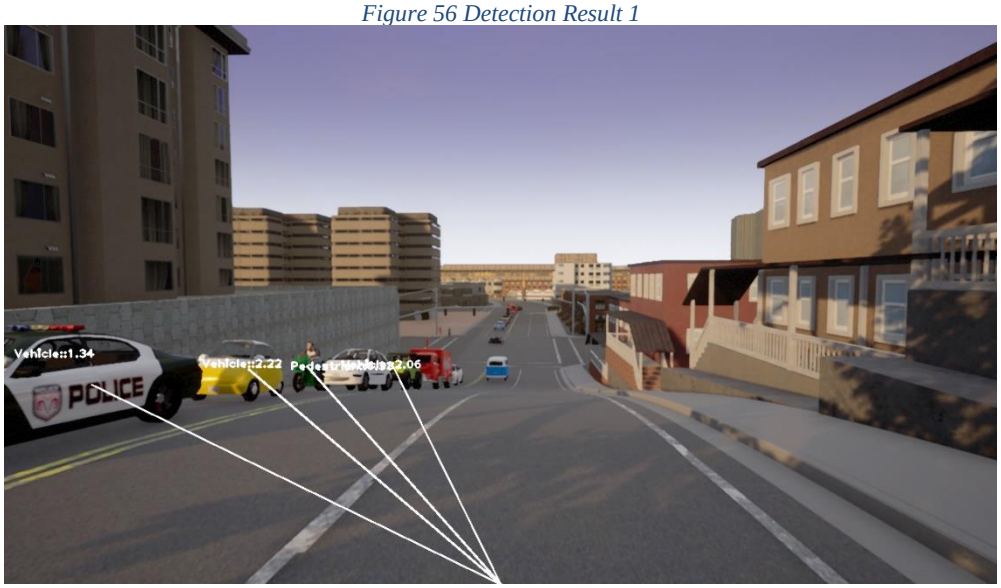


Figure 57Detection Result 2

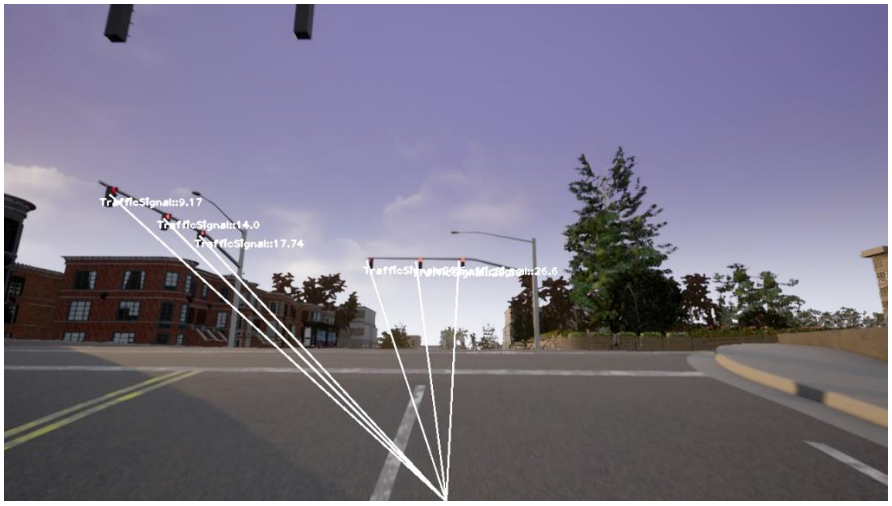


Figure 58 Detection Result 3

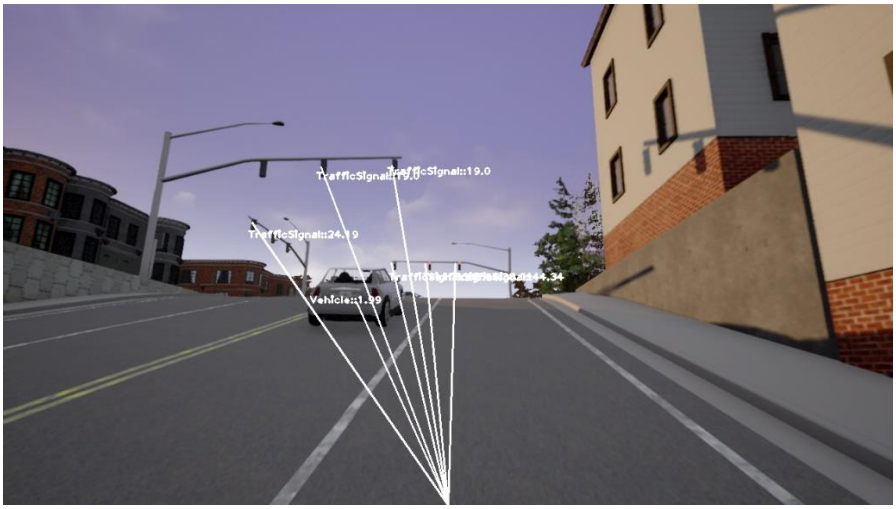


