Lateral Dynamics and Autonomous Navigation of Ground Vehicles

A project report

Submitted in partial fulfillment of the

requirement for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

MECHANICAL ENGINEERING

Submitted by

Gautam Kumar (160004017)

Under the Supervision of

Dr. Shanmugam Dhinakaran

Indian Institute of Technology Indore

November 2019

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Lateral Dynamics and Autonomous Navigation of Ground Vehicles"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Mechanical Engineering' completed under the supervision of **Dr. Shanmugam Dhinakaran, Associate Professor, IIT Indore** is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Gautam Kumar (160004017) **Date**

CERTIFICATE by BTP Guide

It is certified that the above statement made by the students is correct to the best of my knowledge.

Signature

Dr. Shanmugam Dhinakaran
Associate Professor,
Mechanical Engineering,
IIT Indore

Preface

This report on "Lateral Dynamics and Autonomous Navigation of Ground Vehicles" is prepared under the guidance of Dr. Shanmugam Dhinakaran.

An Ackermann vehicle has been mathematically modeled and control algorithms have been formulated to check its performance for various real-life situations. Then an autonomous navigation has been implemented on a robot and its efficiency has been realized through actual experimentation in indoor environment.

The results obtained from the experiments and simulation has been tabulated as well as presented in graphical form and a thorough study has been done.

Gautam Kumar (160004017)

B.Tech. IV Year
Discipline of Mechanical Engineering
IIT Indore

Acknowledgment

First and foremost, I would like to thank my supervisor **Dr. Shanmugam Dhinakaran** for guiding me thoughtfully and efficiently throughout this project, giving me an opportunity to work at my pace while providing with useful directions whenever necessary.

I would like to thank **Dr. Upendra Kumar Singh**, Director, Centre for Artificial Intelligence and Robotics (CAIR), DRDO for giving me the opportunity to do the project within the organization.

I am also highly indebted to **Mr. Nitin Kumar Dhiman**, Scientist 'E', CAIR, DRDO, for providing me the facilities to accomplish this internship and guiding me thoughtfully and efficiently throughout this project .

I would also like to thank all the people that worked along with me at CAIR, DRDO with their patience and openness to create a better working environment.

Finally, I offer my sincere thanks to all other persons who knowingly or unknowingly helped us in completing this project.

Gautam Kumar (160004017)

B.Tech. IV Year
Discipline of Mechanical Engineering
IIT Indore

Abstract

This project deals with the development of autonomous navigation in mobile robots which are used for surveillance purposes. There are various types of inaccessible areas where human safety is a major concern especially in border areas. To develop a complete robot which could decide how to navigate through its surroundings, it deals with four aspects:

1. Sensing the environment
2. Localizing itself
3. Planning the path to reach its goal
4. Execution of velocity inputs to track the planned path

The first question is answered by mounting sensors on the robot. IMU (Inertial Measurement Unit), GPS (Global Positioning System), wheel encoder and LiDAR (Light Detection and Ranging) have been used to provide the information about the environment to the robot. A cost effective way has been developed to get the position estimates of the robot by fusing IMU and encoder data using EKF (extended kalman filter).

To localize the robot, a particle filter algorithm called AMCL (adaptive monte carlo localization) have been used. Using the laser observations, the robot creates a map of its surrounding. Using these map landmarks, it predicts its current position in the map.

Two path planner algorithms have been implemented: TEB (Timed-Elastic Band) and DWA (Dynamic Window Approach). Their purpose is to detect the obstacle during the robot navigation and plan out paths to avoid the obstacle. TEB is found out to be good for Ackermann robots while DWA is suitable for differential-drive robots.

To track the path given to the robot two motion controllers, PID (Proportional-Integral-Derivative) and SMC (Sliding Mode Controller), have been designed. It has been found out that SMC can handle uncertainties which are prevalent when the robot moves in real world. A non-linear dynamic mathematical model of the robot has also been formulated to incorporate the non-linearities present in real world.

Table of content

List of Figures

List of Tables

Chapter 1

Introduction

As the autonomous cars have come into picture, there is a lot of enthusiasm about these running in real world. It is expected that an autonomous cars could navigate around their environment without any human intervention. After coming of optic vision guided Mercede-Benz robotic van in 1980, there have been huge rise in advancement of autonomous cars. The main aim is to develop vision guided systems using LiDAR, GPS, RADAR and computer vision. This has resulted into technologies like adaptive cruise control, lane parking, steer assist etc. which have become keys for autonomous navigation. With this pace several forecasts of automobile companies say that autonomous vehicles will become reality in near future.

Currently a lot of technologies are in development in order to make the autonomous ground vehicles (AGVs) work in a real world. This field is quite vast and there are many areas which are yet to be explored. Safety, reliability, robustness and security are some of the key aspects which need to be maintained for the AGVs. Motion controller and Path planning are one of the highly focussed fields and there has been a lot of development in improving and establishing high performance motion control to improve efficiency and safety factor in AGV's design.

1.1 History and development

The first step towards autonomous cars was a radio controlled car, called *Linriccan Wonder*. It was demonstrated by Houdin Radio Control in New York City. It was basically a 1926 *Chandler* consisting of transmitting antennae on its rear compartment and was operated by another car that sent out radio impulses while following it. RCA Labs built a miniature car in 1953. It was controlled and guided by wires that had been laid in a pattern on a laboratory floor. Based on advanced models, in 1959, and throughout the 1960s, in Motorama (which was an auto show by GM), Firebird had been showcased by General Motors, which was a series of experimental cars which had an electronic guide system which could rush it over an automatic highway without driver's involvement.

DARPA, Defence Advanced Research Projects Agency of the U.S. Department of Defence is also responsible for the progress in the field of autonomous vehicles. Autonomous Ground Vehicle (AGV) project in the United States made use of the then technologies. These technologies had been developed by the Carnegie Mellon University, the Environmental Research Institute of Michigan, University of Maryland, Martin Marietta and SRI International. The ALV project achieved the first road- following demonstration that used computer vision, LIDAR and autonomous control to guide a robotic vehicle at speeds of up to 3.1 km/h HRL Laboratories (formerly Hughes Research Labs) exhibited the first off-road

map and sensor- based autonomous navigation on the ALV. The vehicle travelled over 610 m at 3.1 km/h on complex terrain with steep slopes, ravines, large rocks, vegetation and other natural obstacles. [1]

In 1995 itself, the Carnegie Mellon University's NAVLAB project achieved 98.2% autonomous driving on a 5,000 km cross-country journey which was titled as "No HandsAcross America" or NHOA. [3] The car had been made semi-autonomous by nature: it used neural networks to control the steering wheel, but throttle and brakes were still human-controlled. An advanced autonomous vehicle was exhibited by Alberto Broggi of the University of Parma. ARGO Project launched by him, which worked on making a modified Lancia Thema to follow painted lane marks on a normal highway, in 1996. The apotheosis of the project was a journey of 1,900 km ix days on the roads of northern Italy, with an average velocity of 90 km/h. The car operated in complete automatic mode for 94% of its journey, with the longest automatic stretch of 55 km. The vehicle had two low-cost video cameras on board and used stereoscopic vision algorithms to analyse its environment.

In the early 2000s, the ParkShuttle, an autonomous public road transport system, became functional in the Netherlands. US government also began working on autonomous vehicles, mostly for military purposes. Demo I (US Army), Demo II (DARPA), and Demo III (US Army), were funded by the US Government (Hong, 2000). The ability of autonomous ground vehicles to navigate autonomously miles of difficult off-road terrain, avoiding obstacles such as rocks and trees was demonstrated by Demo III (2001).

1.2 Need for Autonomous Vehicles

Transportation accident is one of the main causes of death in the world. By 2026, this world could prevent 5 million human fatalities and 50 million serious injuries by introduction of latest and innovative methodologies and investments in road safety, from local to international levels. The Commission for Global Road Safety thinks that this is very necessary to stop this unacceptable and horrendous rise in road accidents, and initiate year on year reductions. [25] Deshpande et al. gave a data of almost 3000 deaths because of road injuries every day, with more than half of the passengers not travelling in a vehicle. Also, it has been currently reported by Deshpande et al. that if we don't take any paramount and efficacious action, transportation injuries are going to rise to 2.4 million per year, becoming the fifth leading reason of death in the world. It is believed that number of traffic collisions will drastically reduce due to increased reliability and faster reaction time in an autonomous system as compared to human drivers. Reduced traffic congestion is one bonus point, and thus roadway capacity is increased since autonomous vehicles will require a reduced need of safety gaps and better traffic flow management. Parking scarcity have become a historic phenomenon with the advent of autonomous cars, as cars could drop off passengers, and park at best space, and then return back to pick up the passengers. Thus, there would be a huge decrease in parking areas. Need of physical road signage will decline as autonomous vehicle will receive required information via network. Autonomous cars can surely reduce government spending on unnecessary things like traffic control officers. The need for vehicle insurance will also decline, along

with reduction in the incidents of car theft. We can be implement an efficient car sharing and goods transport systems (as in case of taxis and trucks respectively), with total elimination of redundant passengers. Not everyone is good at driving, so, autonomous cars provide a relief from driving and navigation chores. [2]

In Defence it is notably necessary to implement intelligence in robots. These robots really cost a lot to the government expenditure and its development is monitored thoroughly over a long time. These vehicles are built to work for the surveillance in areas where there is a concern about their safety. Also, mapping can be done using the robots in strategically important areas without any human intervention. This can greatly improve the death tolls of soldiers protecting our borders.

1.3 Aspects of Autonomous Navigation

Autonomous navigation is stated as the ability of the mobile robot to determine its current position within the reference frame environment using suitable sensors, plan its path through the terrain from the start toward the goal position using high planner techniques and perform the path using actuators, all with a very high level of autonomy. Robots are one of the modern technologies that humans are working on for many years. All these years of scientific study and research on robots have shown almost infinite possible application of robotic systems. In other words, the robot during navigation has to be able to answer the following questions:

- Where have I been? It is solved using cognitive maps.
- Where am I? It is determined by the localization algorithm.
- Where am I going? It is done by path planning.
- How can I go there? It is performed by motion control system

Navigation can be stated as the process by which we can accurately determine a system's position, planning and following a route. In the field of robotics, navigation means the way by which a robot finds its way in the environment and is a common necessity and requirement for almost any mobile robot. Robot navigation is a vast field and can be divided into subcategories for better understanding of the problems it addresses, the general problem of mobile robot navigation by four questions, each one addressed for a subcategory: Perception, Localization, Mapping and Path Planning

A. Perception. : Wheeled mobile robots need to sense the environment using sensors in order to autonomously perform their mission. Sensors are used to cope with uncertainties and disturbances that are always present in the environment and in all robot subsystems. Mobile robots do not have exact knowledge of the environment, and they also have imperfect knowledge about their motion models (uncertainty of the map, unknown motion models, unknown dynamics, etc.). The outcomes of the actions are also uncertain due to non-ideal actuators. The main purpose of the sensors is therefore to lower these uncertainties and to enable estimation of robot states as well as the states of the environment

B. Localization and Mapping: AGVs in manufacturing typically need to operate in large facilities. They can apply many features to solve localization and navigation. Quite often a robust solution is sensing induction from the electric wire in the floor or sensing magnetic tape glued to the floor. Currently the most popular solution is the usage of markers (active or passive) on known location and then AGVs localize by triangulation or trilateration. The latter is usually solved by a laser range finder and special reflecting markers. Other solutions may include wall following by range sensors or camera- or ceiling-mounted markers. All of the mentioned approaches are usefully combined with odometry. However, the recent modern solutions apply algorithms for SLAM which makes them more flexible and easier to use in new and/or dynamically changing environments. They use sensors to locate usually natural features in the environment (e.g., flat surfaces, border lines, etc.). From features that were already observed and are stored in the existing map the unit can localize, while newly observed features extend the map. Obtained maps are then used for path planning.

C. Path Planning: In environments that are mostly static the robots can operate using a priori planned routes. However, in dynamically changing environments they need to plan routes simultaneously. The most usual path planning strategy applies the combination of both mentioned possibilities where sensed markers on known locations in the environment enable accurate localization. When an unexpected obstacle is detected the AGV needs to find a way around the obstacle and then return and continue on the pre-planned path.

D. Motion Control: Motion control of AGVs is mostly solved by trajectory tracking, path following, and point sequence following approaches. These paths can be pre-computed or better planned online using a map of the environment and path planning algorithms. In situations where magnetic tape on the floor marks desired roads of AGVs they can use simple line following algorithms with the ability to detect obstacles, stop, and move around them. Efficiency of AGVs in crowded areas greatly depends on how the obstacle avoidance problem is solved.

1.4 Literature Review

Generally, the control algorithms assume that the kinematic relationship of the vehicle is linear. When the upper-limits on yaw rate and saturation constraints of steering actuators are not considered, the vehicle become unstable. To avoid such problem, [4] proposes a path planner based on optimized RRT (Rapid Random Tree) and a speed planner decoupled from path to conduct smooth trajectory. Based on longitudinal and lateral dynamics, a unified conditional internal control law is devised to steering control and driving torque/brake pressure. To execute the planned trajectory given by the user, it uses a hierarchical motion controller which generates desired driving torque/brake pressure and steering angle.

Experimental tests have been done on a modified electric intelligent car, which was equipped with a centralized drive motor, a steering motor and an electronically controlled hydraulic brake system.

Environmental information was obtained by LiDAR and monocular cameras for decision making and trajectory planning. The exact position of the vehicle was obtained by RTK and IMU. Experiments of turning left and right scenarios were carried out at an intersection. The width of road is 3.5m. Due to the large curvature during the turning process, a relative low uniform speed (10km/h) was chosen.