Figure 59 Detection Result 4

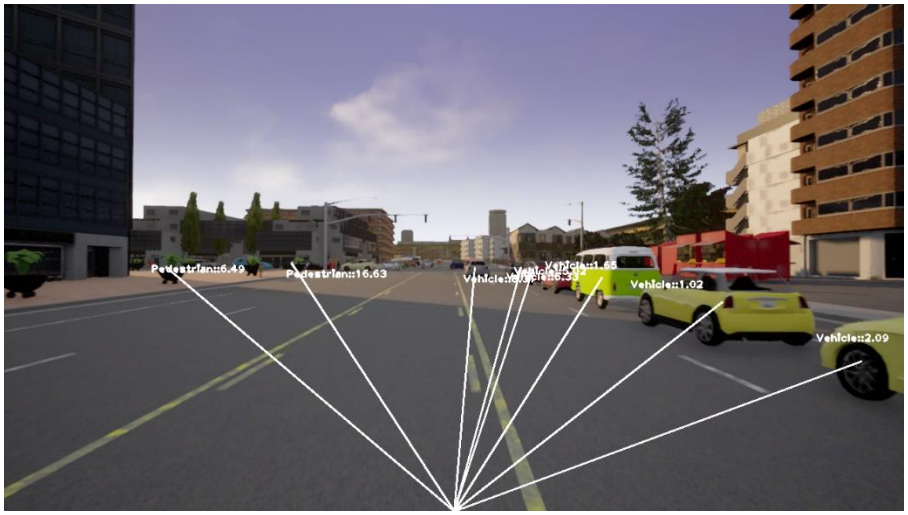


Figure 60 Detection Result 5

CXC

1.26 GPS & IMU Sensor Fusion Tests

1.26.1 Test Case 1

Test Attribute: Accelerometer Standard Deviation 0.2

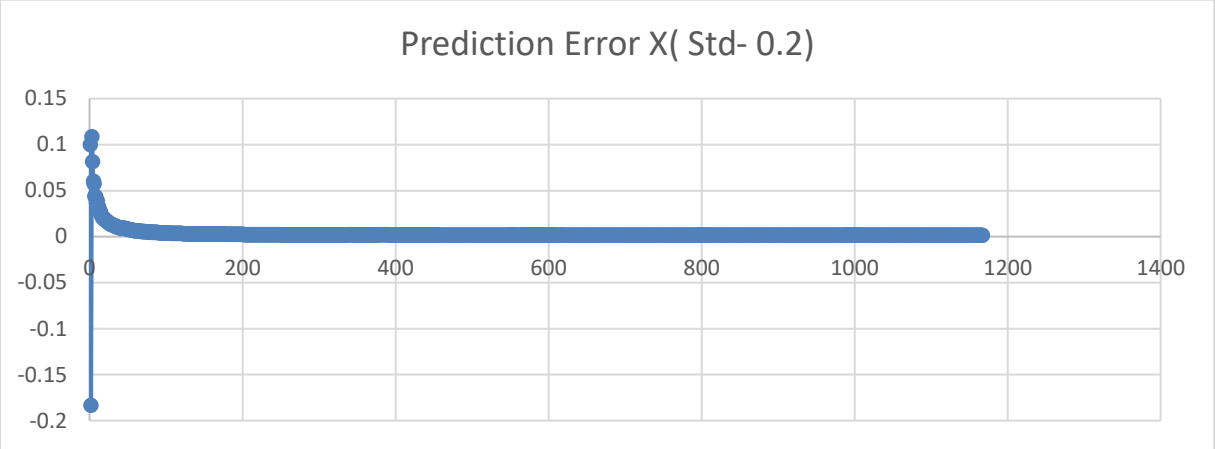


Figure 61 Prediction_X_T1

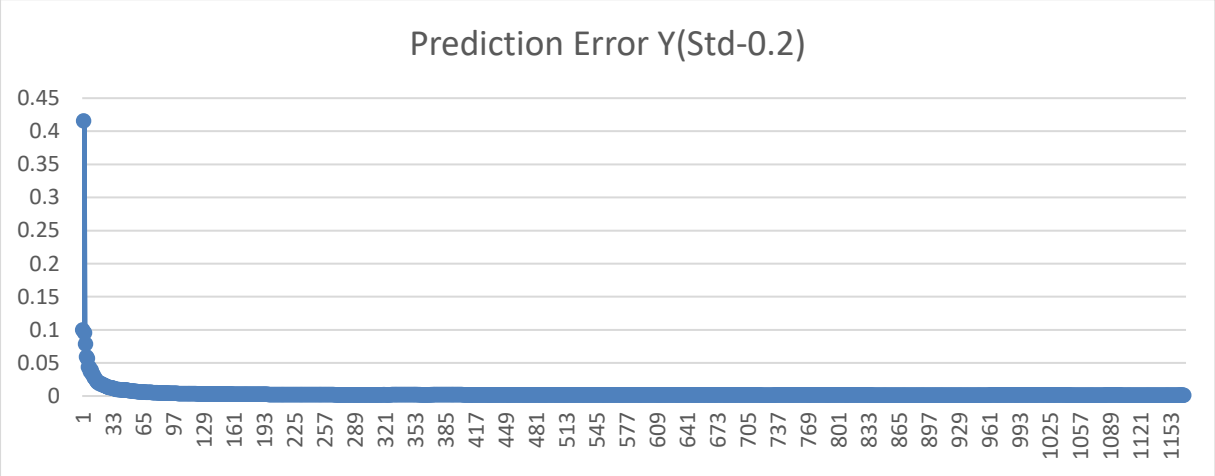
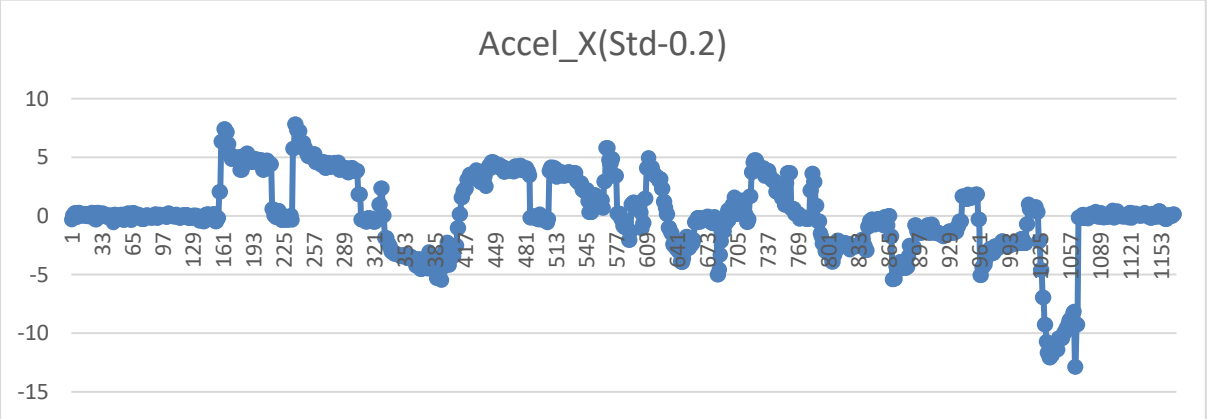
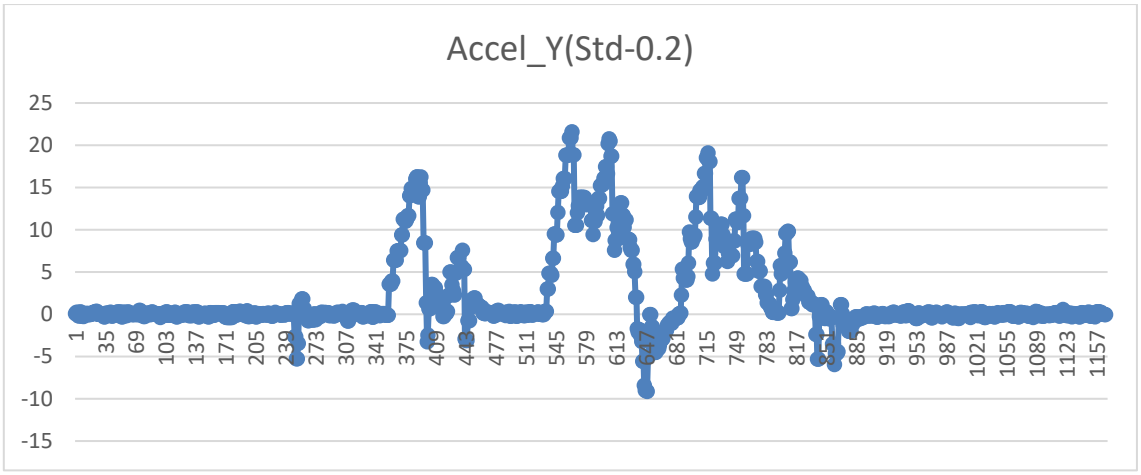