M.Bergerman [5] proposed the position and heading control of a robotic helicopter. They cascaded the model based LQR to stabilize the poles of a tested robotic with the Feedback Linearization Controller (FLC) to decouple and linearize the system with a simple linear PD controller. Suppachai [6] proposes double loop controller for vehicle's heading control under the real environment. The controller is decoupled into 2 loops that are cascaded. The inner loop was done by PID position control algorithm and the outer loop was done by PD heading control algorithm. The combination of PD-PID controller could improve the transient response of a vehicle while the desired heading changes abruptly. It uses transfer function found out through experiments. The model parameters were estimated using Least Squares Method.

In [7], multirate lane-keeping control scheme has been proposed to improve the lane-keeping efficiency and to avoid undesirable disturbance in yaw rate which can make the vehicle ride uncomfortable as chattering phenomena becomes higher. A virtual lane prediction algorithm was also considered in case of any momentarily failure of sensors.

Reference [8] describes a fuzzy and sliding mode control algorithm based on visual preview distance to promote the performance of tracking reference trajectory. The fuzzy control is quite effective in alleviating the chattering caused by SMC.

Gilles [9] has used the super-twisting algorithm to reduce the lateral displacement of the autonomous vehicle with respect to a given reference trajectory. It designs and experimental validation of a vehicle lateral controller for autonomous vehicle based on a higher-order sliding mode control. The advantage of this strategy is to reduce the chattering level and handle the uncertainties and nonlinearities in the vehicle.

Alcala [10] presents the comparison of two nonlinear model-based control strategies for autonomous cars. A control oriented model of vehicle based on a bicycle model has been used. The two control strategies used a model reference approach. Using this approach, the error dynamics model has been developed. The first control approach is based on a non-linear control law that is designed by means of the Lyapunov direct approach. The second approach uses a sliding mode-control that defines a set of sliding surfaces over which the error trajectories is expected to converge. The main advantage of the sliding-control technique is the robustness against non-linearities and parametric uncertainties in the model. However, the main drawback of first order sliding mode is the chattering, so it has been implemented a high order sliding mode control.

Anil [11] proposed a lateral vehicle dynamics control based on tyre force measurements. In this method, active front steering is employed to uniformly distribute the required lateral force among the front left and right tyres. The force distribution is found numerically through the tyre utilisation coefficients. In order to

consider the nonlinearities and uncertainties of the vehicle model, a gain scheduling sliding-mode control technique is used. In addition to stabilising the lateral dynamics, the proposed controller is good enough to maintain maximum lateral acceleration.

Kanghyun [12] proposed a robust yaw stability control system to stabilize the vehicle yaw motion. , a sliding mode control methodology is implemented to make vehicle yaw rate to track its reference with robustness against model uncertainties and disturbances. A parameter adaptation law is used to estimate varying vehicle parameters with respect to road conditions and is incorporated into sliding mode control framework.

Hamed Tabatabaei [13] discussed the effects of the Cornering Stiffness (CS) variation on directional stability of the Articulated Heavy Vehicle. . A linear planar model of articulated vehicle is applied to investigate the effects of CS variation on directional stability. Furthermore, the results derived by this analysis are verified through lane change manoeuvre simulation by a full nonlinear planar model of the articulated vehicle.

Sahoo [14] discusses about a realistic mathematical model of the vehicle considering the steering actuator dynamics. The cornering stiffness is calculated from the basic tire information and the vertical load on each tire. A heading angle controller of the UGV has been applied using the Point-to-Point navigation algorithm. Then, these controllers have been implemented on a test platform equipped with an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS).

In [15], Sahoo proposes a heading controller for an autonomous ground vehicle (AGV) to be designed and implemented taking into account the dynamic vehicle parameters. The well-known "bicycle model" approximation has been considered that considers the vehicle slip angle and ground-wheel interaction for the wheeled ground vehicle. Proportional and proportional-integral (PI) controllers have been designed, simulated and implemented to achieve the desired heading angle.

Zhengrong [16] proposes a new automated steering control method for vehicle lane keeping. The design of the steering controller is first found out to be established on the linear active disturbance rejection control, and then the controller is tuned in the framework of the quantitative feedback theory to get the required design related parameters on sensitivity and closed-loop stability. The parameter uncertainties of the vehicle system are applied at the tuning stage. The proposed steering controller is simulated and tested on a reduced-scale vehicle. Both the simulation and experimental results establish that the scaled vehicle controlled by the proposed controller performing the lane keeping.

Nakhaeinia [17] reviews various control architectures which are used extensively in autonomous navigation of wheeled robots. The advantages, significance and drawbacks of the architectures are thoroughly discussed and compared with each other. The control architectures can be seen into three divisions: Deliberative (Centralized) navigation, Reactive (Behaviour-based) navigation and hybrid (Deliberative - Reactive) navigation.

Chia-Feng [18] proposed a method that uses two wheeled, mobile robots to navigate unknown environments while cooperatively carrying an object. In the navigation method, a leader robot and a follower robot simultaneously perform either obstacle boundary following (OBF) or target seeking (TS) to reach a goal. The two robots are controlled by fuzzy controllers (FC) whose rules are accomplished through an *adaptive fusion of continuous and colony optimization particle swarm optimization* (AF CACPSO), which reduces the time-consuming task of manually designing the controllers.

Ghazi [19] proposes development of an overall routing system which uses input from common users via a simple android application and as a result directs the nearest vacant Cab towards the passenger. Two algorithms for the implementation of the project have been developed. The first algorithm is an autonomous route calculation algorithm in which a remote system has been used to calculate coordinates at each road intersection between any two input coordinates. The 2nd algorithm is a control algorithm that navigates the prototype robots. It is done by using Haversine heading and distance formulae.

Tzafestas [20] provides a global overview of mobile robot control and navigation methodologies developed over the last decades. It considers the following levels of wheeled mobile robots: kinematic modelling, dynamic modelling, conventional control, offline model-based control, invariant manifold-based control, model reference adaptive control, sliding-mode control, fuzzy and neural control, vision-based control, path and motion planning, localization and mapping, and control and software architectures.

Andreas [21] describes the fusion of sensor data for the navigation of an autonomous vehicle as well as two lateral control concepts to track the vehicle along a desired path. The fusion of navigation data is established on information provided by multiple object-detecting sensors. The object data is fused to increase the accuracy and to obtain the vehicle's state from the relative movement with respect to the objects.

Dieter Fox [22] proposes a new planner, Dynamic Window Approach (DWA), which generates velocities by keeping account of three important parameters: goal-distance, path-tracking and robot velocities. This algorithm has been devised for reactive collision avoidance. It differs from other approaches in that for finding commands which control the linear and rotational velocity of the robot has been done directly in the space of velocities. The advantage of this approach is that it correctly and in an elegant way incorporates the dynamics of the robot.

Guan M [23] presented an improved dynamic window approach method with collision suppression cone to handle the issue of a robot avoiding moving obstacles in a partially known dynamic environment. The concept of collision suppression cone is firstly proposed to define a probable collision area. When moving obstacles approach this area, the proposed DWA-CSC will be triggered to allow the robot avoiding the obstacles smoothly and thus preventing collision with them by controlling its motion direction and velocity, similar to that of obstacle avoidance done by human beings.

Garcia [24] proposed a sensor fusion methodology which gives intelligent vehicles with augmented environment information and knowledge, enabled by vision-based system, laser sensor and global positioning system. The above approach reaches roads safely by data fusion techniques, especially in single-lane carriageways where casualties are higher than in other road classes, and focuses on the interplay between vehicle drivers and intelligent vehicles. The system has been built on the reliability of laser scanner for obstacle detection; the use of camera based identification techniques and advanced tracking and data association algorithms.

1.5 Challenges

Autonomous car seems to be a pretty good idea but still there need to be a lot of improvement and research before it can be practically implemented. Although the notion has not been considered, but it is believed that an advent of autonomous cars would lead to reduction in driving related jobs. Also, conditions like inability of drivers to regain control of their vehicles due to inexperience of drivers, etc. are an important challenge. Lots of people enjoy driving, and it would be difficult for such people to forfeit control of their cars. Autonomous cars also face challenges while interacting with human-driven cars on the same route. Again, one challenge to autonomous cars is that who should be held responsible for damage- the manufacturing company, the government or the car's driver/owner. Thus, implementing a legal framework and establishing government regulations for autonomous vehicles is a major problem.

Apart from the above there are several technical challenges. Implementing intelligence in cars require a lot of training given to them. Accumulating such huge chunks of data for different road environment scenarios it poses a lot of challenges. Again adaptability of the vehicle is something where researchers need to work a lot. The most basic problem, i.e., moving a vehicle from one goal point to other is itself a big task because drift is always involved when a vehicle moves in straight line. These drifts get accumulated as the vehicle proceeds tracking the desired path. Accumulation of the drifts causes a lot of increase in the tracking error of the vehicle. The motion controllers designed for path tracking still pose a problem for accuracy and adaptability to change in vehicle parameters.

1.6 Motivation

Working with the robots requires a lot of sensors and every process needs to be controlled in real time. To use the sensors and actuators which need to be updated every 10-20 milliseconds, we need a type of interface/framework that gives this kind of benefits. Robot Operating System (ROS) provides us exactly with the same architecture to achieve this. It is open source and there are a lot of codes available from good research institutes.

Research institutes have come up with many algorithms which one can easily use and implement in their own robots. Further robot's engineers earlier didn't have a common platform for collaboration and communication which resulted into a delay of the adoption of robotic butlers and other related

developments that could have been done in no time. The robotic innovation has quickly paced up as ROS has come up since last decade wherein the engineers can readily build robotic apps and programs. Autonomous navigation is a very wide field which most of the researchers are trying to implement in the field of robotics. For a wheeled robot system to be autonomous, it has to analyse data from different sensors and perform decision making in order to navigate in an unknown environment. ROS helps us in solving different problems related to the navigation of the mobile robot and also the techniques are not limited to a particular robot but can be reused in different research development projects in the field of robotics.

1.7 Objective

This project has been completed in two phases. The first part deal with the development of mathematical model for the vehicle while the second part deals with the implementation of various algorithms to achieve autonomous navigation of the vehicle.

Firstly, the objective is to design a mathematical model for both the Ackermann vehicle and differential-drive vehicle and then the motion controller has been designed considering the dynamic and kinematic parameters of the vehicle. This is important as we can't test our algorithms and motion controller schemes each and every time on the robot. This seems practically not feasible as well as time consuming. So the first part of my project is to formulate a mathematical model of the vehicle which can incorporate various non-linearities and robustness of the vehicle so that we could closely predict the behaviour of the vehicle when its parameter gets changed. I have performed a sensitive analysis on various parameters which could affect vehicle's performance.

Secondly, autonomous navigation has been implemented on a differential-drive robot to test the efficiency of path planning algorithms. The desired objective is to make a robot completely autonomous so that it can move around in environments where human accessible is not possible. The robot is expected to localize its current position estimate and then based on the goal position, it should plan its path to achieve it. If any obstacle comes in between achieving its goal, it should avoid that and find a suitable path which can be taken. If no path is available, then it should stop its navigation and should not take undesirable paths. The passenger and payload safety is the primary concern when designing this robot.

Chapter 2

Vehicle Modelling and Motion Control

When people coined the term “vehicle model” around 90’s, what came to our thought was most likely a vehicle prototype that was highly complex, very expensive, and difficult to build. Engineers need to drive, and sometimes apply break, the then called “vehicle model” of the early 90’s to gather data. Today, most people would connect the term “vehicle model” with a computer representation that can be simulated under certain scenarios.

Some advantages of vehicle modelling are that the final product can be built quickly, it can be surely better (in terms of engineering requirements), and meanwhile reducing cost. Building and simulating a vehicle model in computers enables the engineers to analyse and determine if requirements are met for each design like several powertrain configurations. If engineering requirements are not met by a powertrain configuration, it is much easier to change a parameter in a computer model than it is to make a change to a vehicle prototype that is already built. For example, it doesn’t take much effort to change the power rating of an electric motor in a computer model, but we know it can be challenging to swap motors in a real vehicle. Another advantage of vehicle modelling is that the engineers get a good idea of the performance and energy consumptions aspects of a powertrain configuration from a vehicle model. Finally, let’s not forget that running computer simulation is faster and more cost effective than building and driving an actual vehicle prototype.

2.1 Vehicle Modelling Tools

There have been several softwares developed for modelling the car. In this project MATLAB & Simulink environment have been used for implementing the mathematical model and CarSim has been used to validate the results.

2.1.1 MATLAB & Simulink

MATLAB (matrix laboratory) is a proprietary programming language and a multi-paradigm numerical computing environment developed by MathWorks. MATLAB allows us several features like plotting mathematical functions and data, matrix manipulations, interface with other programs by interacting with other programming languages, and create user interface and many more. In this environment we can successfully implement the mathematical model developed for the vehicle. This method is quite fast and is very helpful in finding the effect of various parameters. Thus diagnostics of problem is relatively easy and fast.

Simulink is a graphical programming environment for modelling, simulating, and analysing dynamic systems. It provides built-in packages with formulated equations. We just need to add blocks to simulate a

vehicle. However, in this case it's difficult to change the parameters of the vehicle. We might have to change the transfer function of the entire block if we have to show variation of some parameters.
Though these methods are very simple but are not accurate. We have to take care of various uncertainties and non-linearities. This could only be done if the mathematical model formulated for the vehicle handles these. And to validate the built mathematical model we have to perform experiments on a real car.