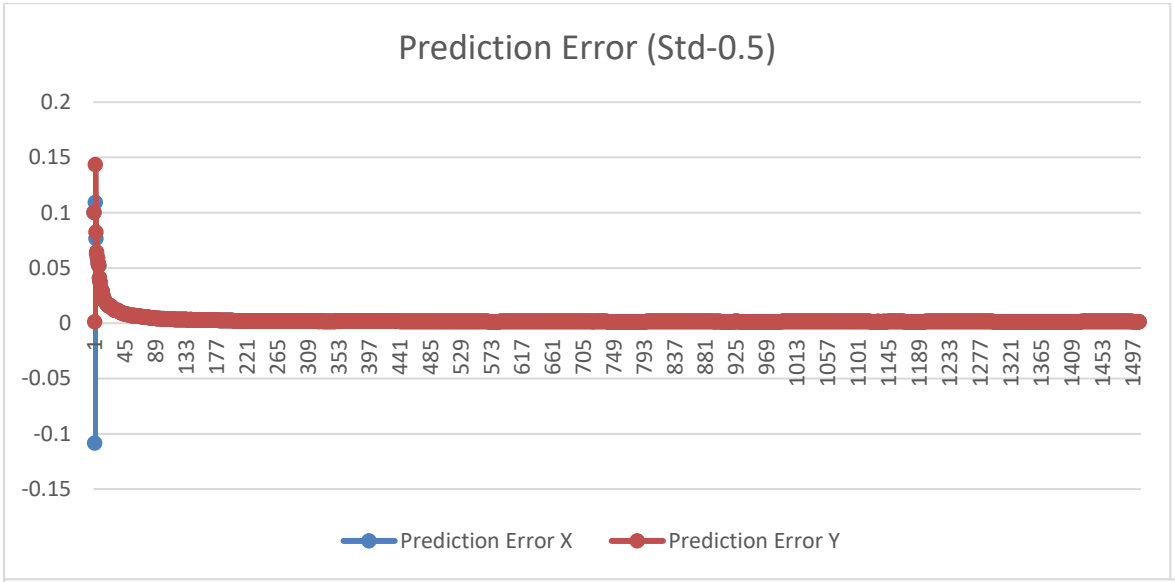
Figure 62 Prediction_Y_T1

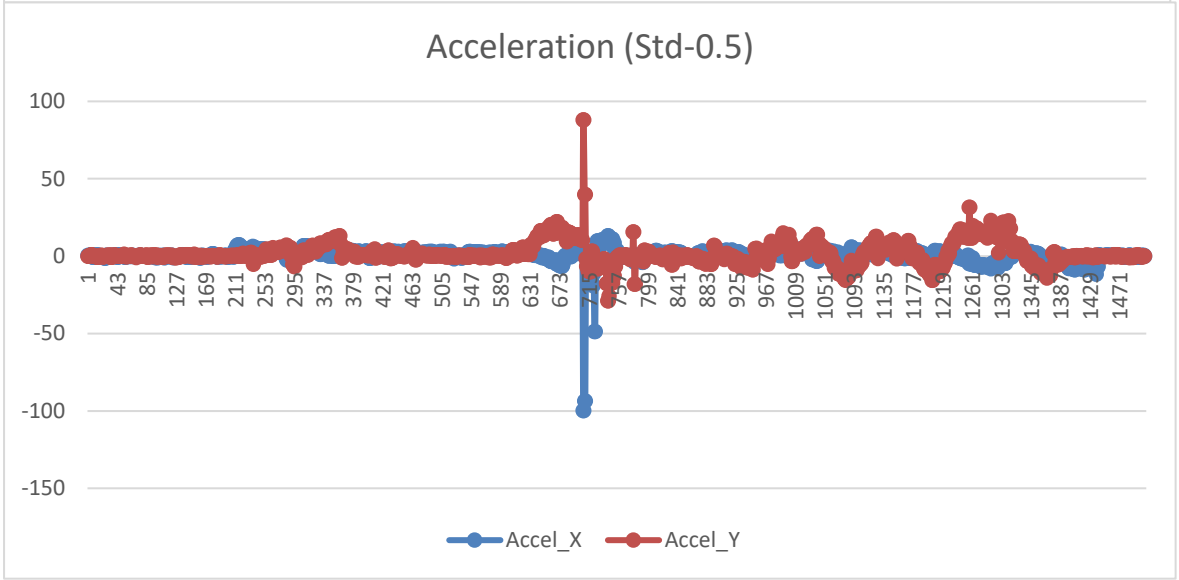




1.26.2 Test Case 2

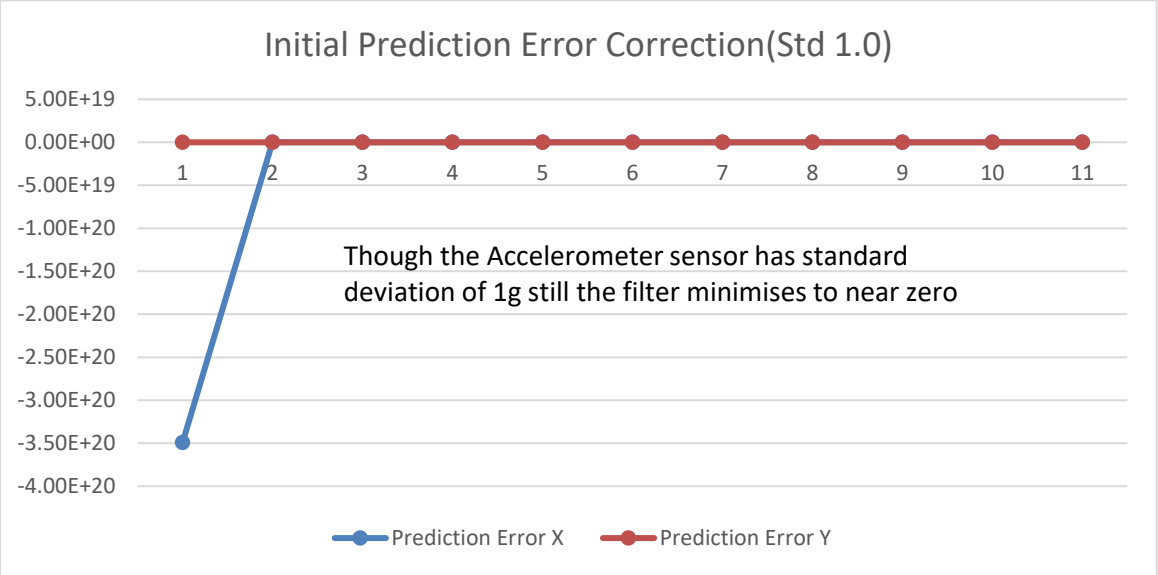
Test Attribute: Accelerometer Standard Deviation 0.5

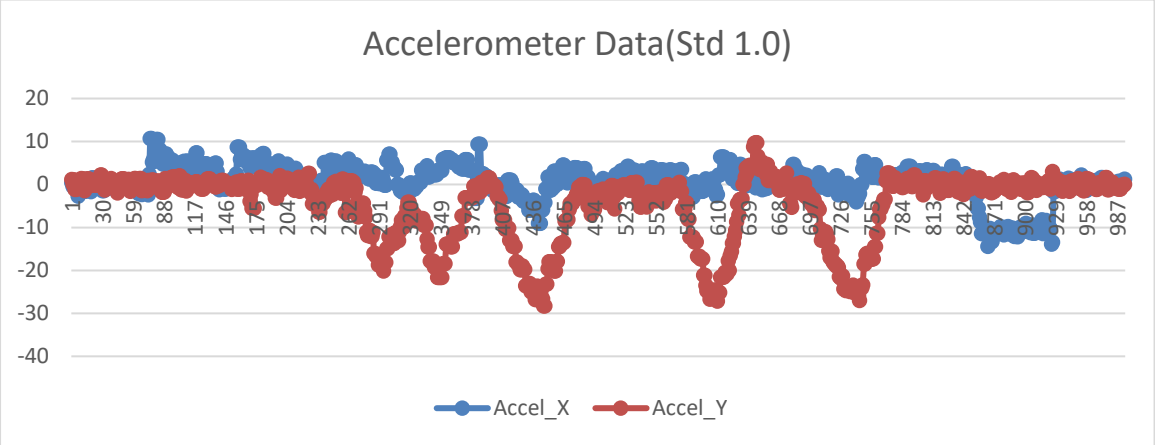


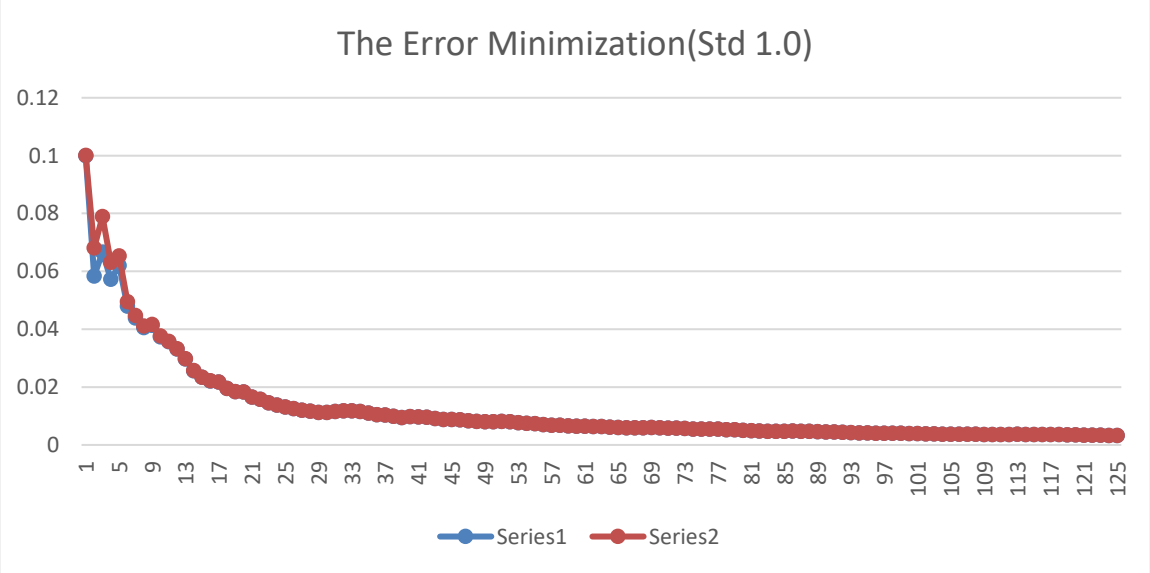


1.26.3 Test Case 3

Test Attribute: Accelerometer Standard Deviation 0.5







14.Test Result Comparison Table

	Gmapping	**Hector map**	**Octograph**	**RTAB**	**DNN based Visual SLAM**
Sensors Used	2D LiDAR	2D LiDAR	3D LiDAR(32 Channel)	RGBD sensor,2D LiDAR	1RGB Camera
Average CPU(%)	88%	18%	91%	40%	25%
Average RAM(Gb)	1.5Gb & Incremental	1.5Gb	2.4Gb & Incremental	0.4Gb RAM , 20Gb Virtual Memory	1.2Gb
Max Achievable Vehicle Speed	8kmph	15	20	3	40kmph
Frame Processing Time	NA	NA	NA	~10sec (Loop Closure)	17ms
Cost	$	$	$$	$	$
Semantic Information	No	No	No	No	Yes

15.Conclusions

In the project a low-cost autonomous driving sensor kit is proposed , In Section 16 , the comparison table summarises the project results , clearly the DNN based SLAM outperforms the standard slam implementations using only one camera in terms of memory , processing and real-time response. As camera cannot provide navigation data , a robust navigation stack using IMU and GPS fusion is implemented and evaluated. The project was able to achieve ~60fps frame processing capability using DNN based object classifier tiny-Yolov3 . Furthermore, standard SLAM implementations with different sensor suite is integrated with Carla simulator which can be used for future slam implementation reference .

Overall project objectives are met as defined in the project scope

16.Future Work

- Depth perception using only one camera.
- Implementation of current project architecture in SBC like Jetson Nano.
- Real-world evaluation of the sensor kit.

17.Bibliography

Ackerman, E., 2016. *Sweep Is a $250 LIDAR With Range of 40 Meters That Works Outdoors.* [Online]
Available at: https://spectrum.ieee.org/automaton/robotics/robotics-hardware/sweep-lidar-for-robots-and-drones
[Accessed 11 01 2020].

Appiah, N. & Bandaru, N., 2016. *Obstacle detection using stereo vision for self-driving cars,* California: Stanford University.