2.1.2 CarSim

This software has been launched by Mechanical Simulation Corporation to simulate realistic vehicle responses to ADAS (Advanced Driver Assistance Systems) controls in these scenarios. CarSim, TruckSim, and BikeSim use vehicle data that describes suspension behaviour, powertrain properties, active controller behaviours, tire properties, and also road slope, obstacles, weather conditions, and asphalt type. At the core of the software is a simulation solver that can know in advance how the vehicle will react, for example whether it will be tipped off or skid under specific conditions or whether it will brake quickly enough on a wet surface. This software works on the data obtained from the real car experiments and thus the results predicted by it are quite close to real world. The vehicle model used in this software incorporates various non-linearities like friction, aerodynamic factors, non-linear tire models, non-uniform mass distribution in the vehicle etc.

2.2 Vehicle Models

The coordinate system used in vehicle dynamics modelling will be according to SAE J670e [18] as shown in Figure 2.1. The x-axis gives the forward direction or the longitudinal direction, the y-axis, represents the lateral direction, is considered positive when it points to the right of the driver, and the z-axis represents the ground satisfied by the right hand rule.

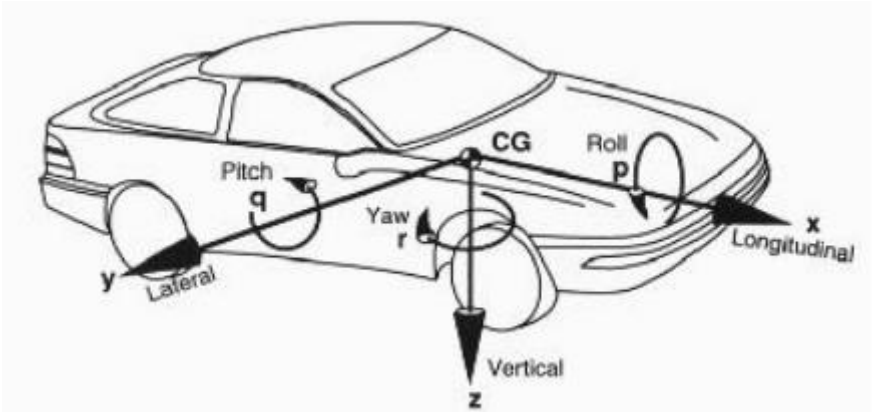


Figure 2.1 Vehicle Axis Systems

In most studies related to handling and directional control, only the X-Y plane of the vehicle is considered. The vertical axis, Z, is often used in the study of ride, pitch, and roll stability type problems. The following list defines relevant definitions for the variables associated with this report.

In most cases related to handling and directional control, only the ground plane of the vehicle is considered. The three stability criteria of ride, pitch and roll stability is generally studied using the analysis of the vertical axis, Z. The following list defines essential definitions for the variables associated with this report.

Longitudinal direction: vehicle moving along the forward direction. We can look at the forward direction in two ways, one can be with respect to the vehicle body itself, and other can be with respect to a fixed reference point.

Lateral direction: vehicle moving sideways direction. Again, we can look at the lateral direction in two ways, one is with respect to the vehicle and the other is with respect to a fixed reference point.

Tire slip angle: This is same as heading in a particular direction but moving at an angle to a direction by sideways displacing each foot laterally as we move along the ground.

There are various degrees of freedom associated with vehicle dynamics. The easiest vehicle dynamic model is *a two-degree-of-freedom,* fig 2.2 (a), bicycle model, representing just the lateral and yaw motions. The idea behind this model is that we just need to look along the lateral direction as dynamics is not much affected along the longitudinal directions at very lower speeds. Thus the longitudinal dynamics can't affect the lateral or yaw stability of the car. A *three-degree-of-freedom*, fig 2.2 (b), model considers longitudinal acceleration to the model, therefore allows us to describe the complete vehicle motion in the ground plane. In some cases, the rotational degrees of freedom for the front and rear wheels are included to the vehicle model to consider the effects of tire slip phenomena which increases as the vehicle speed increases. This *five-degree-of freedom*, fig 2.2 (c), model enables one to perform an in-depth study of traction and braking forces on handling manoeuvres by including the effects of wheel spin. An *eight-degree-of-freedom*, fig 2.2 (d), model no longer assumes symmetry in dynamic behaviour between right and left sides. In this vehicle model we consider rotational degree of freedom for each of the four tires instead of two tires. This model is widely used in the suspension design or ride comfort analysis. This model specifically looks at the effects of these issues with respect to roll and side-to-side load transfer.

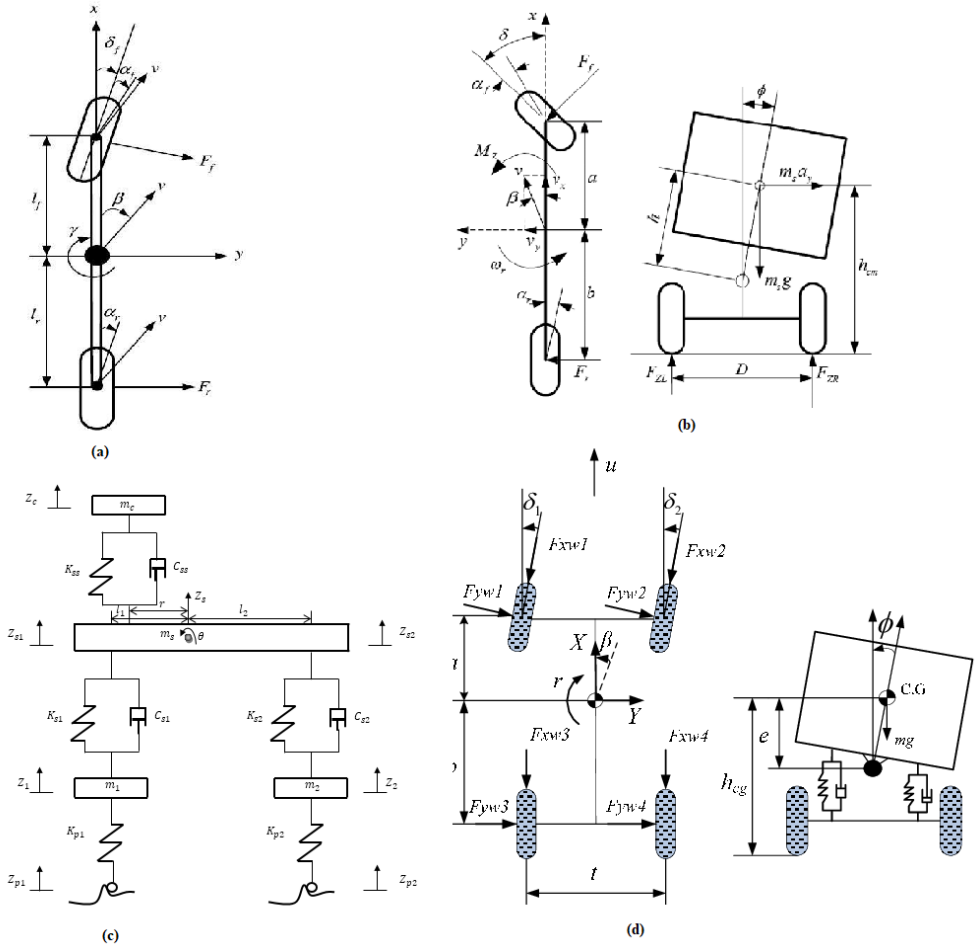


Figure 1.2 various vehicle models [(a) 2-DoF (b) 3-DoF (c) 5-DoF (d) 8-DoF]

2.3 Mathematical Model of Ackermann Steering Vehicles

A simple approximation of the lateral dynamics of land vehicles is the “bicycle model”. The two degrees of freedom are represented by the vehicle lateral position, y, and the vehicle yaw angle, θ. The vehicle lateral position is measured along the lateral axis of the vehicle and the vehicle yaw angle is measured with respect to the global X-axis. The lateral force at the tire-road interface depends on the slip angle. Figure 2.3 illustrates the bicycle model for a vehicle with no roll motion.

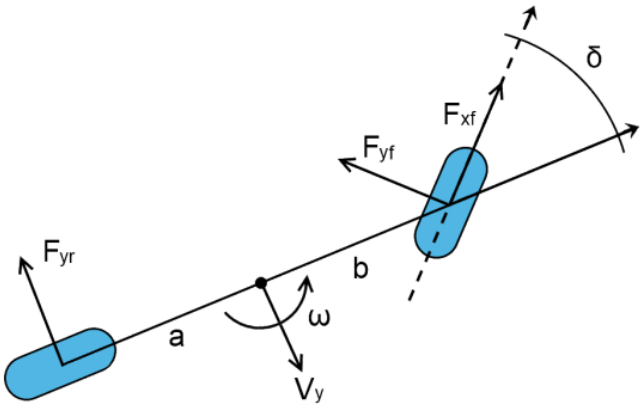


Figure 2.3 2-DOF Bicycle model

Nomenclature:

- m: mass of the vehicle
- m_f: mass on the front axle
- m_r: mass on the rear axle
- I_z: yaw moment inertia of vehicle
- δ: steering angle
- v_f: front tire velocity
- v_r: rear tire velocity
- v_x: longitudinal velocity of the vehicle
- v_y: lateral velocity of the vehicle
- F_{yf}: lateral force on the front wheel
- F_{yr}: lateral force on the rear wheel
- l: wheelbase length
- l_f: distance of front axle from vehicle CG
- l_r: diatance of rear axle from vehicle CG
- θ: yaw angle
- r: yaw rate
- C_f: Cornering stiffness of the front tire
- C_r: Cornering stiffness of the rear tire

Assumptions taken for our model:

- Left and right axles are lumped into a single wheel.
- Side-slip angles are small for linearization
- Tires operate at the linear region in which the slope of tire slip-angle and lateral force curve is constant.
- Suspension movement, road inclination and aerodynamic drag are neglected.
- Frictional losses are neglected and ideal road condition is chosen,
- Masses on each wheel are calculated by considering point mass loads. Similarly, inertia moment is calculated using above method.
- Only the front tires can be steered.

The lateral dynamics is sufficient to predict the vehicle behaviour for lower speeds and ideal road conditions. The vehicle is considered to move with a constant velocity, i.e., traction forces and rolling resistance are neglected. So it can be implied that F_{xr} and F_{xf} can be assumed to be zero.

The velocity vector $\dot{R} = u\hat{\imath} + v\hat{\jmath}$ which can be further differentiated to get acceleration vector

$$\ddot{R} = \dot{u}\hat{\imath} + u\dot{\hat{\imath}} + \dot{v}\hat{\jmath} + v\dot{\hat{\jmath}}$$

It can be demonstrated that

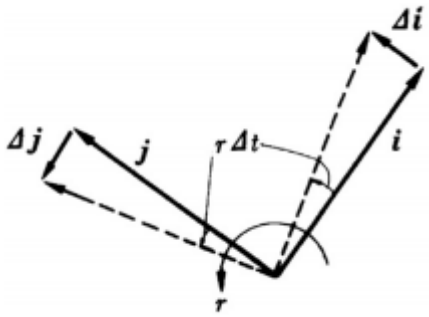


Figure 2.4 Velocity vector

$$\Delta i = r\Delta t\hat{\jmath} \;\; and \;\; \Delta j = -r\Delta t\hat{\imath}$$

So

$$\ddot{R} = (\dot{u} - rv)\hat{\imath} + (\dot{v} + ru)\hat{\jmath},$$

where $r = \dot{\theta}$

Since we are only concerned with lateral dynamics of our vehicle during this analysis, we keep longitudinal forces to be zero. Also we have already assumed longitudinal velocity to be constant.

By balancing forces along lateral direction, we have

$$m * (v + ur) = F_{yr} + F_{yf} * cos\delta$$

(2.1)

By balancing moment about COG of vehicle, we have

$$I_z * \ddot{\theta} = F_{yr} * l_r + F_{yf} * cos\delta * l_f$$

(2.2)

Tire side-slip angle (α) is defined as the angle between the tire traveling direction and the tire heading direction or the tire rotation plane. The rigid body vehicle has two velocity components: u in the longitudinal (x) direction and v in the lateral (y) direction. The vehicle also considers an angular velocity component around the centre of gravity. Consequently, each tire will have the velocity component of the centre of gravity and the velocity component due to rotation around the centre of gravity.

The lateral forces $F_{yr}\ and\ F_{yf}$ are related to the slip angles by cornering stiffness.

$$F_{yr} = C_r * \alpha_r$$

$$F_{yf} = C_f * \alpha_f$$

Where slip angles are given by

$$\alpha_f = \beta - \delta + \frac{l_f r}{V}$$

$$\alpha_r = -\delta + \frac{l_r r}{V}$$

By putting the values of α and solving equations by linearizing the parameters, we get the equation as

$$\begin{bmatrix} m\dot{v} \\ I_Z\dot{r} \end{bmatrix} = \begin{bmatrix} \frac{C_f+C_r}{u} & (-mu + \frac{C_f l_f - C_r l_r}{u}) \\ -(\frac{C_f l_f - C_r l_r}{u}) & \frac{C_f l_f^2 + C_r l_r \text{\^{}} 2}{u} \end{bmatrix} * \begin{bmatrix} v \\ r \end{bmatrix} + \begin{bmatrix} C_f \\ C_{f*} l_f \end{bmatrix} * \delta$$

(2.3)

The non-linear equations are derived as:

$$\dot{v}_y = -V * \theta + \frac{1}{m} * (C_r * \tan^{-1} \frac{(v_y - l_r * \dot{\theta})}{V} + C_f * \cos\delta * \tan^{-1} \frac{(v_y + l_f * \dot{\theta})}{V})$$

$$\dot{r} = \frac{1}{I_z} * (C_r * l_r * \tan^{-1} \frac{(v_y - l_r * \dot{\theta})}{V} + C_f * l_f * \cos\delta * \tan^{-1} \frac{(v_y + l_f * \dot{\theta})}{V})$$

(2.4)

2.4 Mathematical modelling of Differential-drive Robot

Differential drive is a simply designed driving mechanism that is widely used in practice, especially for smaller mobile robots used as surveillance and indoor environments. Robots with this mechanism usually consist of one or more castor wheels to support the vehicle motion and prevent tilting. Both the drive wheels are placed on a common axis. The angular velocity of each wheel is controlled by a separate actuator.