arxiv, 1991. *arxiv.* [Online]
Available at: https://arxiv.org/
[Accessed 02 02 2020].

Bovbel, P., 2019. *pointcloud_to_laserscan.* [Online]
Available at: https://github.com/ros-perception/pointcloud_to_laserscan
[Accessed 12 08 2019].

Boyen, X. & Koller, D., 2013. *Tractable Inference for Complex Stochastic Processes.* s.l., arXiv.

Brekke, A., Vatsendvik, F. & Lindseth, F., 2019. *Multimodal 3D Object Detection from Simulated Pretraining,* Cham: Springer.

Carla Installation, 2019. *Linux Build.* [Online]
Available at: https://carla.readthedocs.io/en/latest/build_linux/#unreal-engine
[Accessed 02 01 2020].

Carla, 2019. *Carla Simulator.* [Online]
Available at: http://carla.org/
[Accessed 10 09 2019].

Comma.ai, 2018. *Comma.ai.* [Online]
Available at: https://comma.ai/
[Accessed 01 02 2020].

Davison, A. J., Reid, I. D., Molton, N. D. & Stasse, O., 2007. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Volume 29, pp. 1052-1067.

Filbrandt, G., Stevenson, N., Ho Son, J. & Popat, R., 2019. *Extended Kalman Filter - SLAM.* [Online]
Available at: https://www.doc.ic.ac.uk/~ns5517/topicsWebsite/EKF-SLAM.html

Frese, U., Wagner, R. & Rofer, T., n.d. *A SLAM Overview from a User's Perspective,* Bremen: Deutsches Forschungszentrum fu¨r Ku¨nstliche Intelligenz GmbH.

Gerkey, B., 2019. *slam_gmapping.* [Online]
Available at: http://wiki.ros.org/gmapping
[Accessed 02 12 2019].

Girshick, R., 2015. *Fast R-CNN,* s.l.: ICCV .

Girshick, R., Donahue, J., Darrell, T. & Malik, J., 2013. *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report(v5),* berkeley: arXiv.

Grisetti, G., Stachniss, C. & Burgard, W., 2007. *Improved techniques for grid mapping with rao-blackwellized particle filters.* s.l., s.n., pp. 34-46.

Hermalyn, B., 2019. *Building Self-Driving Hardware at Scale.* [Online]
Available at: https://medium.com/cruise/building-self-driving-hardware-at-scale-29589d2b4a09
[Accessed 30 01 2020].