According to Fig. 2.5 the input (control) variables are the velocity of the right wheel v_R(t) and the velocity of the left wheel v_L(t). Other variables in Fig 2.5 represent: r is the radius of each wheel; L is the axial distance between the wheels and R (t) is the instantaneous radius of the vehicle driving trajectory, can be called as the distance between the vehicle centre and ICR point. In each instance of time, it is necessary that both wheels have the same angular velocity ω(t) around the ICR or it might cause instability.

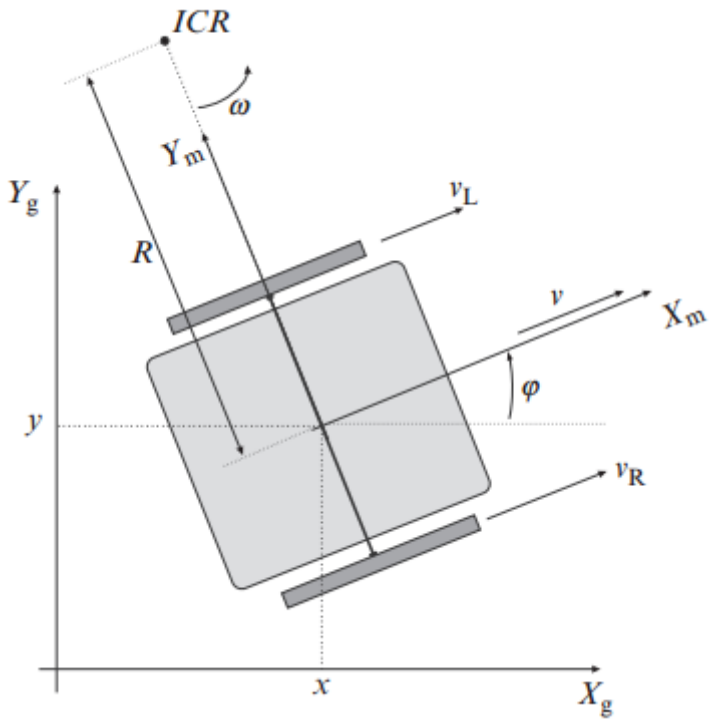


Figure 2.5 Differential drive kinematics

$$\omega = \frac{v_L(t)}{R(t) - \frac{L}{2}}$$

$$\omega = \frac{v_R(t)}{R(t) + \frac{L}{2}}$$

(2.5)

From where ω(t) and R(t) are found out to be

$$\omega(t) = \frac{v_R(t) - v_L(t)}{L}$$

$$R(t) = \frac{L}{2} * \frac{v_R(t) + v_L(t)}{v_R(t) + v_L(t)}$$

(2.6)

Tangential vehicle velocity is then calculated as

$$v(t) = \omega(t) * R(t)$$

Wheel tangential velocities are $v_L(t) = \omega_L(t) * r \; and \; v_R(t) = \omega_R(t) * r$ where $\omega_L(t) \; and \; \omega_R(t)$ are right and left angular velocities of the wheels around their axes joining the wheels, respectively. Considering the above relations the internal robot kinematics (in local coordinates) can be expressed as

$$\begin{bmatrix} \dot{x_m} \\ \dot{y_m} \\ \dot{\varphi_m} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{L} & \frac{r}{L} \end{bmatrix} * \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix}$$

The above relation, Eq.2.5, is important to understand how the robot is behaving internally but in practical robots users give angular velocity about z-axis, ω and a longitudinal velocity (*v)* commands.

So, robot external kinematics is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos(\varphi(t)) & 0 \\ \sin(\varphi(t)) & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$$

(2.7)

where v(t) and ω(t) are the control input variables.

Discretizing the above model, Eq. 2.6, using Euler integration and evaluated at discrete time instants t= k*T_s, k= 0,1,2 ….. where T_s is the following sampling interval:

$$x(k+1) = x(k) + v(k) * T_s \cos(\varphi(k))$$

$$y(k+1) = y(k) + v(k) * T_s \sin(\varphi(k))$$

$$\varphi(k+1) = \varphi(k) + \omega(k) * T_s$$

(2.8)

2.4.1 Forward and inverse kinematics

Forward kinematics considers the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters.

However inverse kinematics considers the kinematics equations to determine the joint parameters that provide a desired position for each of the robot's end-effectors.

The robot position at some time interval t is obtained by integrating the kinematic model, which is known as odometry or dead reckoning and is obtained in equations 2.9. Direct/Forward kinematics is the determination of the robot position for given control input variables.

$$x(t) = \int_0^t v(t) * \cos(\varphi(t))\, dt$$

$$y(t) = \int_0^t v(t) * \sin(\varphi(t))\, dt$$

$$\varphi(t) = \int_0^t \omega(t) dt$$

(2.9)

Applying Euler integration on Eq.2.9 gives the same results as Eq. 2.8 we obtained above:

$$x(k+1) = x(k) + v(k) * T_s \cos(\varphi(k))$$

$$y(k+1) = y(k) + v(k) * T_s \sin(\varphi(k))$$

$$\varphi(k+1) = \varphi(k) + \omega(k) * T_s$$

(2.10)

Trapezoidal rule gives a better approximation than the Euler's method

$$x(k+1) = x(k) + v(k) * T_s \cos\left(\varphi(k) + \omega(k) * \frac{T_s}{2}\right)$$

$$y(k+1) = y(k) + v(k) * T_s \sin\left(\varphi(k) + \omega(k) * \frac{T_s}{2}\right)$$

$$\varphi(k+1) = \varphi(k) + \omega(k) * T_s$$

(2.11)

Applying exact integration, we get the forward kinematics as:

$$x(k+1) = x(k) + \frac{v(k)}{\omega(k)} * (\sin(\varphi(k) + \omega(k) * T_s) - \sin(\varphi(k)))$$

$$y(k+1) = y(k) + \frac{v(k)}{\omega(k)} * (\cos(\varphi(k) + \omega(k) * T_s) - \cos(\varphi(k)))$$

$$\varphi(k+1) = \varphi(k) + \omega(k) * T_s$$

(2.12)

where integration in above equation is done inside the sampling time interval where constant velocities v and ω are assumed to obtain increments:

$$\Delta x(k) = v(k) \int_{kT_s}^{(k+1)T_s} \cos\big(\varphi(k) + \omega(k)(t - kT_s)\big)\, dt$$

$$\Delta y(k) = v(k) \int_{kT_s}^{(k+1)T_s} \sin\big(\varphi(k) + \omega(k)(t - kT_s)\big)\, dt$$

(2.13)

Developing the inverse kinematics of a robot is a challenging task than the above cases of direct kinematics. We use inverse kinematics to know about control variables to drive the robot to the desired robot pose or path trajectory. Robots are usually subjected to nonholonomic constraints, which mean that not all driving directions are possible. There are also many possible solutions to get to the desired position.

One of the simplest solution to the inverse kinematics problem would be if the mobile differential robot is allowed to drive only forward (vR(t) = vL(t) = vR ⇒ ω(t) = 0, v(t) = vR) or only perform on-spot rotation (vR(t) = −vL(t) = vR ⇒ ω(t) = 2LvR , v(t) = 0) at constant speeds.

For rotation motion equation simplifies to

$$x(t) = x(0)$$

$$y(t) = y(0)$$

$$\varphi(t) = \varphi(0) + \frac{2vRt}{L}$$

(2.14)

and for straight motion equation simplifies to

$$x(t) = x(0) + vR\,cos(\phi(0))t$$

$$y(t) = y(0) + vR\,sin(\phi(0))t$$

$$\phi(t) = \phi(0)$$

(2.15)

Motion strategy could then target to orient the robot to the target position by rotation and then drive the robot to the goal position by a straight motion and finally align (with rotation) the robot orientation with the user defined orientation in the desired robot position. The desired control input variables for each part of motion (rotation, straight motion) can then easily be calculated from the equations derived above.

There are many other ways to drive the robot to the desired position using trajectories that are smoothly designed. The inverse kinematic problem becomes easier for the desired smooth target trajectory as the computation become really less and the robot follow such that its orientation is always tangent to the trajectory computed. Trajectories are defined in time interval t ∈ [0, T]. Suppose that the robot's initial position is on trajectory, and there is a perfect kinematic model without non-linearities and disturbances, then we can calculate required control input variables v(t) as follows:

$$v(t) = \pm\sqrt{\dot{x}^2 + \dot{y}^2}$$

(2.16)

where the positive/negative sign depends on the direction(+ for forward and − for reverse) in which the vehicle is supposed to be drived. Then the tangent angle of each point on this trajectory is computed as

$$\phi(t) = arctan2(\dot{y}(t), \dot{x}(t)) + l\pi$$

(2.17)

where l ∈ {0, 1} defines the direction (0 for forward and 1 for reverse) in which the vehicle is to orient and the function arctan2 is stated as the four-quadrant inverse tangent function.

By differentiating the orientation angle with respect to time in eq. (2.17) the robot's angular velocity ω(t) is obtained as:

$$\omega(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{\dot{x}2(t) + \dot{y}2(t)} = v(t)\kappa(t) \tag{2.18}$$

where κ(t) is the path curvature. Using relations (2.14), (2.16) and the defined desired robot trajectory x(t), y(t), and the robot control variables v(t) and ω(t) are calculated. One of the most important criteria is that the path designed should be twice-differentiable and the tangential velocity v(t) should be non-zero. If for some given time *t* the tangential velocity becomes v(t) = 0, the robot starts rotating at a fixed point with the angular velocity ω(t). And then angle ϕ(t) cannot be obtained from Eq. (2.14), and therefore, ϕ(t) must be given explicitly. Usually we use this approach to determine the feed forward part of the control supplementary variables to the feedback part which takes care of the inaccurate kinematic model, disturbances, and the initial position errors which further leads to error accumulation.

2.5 Motion Controllers

Motion control of wheeled mobile robots in the environment is generally performed by controlling motion from some start pose to some goal pose (classic control, where intermediate state trajectory is not prescribed) or by reference trajectory tracking.

To track the desired heading angle, a closed loop negative feedback system was considered, as shown in Figure 2.8. This heading-angle controller will compute the required steering angle based on the error in the heading angle of the vehicle. The design criteria of the controller was to keep the steady-state error within the limit of 5% and the maximum overshoot less than 10%, with a settling-time requirement of less than 3 s.

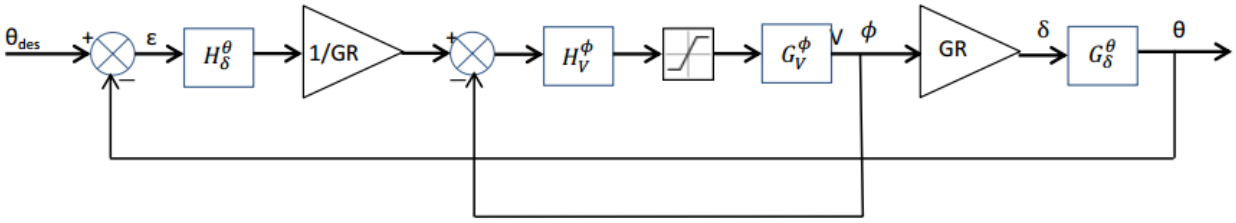


Figure 2.6 Block diagram of the vehicle

θ_{des}= input heading angle

H_δ^θ= heading angle controller

GR= gear ratio

G_δ^θ = Vehicle plant model

H_V^φ= Steering angle controller

G_V^φ= Steering actuator system

The steering-angle input to the vehicle depends on the desired heading angle for the given speed of the vehicle. The controller designed for this system consists of two loops. The inner-loop controller, the transfer function of which is H_V^φ (s), minimizes the error between the desired angular position and the current angular position of the steering motor. The input of this loop is a function of the steering angle which is calculated on the basis of the error in the heading angle of the vehicle. The outer-loop controller, the transfer function of which is H_δ^θ (s), decreases the error between the desired heading angle and the actual heading angle of the vehicle. The transfer function G_δ^θ (s), which relates the response of the heading angle to the steering angle, was obtained using equation (2.3,2.4). The steering actuator model was derived analytically from first principles in. A saturation block was added to restrict the voltage input from –20 V to + 20 V.

2.5.1 Proportional-Integral-Derivative (PID) Controller

This is a control loop feedback mechanism employed in various systems to regulate the errors by calculating the error in each loop and thus attempts to decrease the error values by adjusting control input to the process.

PID controllers have three control modes:

- Proportional Control- It changes the controller output in proportion to the error. If the error gets larger, the control action gets larger. If the controller gain is set higher then the control loop will start oscillating around the desired value and become unstable. If the controller gain is set very low, it cannot respond adequately to disturbances which might lead to large steady state errors.

- Integral Control- Integral action drives the controller output far enough by integrating error values over the period of time to reduce the error to zero. If the error is very large, the integral mode tries to increment/decrement the controller output faster to reduce the errors to zero. If the error values are smaller, the controller changes will be slower.

- Derivative Control- The derivative control mode produces an output based on the rate of change of the error. This mode produces more control input action if the error changes at a faster rate by looking at the derivatives of the error values. If there is no change in the error, the derivative action is zero.

2.5.2 Sliding Mode Controller

Sliding mode control (SMC) is a nonlinear control technique featuring remarkable properties of accuracy, robustness, and easy tuning and implementation. SMC systems are designed to drive the system states onto a particular surface in the state space, named sliding surface. Once the sliding surface is reached, sliding mode control keeps the states on the close neighbourhood of the sliding surface. Hence the sliding mode control is a two part controller design. The first part involves the design of a sliding surface so that the sliding motion satisfies design specifications. The second is concerned with the selection of a control law that will make the switching surface attractive to the system state. There are two main advantages of sliding mode control. First is that the dynamic behaviour of the system may be tailored by the particular choice of the sliding function. Secondly, the closed loop response becomes totally insensitive to some particular uncertainties. This principle extends to model parameter uncertainties, disturbance and nonlinearity that are bounded. From a practical point of view SMC allows for controlling nonlinear processes subject to external disturbances and heavy model uncertainties.

The most typical choice for the sliding manifold is a linear combination given by

$$\sigma = \dot{e} + c_0 e$$

From a geometrical point of view, the equation σ = 0 defines a surface in the error space, that is called "sliding surface". The trajectories of the controlled system are forced onto the sliding surface, along which the system behaviour meets the design specifications. A typical form for the sliding surface is the following, which depends on just a single scalar parameter, p.

$$\sigma = \left(\frac{d}{dt} + p\right) * e$$

Above model is first order sliding mode control. We used a saturation control law to avoid chattering.

The control is discontinuous across the manifold, $\sigma = 0$.

$$u = \begin{Bmatrix} sign(\sigma), when\ \left|\frac{\sigma}{\varepsilon}\right| > 1 \\ \frac{\sigma}{\varepsilon}, when\ \left|\frac{\sigma}{\varepsilon}\right| \leq 1 \end{Bmatrix}$$

We use sigmoid function as the control law for input. Saturation and signum function gives a lot of chattering around the sliding surface and thus are avoided. The sigmoid control law is given by

$$f(\sigma) = \frac{2}{1 + e^{-a*\sigma}} - 1$$

The above function helps in smoothening the path followed by the plant, thus reducing chattering a lot and brings it around the sliding surface quickly.

Chapter 3

Simulations & Experimental Setup

After completing the mathematical modelling, various necessary parameters related to the vehicle were found out. Two types of vehicle model have been discussed in the previous chapter- Ackermann steering and differential drive. The simulations have been done for the Ackermann steering vehicle and sensitivity analysis of its various parameters has been found out. Apart from that, two motion controllers have been implemented in the simulation environment- PID controller and Sliding Mode Controller (SMC). Due to some issues in research facility, it was first necessary to perform the experiments on a smaller vehicle which is a differential drive and then move to the bigger Ackermann drive vehicle. So to validate our mathematical model and efficiency of motion controller, simulations in MATLAB environment are compared with CarSim results which provide gives data closer to the real world cars.

After the mathematical modelling, a differential drive robot has been setup for the autonomous navigation and collision avoidance. This chapter discusses the necessary software and hardware requirements to complete our setup. We also discuss the parameter estimation done for finding the parameters of our Ackermann drive vehicle.

3.1 Ackermann Vehicle Parameters Estimation

It is quite challenging to find the dynamic parameters of the vehicle such that a perfect, error-free simulation is achieved. But an applicable measure can be performed using some estimation. In order to calculate vehicle's mass (m) , moment of inertia (I_z) and centre of gravity (CG), split mass acting on each wheel is considered namely given by m_{fl}, m_{fr}, m_{rl}, m_{rr}. The mass on front axle (m_f) is sum of masses on front left (m_{fl}) and front right tire (m_{fr}). Similarly, mass on rear axle, $m_r = m_{rr}+m_{rl}$. Total mass of the vehicle is combined sum of above four masses.

CG is calculated by considering point masses acting on the rear and front axle. Its position is given as: from the front axle,

$l_f = l * (1 - \frac{m_f}{m})$ and from rear axle $l_r = l * (1 - \frac{m_r}{m})$.

Moment of inertia about z-axis is calculated by considering two point masses joined by a mass-less rod.

$$I_z = m_f * l_f^2 + m_r * l_r^2$$

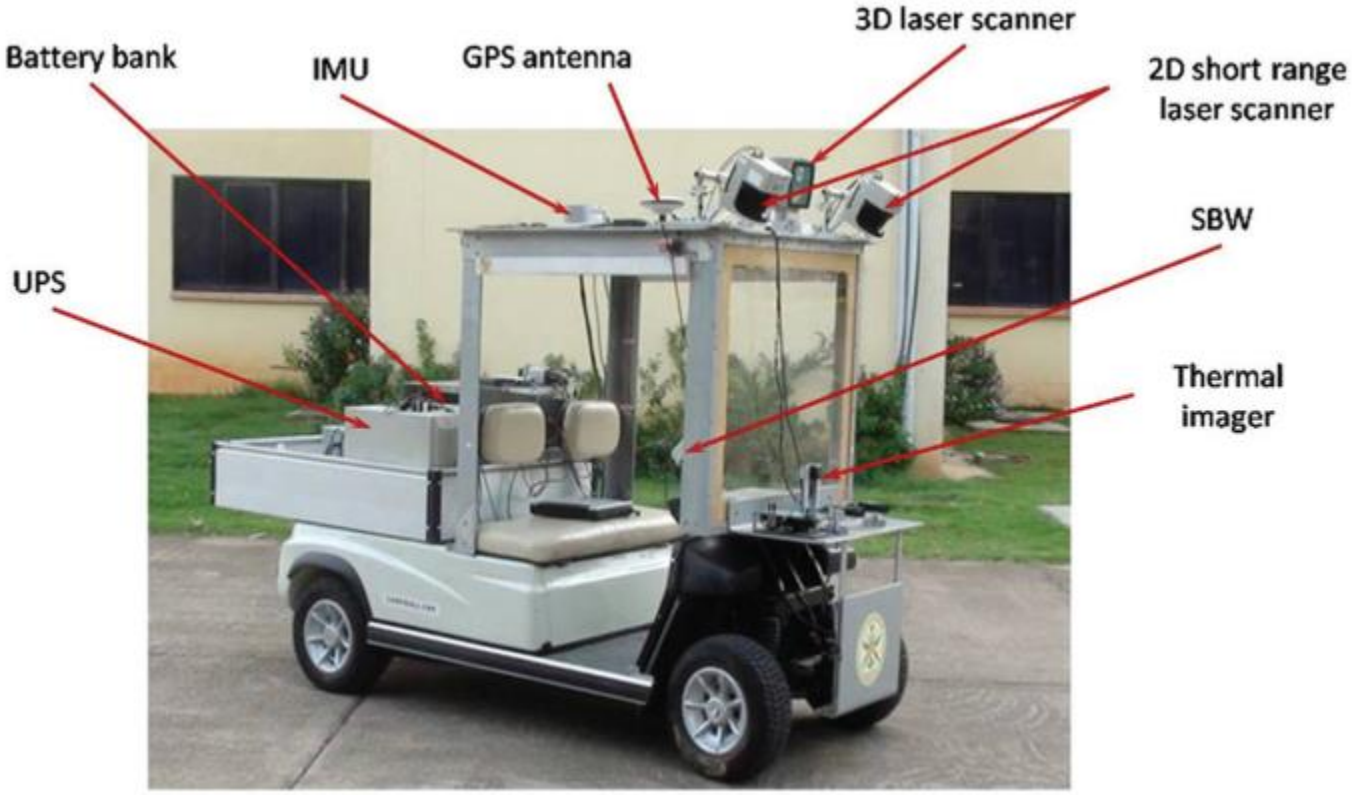


Figure 3.1 Test Vehicle

Table 3.1 Physical parameters of the vehicle

Parameters	Value	Units
l_f	1.31	m
l_r	0.62	m
m_{fr}	137	Kg
m_{fl}	158	Kg
m_{rr}	269	Kg
m_{rl}	360	Kg
I_z	932.4	Kg-m^2
l	1.93	M
W	0.9	M
C_f	134359	N/rad
C_r	134359	N/rad

Estimation of cornering stiffness of the vehicle

Accurate determination of the cornering stiffness of the tyres requires extensive experiments. It is necessary for the tyre manufacturer to print certain information such as the wheel radius, the tyre width,

the aspect ratio (which is the ratio of the tyre section height to the tyre width expressed as a percentage), the load index, the speed rate, the type of tyre construction and the maximum allowed inflation pressure, on the tyre sidewall. From this basic tyre information, the cornering stiffness can be estimated by using a mathematical tyre model. [27]

Considering the above model, final relation between the C_α and tire parameters is given by:

$$C_\alpha = \frac{8Ebw^3}{L[2\pi(r_w + wa) - L]} \tag{3.1}$$

Where L is given by

$$L = 2(r_w + wa)\sin\left[\cos^{-1}(1 - \frac{swa}{r_w + wa})\right]$$

In the above formulation, E is the compression modulus of the belt, b is the thickness of the tyre belt, rw is the radius of the wheels, w is the width of the belt, a is the tyre aspect ratio (the tyre section height divide by the tyre section width), L is the contact patch length and s is the unitized percentage of the sidewall vertical deflection when loaded.

Table 3.2 Parameter of the tyre

Parameter	**Value**	**Units**
Aspect ratio a of the tyre	0.5	-
Thickness b of the tyre belt	0.015	m
Compression modulus E of the belt	$27.3*10^6$	N/m^2
Radius rw of the wheel	0.254	m
Unitized percentage s of the sidewall vertical deflection when loaded	15%	-
Width w of the belt	0.205	m

Steering Actuator Dynamics

In order to maintain the vehicle heading angle, the steering wheels of the UGV should follow the command signals received from the vehicle controller and maintain synchronization with the steering actuator. To perform the simulation of the system, an appropriate actuator model needs to be established. Therefore, the transfer function model is derived analytically from the electrical and mechanical governing equations of the motor that is obtained from first principles. To model the steering actuator, visualization, as shown in Figure 2.5, is considered. The steering motor torque T is related to the armature

current i, by a torque constant Kt. The governing equations based on the Newton's law combined with the Kirchhoff's law are

$$J\frac{d^2\phi}{dt^2} + b\frac{d\phi}{dt} = K_t i$$

$$L\frac{di}{dt} + Ri = V - K_b\frac{d\phi}{dt} \tag{3.2}$$

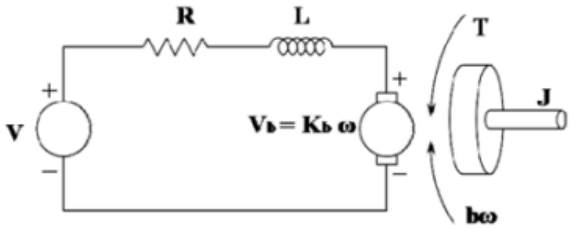


Figure 3.2 Schematic diagram of the steering actuator system

A steering controller has also been implemented in the vehicle model. This controller drives the steering motor which steers the front wheels based on controller commands. That's why required steering angle has to be found out so that the vehicle follows the desired path. The steering control system uses a DC motor (Maxon RE-40) with 156:1 reduction gear ratio to control heading direction. The motor is further connected to the steering shaft with the help of spur gears with GR of 1.47 giving 156*1.47≈230 rotations for one rotation of steering shaft. The standard rack and pinion gear has a GR of 15.5 and thus eventually making GR between the front wheel and steering shaft to 230*15.5≈ 3555. A negative servo feedback is implemented as per the tracking requirement. At steady state, angular velocity of steering motor remains constant. Specification of the steering actuator has been specified in table 3.3.

Table 3.3 Specifications of steering actuator

Parameters	Value	Unit
Terminal resistance (R)	0.317	ohms
Terminal inductance (L)	0.0823	mH
Torque constant (Kt)	30.2	mNm/A
Speed constant	317 (33.196)	rpm/V (rad/sec / V)
Back emf constant (Kb)	0.0301	V/rad /sec
Rotor inertia (J)	138	$g.cm^2$
Speed / torque gradient	3.33	rpm / mNm
Nominal speed (N)	6930	Rpm
Nominal torque (T) (max. Continuous torque)	170	mNm
Nominal voltage	24	V

3.2 CarSim Simulation environment

Considering the advantages of CarSim software discussed previously we simulated our vehicle model by putting the dynamic parameters as that of the vehicle.

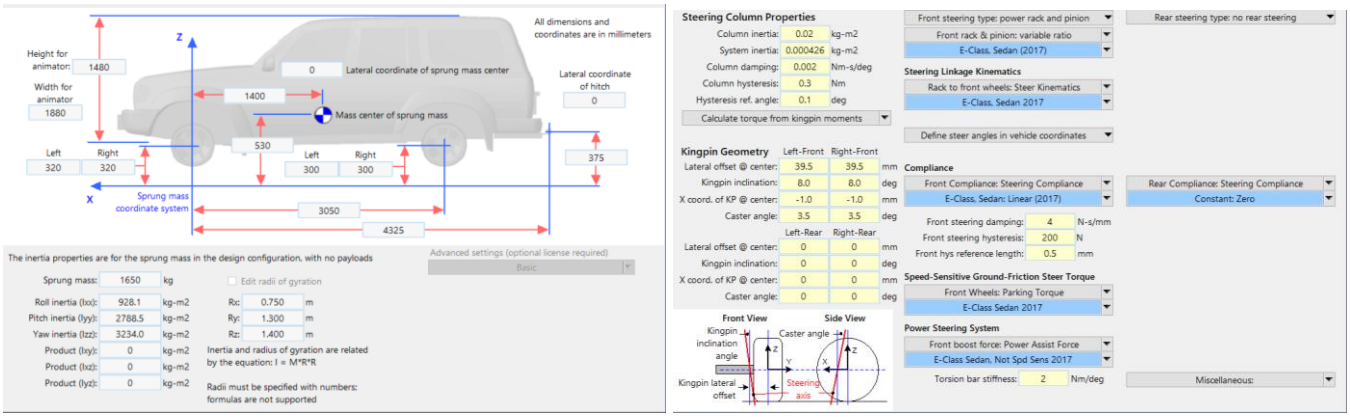


Figure 3.3 Simulation environment setup in CarSim

After which simulation were performed and plots for various parameters were recorded.