Hess, W., Kohler, D., Rapp, H. & Andor, D., 2016. Realtime Loop Closure in 2D LiDAR SLAM. *IEEE International Conference on Robotics and Automation(ICRA), Stockholm*, 16-21 May, pp. 1271-1278.

ieee, 1963. *Ieee Explore.* [Online]
Available at: https://ieeexplore.ieee.org/
[Accessed 02 02 2020].

Intel, 2019. *Beginner's guide to depth.* [Online]
Available at: https://www.intelrealsense.com/beginners-guide-to-depth/
[Accessed 27 01 2020].

Jiang, G. et al., 2019. *A SLAM Framework for 2.5D Map building based on Low-Cost LiDAR and Vision Fusion,* Shenzhen: MDPI.

Kohlbrecher, S., 2019. *hector_mapping.* [Online]
Available at: http://wiki.ros.org/hector_mapping
[Accessed 26 12 2019].

Kohlbrecher, S., Stryk, V., Meyer, J. & Klingauf, U., 2011. A flexible and scalable SLAM system with full 3D motion estimation. *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto*, pp. 155-160.

Kou, B. W., Chang, H. H., Chen, Y. C. & Huang, S. Y., 2011. A Light-and-Fast SLAM Algorithm for Robots in Indoor Environments Using Line Segment Map. *Journal of Robotics*, p. 12.

Kuzmin, M., n.d. *Review, Classification and Comparison of the Existing SLAM Methods for Groups of Robots.* Russia, LETI.

Labbe, M. & Michuad, F., 2013. *RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,* Canada: 3IT.

LeCun, Y., Buttou, L., Bengio, Y. & Haffner, P., 1998. *Gradient-Based Learning Applied to Document Recognition.* Montreal, IEEE.

Lopez, E. et al., 2017. A Multi-Sensorial Simultaneos Localization and Mapping System for Low Cost Micro Aerial Vehicle in GPS-Denied Environment. *Sensors,* Volume 17, p. 802.

Lynn, A., 2019. *The future of autonomous vehicle camera safety systems.* [Online]
Available at: https://www.electronicspecifier.com/products/artificial-intelligence/the-future-of-autonomous-vehicle-camera-safety-systems
[Accessed 1 29 2020].

Mao, M., Zhang, H., Li, S. & Zhang, B., 2019. *SEMANTIC-RTAB-MAP (SRM): A SEMANTIC SLAM SYSTEM.* Beijing, Mathematical Foundations of Computing,AIMS.

Mazzari, V., 2019. *Accueil.* [Online]
Available at: https://www.generationrobots.com/blog/en/what-is-lidar-technology/
[Accessed 11 01 2020].

Montemerlo, M., Thrun, S., Koller, D. & Wegbreit, B., 2002. *FastSLAM: A Factored Solution to the Simultaneos Localization and Mapping Problem.* Stanford, s.n.

Mur-Artal, et al., 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics,* Volume 31, pp. 1147-1163.

Murphy, K. & Russell, S., 2000. Bayesian Map Learning in Dynamic Environments. *NA*, p. NA.

O'Shea, K. & Nash, R., 2015. *An Introduction to Convolutional Neural Networks,* s.l.: ArXiV.

Pedrosa, E., Lau, N. & Pereira, A., 2013. Online SLAM Based on a Fast Scan-Matching. *Springer,* pp. 295-306.

R, D., S, K. & G, M., 2014. *Autonomous Vehicle Navigation and Mapping System ,* Madurai: IJIRSET.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A., 2015. *You Only Look Once: Unified, Real-Time Object Detection,* Washington: University of Washington.

Redmon, J. & Farhadi, A., 2016. *YOLO900,* Washington: arxiv.

ResearchGate, 2008. *Research Gate.* [Online]
Available at: https://researchgate.net
[Accessed 02 02 2020].

ros-bridge, 2019. *ros-bridge.* [Online]
Available at: https://github.com/carla-simulator/ros-bridge
[Accessed 02 01 2020].

Rublee, E., Rabaud, V., Konolige, K. & Bradski, G., n.d. *ORB: An Efficient Alternative to SIFT and SURF,* California: Willow Garage.

SAE, 2018. *SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles.* [Online]
Available at: https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles
[Accessed 05 03 2020].

Scholar, G., 2004. *Scholar.* [Online]
Available at: https://scholar.google.com/
[Accessed 02 02 2020].

Shaoqing, R., Kaiming, H., Ross, G. & Jian, S., 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Curran Associates Inc.*, pp. 91-99.

Sim, R., Elinas, P. & Griffin, M., 2005. Vision-based SLAM Using the RBPF. *IJCAI Workshop,* Volume 9, pp. 500-5009.

Smith, R., Self, M. & Cheeseman, P., 1987. Estimating uncertain spatial relationships in robotics. *IEEE International Conference on Robotics and Automation,Raleigh, NC, USA*, pp. 850-850.