I implemented PI and SMC controller and then compared the results of MATLAB simulation with the CarSim simulation. The vehicle modelling was replaced by the vehicle model exported to Simulink from CarSim. This vehicle model includes all the parameters which are not taken into the account in MATLAB. We have included suspension, brake system , road conditions, aerodynamic drag and various

other practical parameters. After exporting the model, we implemented the simulation with the existing steering actuator model.

Using CarSim for simulation doesn't take a lot of trouble but finding how to implement can take really a long time if you haven't used it before.

We need to first choose the vehicle model.

- In my case I have chosen a D-class minivan 2017 as my plant system. Then I went to its properties and change its inertial and dimensional properties close to the vehicle I am currently working on.
- I haven't changed its aerodynamic, suspension and brake systems. However the tire parameters are changed according to the current vehicle.
- Then for running the vehicle in a certain road condition, the procedure was changed to *Driving->Constant speed with Roughness***.** The roughness was taken as 0.9. It can be defined as our own path but that can be troublesome. It's better to define the path trajectory in MATLAB if you want plot and compare the data. Here you can also choose the time till which you want simulations to be run.

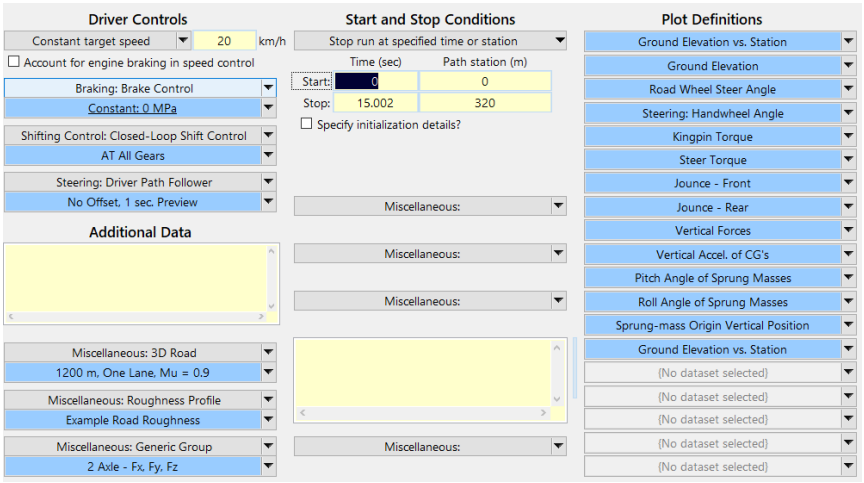


Figure 3.4 Setting up driver controls and variables

At the same time we can also change speed at which vehicle is running and defines the plots you want to see in the video.

- Then come to the home window and since we are going to work with MATLAB not the built-in solvers. So change the *No Linked Library* to *Models: Simulink* by clicking on the Models dropdown. At this time since we are working with just steering controls so go to dataset *Steering and Steering Controls* and select *Four wheel Steering System***.**
- Go to the *Four Wheel Steering System* tab.

Here we can set our own time step by which we are running the vehicle. I have taken same as simulations performed in the MATLAB. Through *Import Channels* we decide the inputs to the vehicle model. I have taken input as the steering commands to the front two wheels.

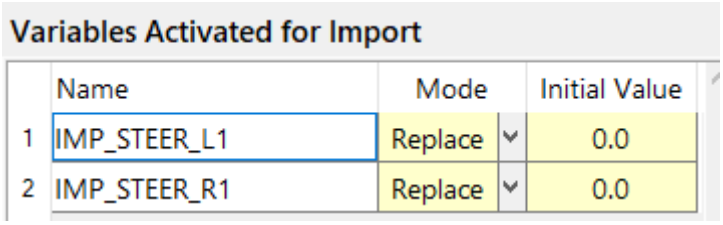


Figure 3.5 Import variables

Through *Export Channels*, we can change the output given by the vehicle model. In my case I have taken four outputs steer controls to the front wheels, yaw rate (from here I calculated yaw angle by integration in MATLAB) and longitudinal velocity to the vehicle.

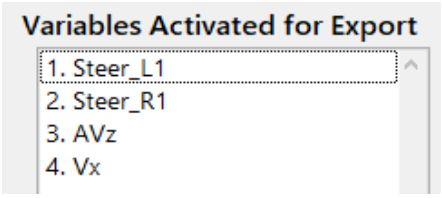


Figure 3.6 Export variables

After deciding the inputs and outputs, our vehicle model is ready to be used in Simulink. Just select *Send to Simulink.* Then just copy the vehicle model and paste it in your own Simulink model and use it as normal transfer function block. If the block parameter of CarSim S-function is not set then first set it as follows:

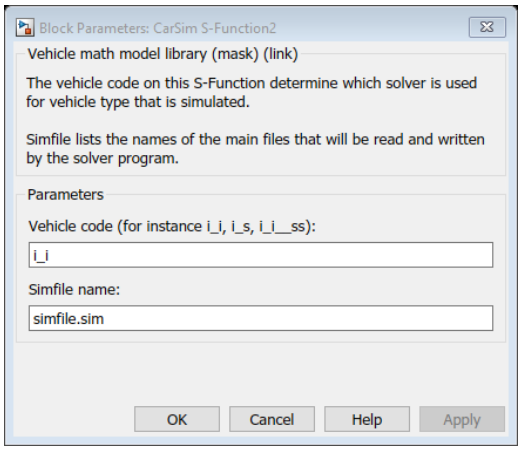


Figure 3.7 Setting up vehicle block in Simulink

3.3 0x delta Series Robot (Differential-drive robot)

These robots have been developed by NEX Robotics. They are a 4 wheel differential drive robot designed mainly for research purposes. Different kind of sensors can be mounted on it for getting a better idea of its surrounding. It has a very good performance on-board computer with high computational power. Its mechanical design is efficient enough to support all sorts of terrain and heavy payload. Rugged construction and safety critical design make it an ideal choice for outdoor environment.

These robots are widely used in research for autonomous navigation and mapping of the environment. It provides a perfect hardware platform for testing of various machine learning algorithms. It provides the facility for different add-ons as well. We can mount 2D/3D LiDAR, thermal camera and optical camera for better sensing of the environment. It also supports robotic arms attachment which could give the robot a more useful purpose like multi-floor navigation through lift.

The on-board PC supports is Intel Core i5 processor, 8GB DDR3 RAM. It supports Wi-Fi connectivity. It supports various programming languages like C, C++ and python libraries. ROS packages are installed for communicating with the remote systems.

For object manipulation, it supports 5-6 axes with payload 2-6 kgs at 900-150 mm. They also support gripper and arms with precise control over them. But currently we are not using these optional features on our robot.

Other technical specifications are mentioned below:

Table 3.4. 0x delta series robot specifications

Parameters	**Values**
Dimensions (L x W x H)	58cm x 62cm x 30cm
Weight	34kg
Max Payload Weight	15kg
Wheel Diameter	26 cm
Axel Length	70 cm
Ground clearance	8.8cm
Maximum Speed	5 km/h
Vertical Obstacle	10cm
Power Supply	24V battery
Working Time	4 hours
Communication Secure	128 bit encrypted

Figure 3.8 0x Delta series robot

3.4 Robot Operating System (ROS)

The Robot Operating System (ROS) is a framework for writing robot software, an open-source, meta-operating system for the robot. It provides same kind of services would be expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. Basically ROS is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. This basically allows for code reuse, and improves the quality of the code by having it tested by a large number of users and platforms.

Advantages of ROS are:

- Distributed computation
- Software reuse
- Rapid testing
- Supports various programming languages

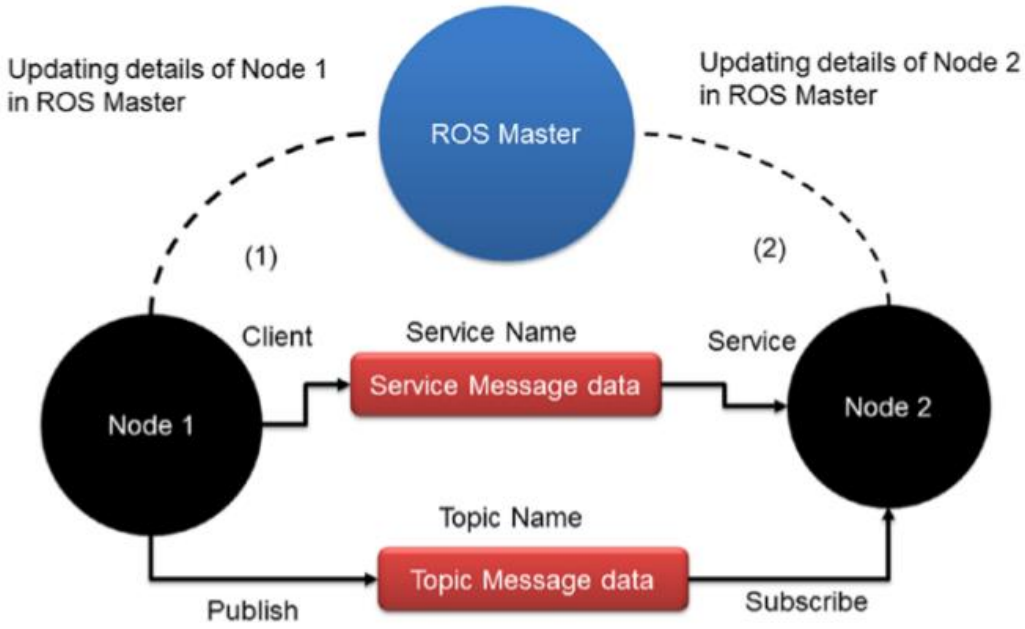


Figure 3.9 ROS communication block diagram

Figure 3.4 shows two programs marked as node 1 and node 2. When any of the programs start, a node communicates to a ROS program called the ROS master. The node sends all its information to the ROS master, including the type of data it sends or receives. The nodes that are sending a data are called publisher nodes, and the nodes that are receiving data are called subscriber nodes. The ROS Master has all the publisher and subscriber information running on computers. If node 1 sends particular data called "A" and the same data is required by node 2, then the ROS master sends the information to the nodes so that they can communicate with each other. The ROS nodes can send different types of data to each other, which includes primitive data types such as integer, float, string, and so forth. The different data types being sent are called ROS messages. With ROS messages, we can send data with a single data type or multiple data with different data types. These messages are sent through a message bus or path called ROS topics.

3.5 Sensors

Wheeled mobile robots need to sense the environment using sensors in order to autonomously perform their mission. Sensors are used to cope with uncertainties and disturbances that are always present in the environment and in all robot subsystems. Mobile robots do not have exact knowledge of the environment, and they also have imperfect knowledge about their motion models (uncertainty of the map, unknown motion models, unknown dynamics, etc.). The outcomes of the actions are also uncertain due to non-ideal actuators. The main purpose of the sensors is therefore to lower these uncertainties and to enabl estimation of robot states as well as the states of the environment.

3.5.1 Sensor Coordinate transformation

Sensors that are mounted on the robot are usually not in the robot's centre or in the origin of the robot's coordinate frame. Their position and orientation on the robot is described by a translation vector and rotation according to the robot's frame. Those transformations are needed to relate measured quantities in the sensor frame to robot coordinates. With these transformations we can describe how the sensed direction vector (e.g., accelerometer, magnetometer) or sensed position coordinates (e.g., laser range scanner or camera) are expressed in the robot coordinates. Furthermore, mobile robots are moving in space, and therefore, their poses or movements can be described by appropriate transformations.

tf is a package that allows the user keeps track of multiple coordinate frames over time. *tf* package performs the function of maintaining the connection between coordinate frames in a tree structured manner buffered in time with every other frames, and allows the user to transform points, vectors, etc. between any two local coordinate frames related to the robot at any desired point in time. This package is necessary for transformation of sensor coordinates to the robot frame.

Static_transform_publisher is the command line tool to publish a static coordinate transform to tf using an x/y/z offset in metres and yaw/pitch/roll in radians or in quaternions. *View_frames* creates a PDF graph for the current transform tree for graphical debugging.

3.5.2 Orientation and Rotation

Orientation of some local reference frame (e.g., sensor) according to the reference frame (e.g., robot) is described by a rotation matrix R:

$$R = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

(3.3)

where u, v, w are orthonormal unit vectors of a local coordinat system. The rows of R are components of body unit vectors along the reference coordinate unit vectors x, y, and z. The elements of matrix R are cosine of the angles among the axis of both coordinate systems; therefore, matrix R is also called the direction cosine matrix or DCM. Basic rotation transformations are obtained by rotation around axis x, y, and z by elementary rotation matrices:

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{bmatrix}$$

(3.3)

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

(3.4)

$$R_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(3.5)

where Raxis (angle) is rotation around the axis for a given angle.

3.5.3 Pose Measurement Methods

There have been several methods to estimate robot pose in the environment using sensors. *Dead reckoning* (also called deduced reckoning) gives the estimate of the robot's (equipped with relative positioning sensors) current position from the known previous position and relative to the measured displacements from the previous position. These increments in position and angle (distance and orientation) are calculated from measured linear and angular speeds over the completed time and heading. Common to these approaches is the use of path integration to estimate the current pose; therefore, the accumulation of different errors (error of integration method, measurement error, bias, noise, etc.) typically appears.

Odometry is used to estimate the robot pose by integration of motion increments that can be measured or gathered from applied motion commands. Relative motion increments are in mobile robotics usually obtained from axis sensors (e.g., incremental encoder) that are attached to the robot's wheels. Using an internal kinematic model these wheel rotation measurements are related to the position and the orientation changes of the mobile robot. The position and orientation changes given in the known time period between successive measurements can also be expressed by robot velocities. However, due to the integral nature of odometry, cumulative error occurs. The main source of the error consists of the systematic and nondeterministic error sources. The former includes errors due to approximate kinematic models (e.g., wrong radius of the wheel), error due to accuracy of applied integration method, and measurement error (unknown bias), and the latter includes slippage of the wheels, noise, and the like.

In *Navigation using environmental features*, Features are located at known locations. Therefore, their observation can improve knowledge about mobile robot location (lower location uncertainty). The list of features with their locations is called a map. This requires either an offline learning phase to construct a map of features or online localization and map building (*simultaneous localization and mapping [SLAM*]). The former approach is methodologically simpler but impractical in practice especially for larger environments. It requires the use of some reference localization system to map observed features,

or this must be done manually. The latter approach builds a map simultaneously while localizing, and the main idea is to localize from observed features that are already in the map and storing newly observed features based on localized location.

3.5.4 Sensors used in the Robot

Wheel Encoders

They are electro-mechanical devices that converts linear or angular position of a shaft to an analog or digital signal, making them the linear/angular transducer, measure position or speed of the wheels, integrate wheel movements to get an estimation of the position i.e. odometry.

Optical encoders function by making use of a rotor disc composed of either plastic or glass that consists of several irregular patterns with transparent and opaque areas that ca be detected as the disc attached with wheel rotates between a light source and an optical detector. Like the magnetic encoder, the simplest configuration usually uses just one sensor and has one half of the disc transparent and the other half opaque. But to obtain higher resolution, the disc is usually divided into many more segments (often in concentric rings) with two or more sensors.

LiDAR

LiDAR are also known as the Light Detection and Ranging, basically are laser range finders. It is a time of flight sensor that achieves significant improvements over the ultrasonic range sensor owing to the use of laser light instead of sound. It uses the light waves or the light source to measure the distance between the object. The laser and the detector are the two of the main components inside this LIDAR system.

The laser in the pulse form is targeted on the object and the reflection or the scattering from the object it is being measured by the detector. And at the detective side, the timing between the transmitted pulse and the received pulses is measured.

Using the light waves, we can even detective very small objects. So using the LIDAR system, we can achieve the much more Precision. So, because of its precision, these LIDAR systems are used for the 3D mapping of the object or even for the surface scanning of the object.

Sensor specifications:

Table 3.5 LiDAR sensor specification

Application	Indoor
Integrated application	Protective field evaluation with flexible fields, output of measurement data, Protective field evaluation with flexible fields, output of measurement data
Working range	0.05 m ... 25 m
Aperture angle	Horizontal (270°), Horizontal (270°)
Angular resolution	0.33°
Number of field sets	16
Enclosure rating	IP67
Colour	White
Digital outputs	3 (PNP, to display a protective field violation, additional 1 x “Device Ready”)
Scanning frequency	15 Hz
Switching mode	PTP

Figure 3.10 Sick LMS 100 2-D LiDAR

IMU

An Inertial Measurement Unit, commonly known as an IMU, is an electronic device that measures and reports orientation, velocity, and gravitational forces through the use of accelerometers and gyroscopes and often magnetometers. An IMU is a specific type of sensor that measures angular rate, force and sometimes magnetic field. IMUs are important components of the inertial navigation systems used in aircraft, autonomous ground vehicles, unmanned aerial vehicles (UAVs) and other unmanned systems, as well as missiles and even satellites. IMU data is processed by computers to track position through dead reckoning. Common applications for IMUs include control and stabilization, navigation and correction, measurement and testing, unmanned systems control, and mobile mapping.

It consists of three motion sensors:

- Accelerometer: These are the most commonly used type of motion sensor. It measures acceleration (change of velocity) across a single axis. Accelerometers measure linear acceleration in a particular direction whereas an accelerometer can also be used to measure gravity as a downward force. Integrating acceleration once gives an estimate for velocity, and integrating again gives you an estimate for position.
- Gyroscope: Accelerometers can just measure linear acceleration but can't measure twisting or rotational movement. Gyroscopes, however, measure angular velocity about three axes: pitch (x axis), roll (y axis) and yaw (z axis). While a gyroscope has no initial frame of reference, user can combine its data with data from an accelerometer to measure angular position.
- Magnetometer: It measures magnetic fields. It can detect fluctuations in Earth's magnetic field, by measuring the air's magnetic flux density at the sensor's point in space.

In our case Xsens MTI100 IMU has been mounted to the robotic platform. The datasheet of the same has been attached in [26].

3.6 Setting up ROS environment with the robot

First ROS packages need to be installed in the Ubuntu system. Since we are using Ubuntu 16.0,ROS-kinetic version is installed in the system. The tutorials related to ROS and its packages installation is provided in the roswiki page which is easily understandable. All the dependencies need to be installed for the proper function of the software.

The robot "0x delta" has an in-built PC which supports Wi-Fi connectivity. All the sensors like 2-D LiDAR, IMU are connected to this PC. This PC has Intel-core i5 processor, 8GB RAM, which controls the vehicle motion. The ROS packages are built within to communicate with the remote system. To

communicate with the external device we connected this PC to a router through Ethernet cable. All the sensors used in the robot have some IP address which should be in the same series. In our case the IP addresses of various components are:

Robot PC: 192.168.101.31

Router: 192.168.101.3

LiDAR: 192.168.101.35

Velodyne (3-D LiDAR): 192.168.101.36

Remote System: 192.168.101.32

After setting up the necessary addresses of these components, we also have to configure the environment in the remote PC. To do this, go to /etc folder in the main directory and then open a terminal window:

$ sudo gedit hosts

In this text file add the guest IP addresses of the system along with their name. One more thing to setup is the .bashrc file. To setup this just open the bashrc file and check that if local host address is as per the host system or not.

After this the robot could be seen connected to the remote system through the Wi-Fi. To see that the robot is connected and our host system is receiving data from the robot, type the below command in the terminal of the host system:

$ ping 192.168.101.31

Here 192.168.101.31 is the robot address and if connected properly the terminal should be displaying 64 bytes sent in some milliseconds. After this we need to run the robot and for that we have to access the robot PC from our host system. To do that, type the below command:

$ ssh –X 192.168.101.31

Then it will ask for the password of the robot PC after which we can access the robot PC's terminal. Then all the packages built within the robot can be easily run remotely using our own host system.

For connecting the sensors to the robot, one should make sure that the port ID of the sensor is known. To get an idea of that, I connected each sensor one-by-one and checked its port number by opening */dev* directory where the port no is in the form of *ttyUSB0X*, where X varies as the number of ports is increased. These port numbers are required to be set while running the driver of sensors.

Chapter 4

AUTONOMOUS NAVIGATION

Autonomous navigation in mobile robots means that they are capable of navigating in an unknown and an uncontrolled environment. An autonomous robot performs its tasks with avery high degree of autonomy. A fully autonomous robot has the capability to obtain information about its environment in which it is navigating, work for longer period without human intervention and to avoid situations that could be harmful to its environment or to itself. An autonomous navigation system is an on-board, integrated suite of sensors and technology that enables perception, path planning and autonomous navigation capabilities. Success in navigation requires success at the four basic aspects of navigation: perception, localization, and cognition and motion control.

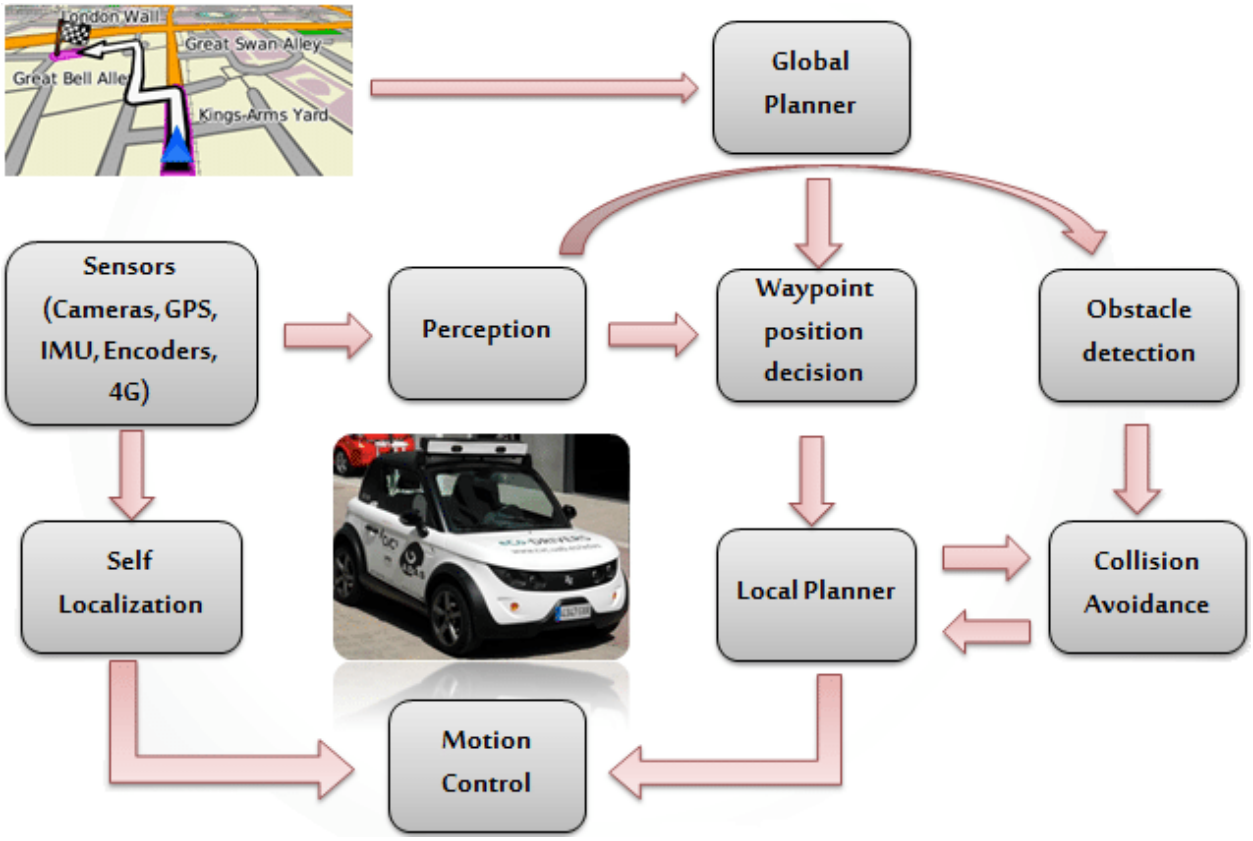


Figure 4.1 Basic Layout of Autonomous Vehicle

4.1 Navigation Stack

It is a set of algorithms that use the sensors of the robot and the odometry, and we can control the robot using a standard message. It can move our robot without problems (for example, without crashing or getting stuck in some location, or getting lost) to another position.

We would assume that this stack can be easily used with any robot. This is almost true, but it is important to tune some configuration files and write some nodes to use the stack.

The robot must satisfy some conditions before it uses the navigation stack:

- The navigation stack is able to only handle a differential drive and holonomic wheeled robots. The shape of the robot needs to be either a square or a rectangle. However, it can also perform certain things with biped robots, such as robot localization, as long as the robot does not move sideways.
- It is required that the robot continuously publishes information about the relationships between all the joints and sensors' position.
- The robot must send information with linear and angular velocities.
- A planar laser must be mounted on the robot to create the map and perform localization. Otherwise, we can also generate something equivalent to several lasers or sonars, or we can project the values to the ground plane if they are mounted in another place on the robot.

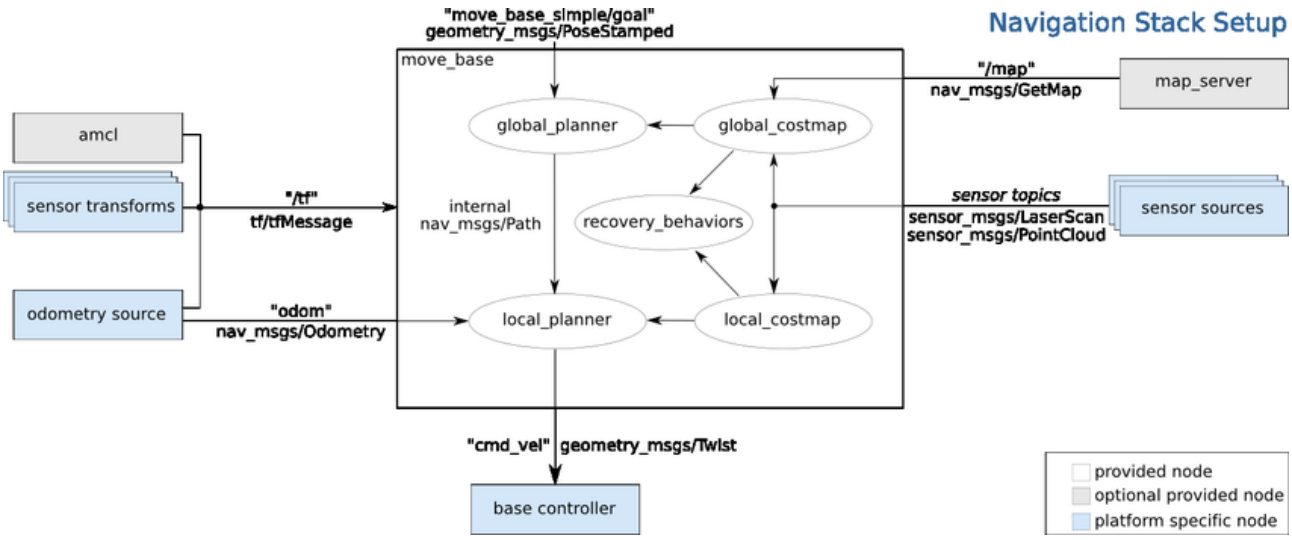


Figure 4.2 ROS Navigation Stack [ref: roswiki]

The following diagram 4.2 shows us how the navigation stacks are organized. We can see three groups of boxes with colours (gray and white) and dotted lines. The plain white boxes indicate those stacks that are provided by ROS, and they have all the nodes to make our robot really autonomous.