Tesla, A., 2019. *Tesla Autopilot.* [Online]
Available at: https://www.tesla.com/autopilot
[Accessed 26 01 2020].

Thrun, S., Burgard, W. & Fox, D., 1998. *A probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots.* Boston, Kluwer Academic Publishers, pp. 1-25.

tomtom, 2019. *Benaluru Traffic.* [Online]
Available at: https://www.tomtom.com/en_gb/traffic-index/bengaluru-traffic#statistics
[Accessed 04 03 2020].

tomtom, 2019. *Traffic Index 2019.* [Online]
Available at: tomtom.com/en_gb/traffic-index/ranking/
[Accessed 06 03 2020].

Tzutalin, 2015. *LabelImg.* [Online]
Available at: https://github.com/tzutalin/labelImg
[Accessed 20 01 2020].

Uijlings, J., Sande, K. V. d., Gevers, T. & Smeulders, A., 2012. *Selective Search for Object Recognition*, Trento: IJCV.

Wurm, K. M., Bennwitz, M. & Burgard, W., 2012. *OctoMap: AnEfficientProbabilistic3DMappingFrameworkBasedonOctrees,* Freiburg: AutonomousRobots.

Xu, Y., Ou, Y. & Xu, T., 2018. *SLAM of Robot based on Fusion of Vision and LiDAR.* Shenzhen, IEEE International Conference on Cyborg and Bionic Systems(CBS).

Yadav, Y. et al., 2017. Comparison Of Processing Time Of Different Size Of Images And Video Resolutions For Object Detection Using Fuzzy Inference System. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, 6(1).

Zhong, F. et al., 2018. *Detect-SLAM: Making Object Detectuib and SLAM mutually Beneficial.* Beijing, IEEE Winter Conference on Applications if Computer Vision.

Appendix D – User Manual

Due to the size of the project files , the project is hosted in gitlab , Please follow the below link to proceed

https://gitlab.kpit.com/kankans/carlarosbridge

CarlaRosBridge

System information on which the above project is tested

Ubuntu 18.04 64bit

Nvidia 1050TI (435- Vulkan enabled)

i7 7th Gen

RAM 24GB

ROS melodic

How to load the project

Clone the git repo by issueing the below command

`git clone --recurse-submodules https://gitlab.kpit.com/kankans/carlarosbridge.git`.

Browse to the cloned folder cd carlarosbridge.

Compile the project by catkin_make.

Source the project source devel/setup.sh , Everytime when a new terminal is opened the carlarosbridge environment is sourced.

Run ROS roscore.

Run carla ./CarlaEU4.sh -windowed -ResX=400 -ResY=320 wait for the carla window to load.

If you want to launch the carla client with RVIZ then roslauch carlabg CarlaVehicleWithRviz.

If you want to launch the carla client without RVIZ then roslauch carlabg CarlaVehicleWithoutRviz.

You can edit the lauch file to play with the carla environment.

You can change the spawn point of the vehicle by changing the X,Y,Z location in the above line <arg name="spawn_point" default="73.193413,-136.704315,9.837398,-0.647311,-178.772690,0.000000"/> .

If you want to change the vehicle model edit the default model in the <arg name="vehicle_filter" default='vehicle.tesla.model3'/>.

If you want to change the weather of the simulation then change(Cloudiness, Precipitation , SunAltitudeAngle) <arg name="weather" default="90.0,0.0,1.0"/>

If you want to change the other actor models then change the below parameters <arg name="NumberofVehicles" default="1"/> <arg name="NumberofPedestrians" default="1"/> <arg name="SafeTravel" default="False"/>.

You can change the sensor config in sensors.json (carlarosbridge/carlabg/config/sensors.json).

The Navigation is broken and does not work properly, however the navigation can be launched by the following command

roslaunch carlaconfig amcl_demo.launch <-- this also launches RVIZ with the map

Note

The slam node did not work for me unless the messages are transmitted slowly.Hence while performing manual maneuver bag the messages by the following command

rosbag record /tf /camera/scan /carla/ego_vehicle/lidar/lidar1/point_cloud

Launch Gmapping node by running following commad

roslaunch carlabg Gmapping_Carla.launch

After the messages are logged we can playback the messages by following command

rosbag play -r 0.01 --clock "Your recorded file name"

This will take significant amount of time to build the map (better than the below error message)

(Error Message to be included ;-P)

Once you are done with the playback , save the message by following command

rosrun map_server map_saver -f "Your File Name"

All recorded data is stored in /src/data/

Hector and GMAPPING is tested against the two scenarios, Results are stored in the data folder ExperintData.tar.xz

Appendix F – Certificate of Ethics Approval

Coventry University

Certificate of Ethical Approval

Applicant:

Kankan Sarkar

Project Title:

Low-Cost Autonomous Mobile Robot Navigation Solution Using Sensor Data fusion and Machine Learning

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

30 October 2019

Project Reference Number:

P96683