The navigation stack work properly when the transform relationships between different frames are published properly. To check that the transforms are published correctly one should open rqt_tf_tree using the below command:

$ rosrun rqt_tf_tree rqt_tf_tree

The tf relation should show graph as given in figure 4.3.

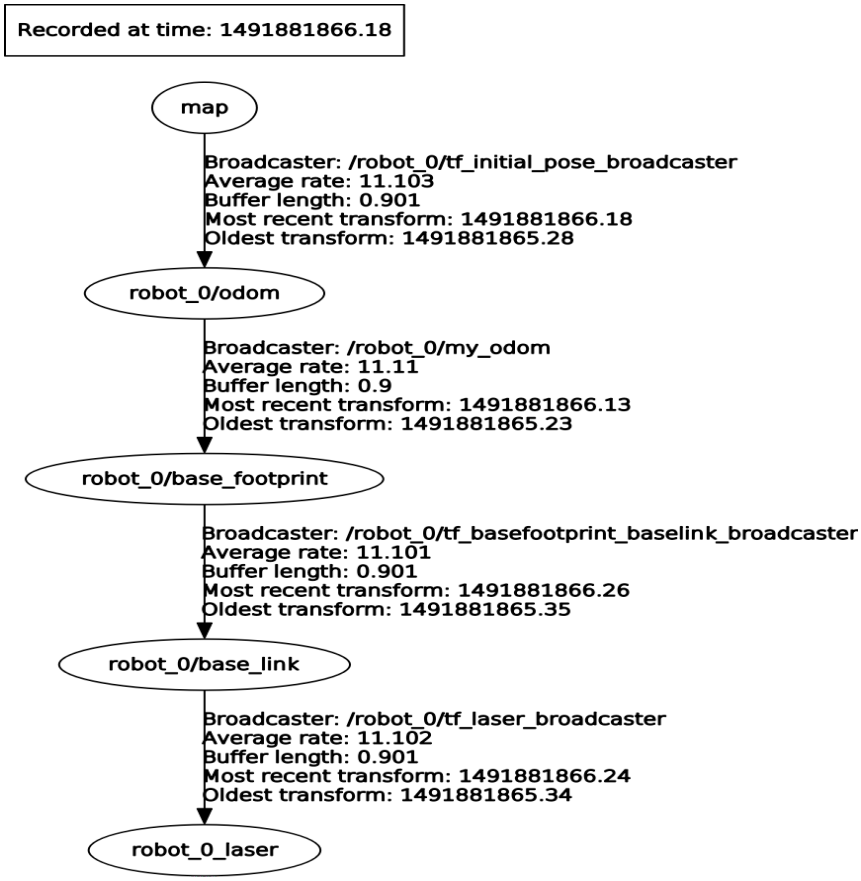


Figure 4.3 transformation tree for the navigation stack

Here, map represents the coordinate frame fixed to the map.

odom – the self-consistent coordinate frame using the odometry measurements only. The map → odom transform is published by amcl or gmapping.

base_link – the base link of the robot, placed at the rotational centre of the robot.

Laser- it represents the laser sensor scan.

4.2. Software Packages

ros0xrobot package

This package provides ROS interface for robot bases in 0x series. 0xRobotCpp library from Nex Robotics supports the ROS interface. Information from the robot base, velocity and acceleration control, is executed via a ros0xrobot node, which publishes topics providing data received from the robot's embedded controller by 0xRobotCpp library, and this sets desired velocity, acceleration and other commands in robot when new commands are received from command topics.

It contains the parameters which defines the robot kinematics. It has the wheel diameter, axel length, counts per revolution for the robot. It has ros0xrobotnode which contains the kinematic equation for our robot. It subscribes to cmd_vel to receive new velocity commands. By using these it publishes the position of the robot using the pose topic. However, this package is only for communicating with 0x series robots from nex Robotics.

lms1xx package

This package works with Sick LMS 1xx laser range finders. It publishes the laser scan data under the topic /scan. It has the parameters to connect with the device using its host name or IP address.

Rviz

Rviz is a simulator based on ROS in which we can visualize every kind of sensor data in the 3D environment, like a kinect camera sensor data by mounting it in the Gazebo model or the LiDAR sensor data to visualise the obstacles in 3D. From the laser scan data, we can build a map and it can be used for auto navigation. Rviz gives access and graphically represent the values obtained from camera image, laser scan etc.

By panel tab we get to view different tabs. The display window is used to views the different topics currently published. We can add those topics by clicking on add options in the display window. By views window we can adjust how to look over the rviz screen.

Costmap 2D

This package implements a 2D costmap that takes in the sensor data from the world, builds 2D or 3D occupancy grid of the data obtained from the LiDAR and inflates costs in a 2D costmap based on the the grids occupied on the map and a user specified inflation radius to include the critical distance from the obstacle. This package also allows support for map_server based initialization of a costmap, rolling window based costmap and parameter based subscription to and con figuration of sensor topics.

This package configures the environment and tells the robot where it can navigate in its environment. It assigns values to the occupancy grid in the maps to know the occupied, unoccupied and unknown regions. It uses sensor scan data and information, in our case a 2-D LiDAR data, to store information about the position of obstacles in the form of static map using costmap_2d::Costmap2DROS object which provides purely a purely two dimensional interface to its users.

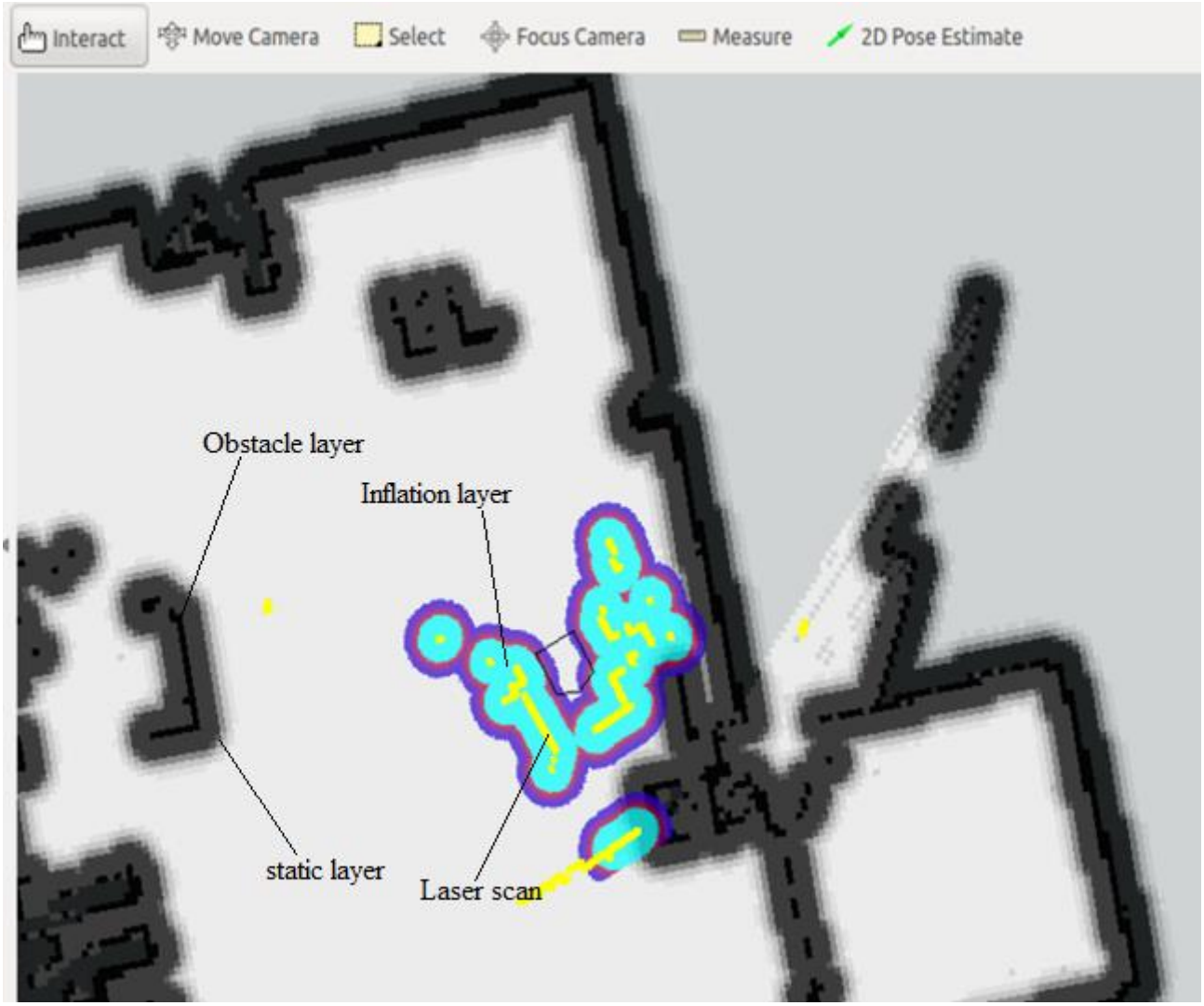


Figure 4.4 Cost map representing different layers

In figure 4.4, we have a pentagonal robot footprint which represents the size of the robot. The footprint significantly decides the path taken by robot when a goal point is given. The costmap_2d::Costmap2DROS object maintains a lot of functionality using LayeredCostmap which is used to keep track of different kind of layers. The costmap_2d::Costmap2D class implements the basic structure to store and access the 2D Costmap representing the obstacles in the map.

There are specific symbols which assign the cost values related to robot. The sensors are used to marks the cells in the maps and assign them the required cost values.

- “LETHAL”: represents an actual obstacle in the cell. In fig 3, the dark black thick lines represent such cost values. In our case its cost value is assigned as “255”.

- “INSCRIBED”: it means that a cell is less than the robot's inscribed radius away from an actual obstacle. So the robot is bound to be in collision with some obstacle if the robot centre is in that particular cell that is at or above the inscribed cost provided by the algorithm.
- “FREESPACE”: it means there is nothing which can hinder the robot motion. It is assigned a value of “0”.
- “UNKNOWN”: it means we don’t have sufficient information about that area. The robot doesn’t have idea about the obstacle present in that area.

In our case, each cell is given three types of cost values, i.e., each cell can be either free, occupied, or unknown. Each of the above status has a special numerical cost value assigned to it based in which it is project into the costmap. Columns that have a certain number of occupied grids are assigned a costmap_2d::LETHAL_OBSTACLE cost, columns consisting of certain number of unknown cells (see unknown_threshold parameter) are assigned a costmap_2d::NO_INFORMATION cost, and other columns are assigned a costmap_2d::FREE_SPACE cost.

Apart from that, for converting the sensor frame to base frame of the robot, the costmap_2d::Costmap2DROS makes extensive use of the tf. It also uses map_server package so that a user-generated static map can be used for informing the robot about its environment.

There are two divisions of costmaps in ROS:

- *Global costmap*- it is used for global navigation. In fig 5, the dark black thick lines represent the global costmap. These obstacles have been detected by moving the robot in the environment. The obstacles were detected using the laser scan and the cells in the map are marked as occupied and given a high cost values. The robot is expected not to make a path through those areas. The attributes defined in global costmap are defined in *costmap_common_params.yaml* and *global_costmap_params.yaml* files. This has been depicted in the below figures.
- Local costmap: this is used for local path planning during the robot reaching its goal. In fig 5, the yellow lines depict the local cost map present in the map. These are obtained by the current laser scan readings and cells are marked as per the given commands. The parameters for local costmap are defined on the local_costmap_params.yaml and costmap_common_params.yaml files.

Layer Specifications: There are three most common layers used in the costmap2d ros package:

- Static map layer: The static map incorporates mostly unchanging data from an external source, especially a map. In our case the robot is first allowed to know its surrounding by teleoperation and scanning the environment by a 2-D LiDAR. It subscribes the topic /map and gets the information about the occupancy of the cell.

- Obstacle layer: The obstacle and voxel layers incorporate information from the sensors in the form of PointClouds or LaserScans. The obstacle layer tracks the robot in two dimensions, whereas the voxel layer tracks in three. The costmap subscribes to the laser data and updates according to the obstacles detected by it. This uses point_cloud topic to update the costmap periodically.
- Inflation layer: it assigns the cost values to the cells according to their distance from the obstacles. In figure 5, we observe that both static and dynamic obstacles are inflated by bright colours. This inflation has been done for the safety of the vehicle, i.e., to avoid collision from the obstacles. However there is also one possibility that if the inflation radius is increased, we might lose some short and easy paths that a robot can take to reach the goal. Therefore the inflation radius should be carefully decided in the parameters. There is one parameter, cost_scaling_factor, defined in the inflation layer. A scaling factor is applied to get cost values during inflation. The cost function is computed as given in figure 4.5 for all grids in the costmap further than the inscribed radius distance given by costmap and less than the inflation radius distance away from a real obstacle.

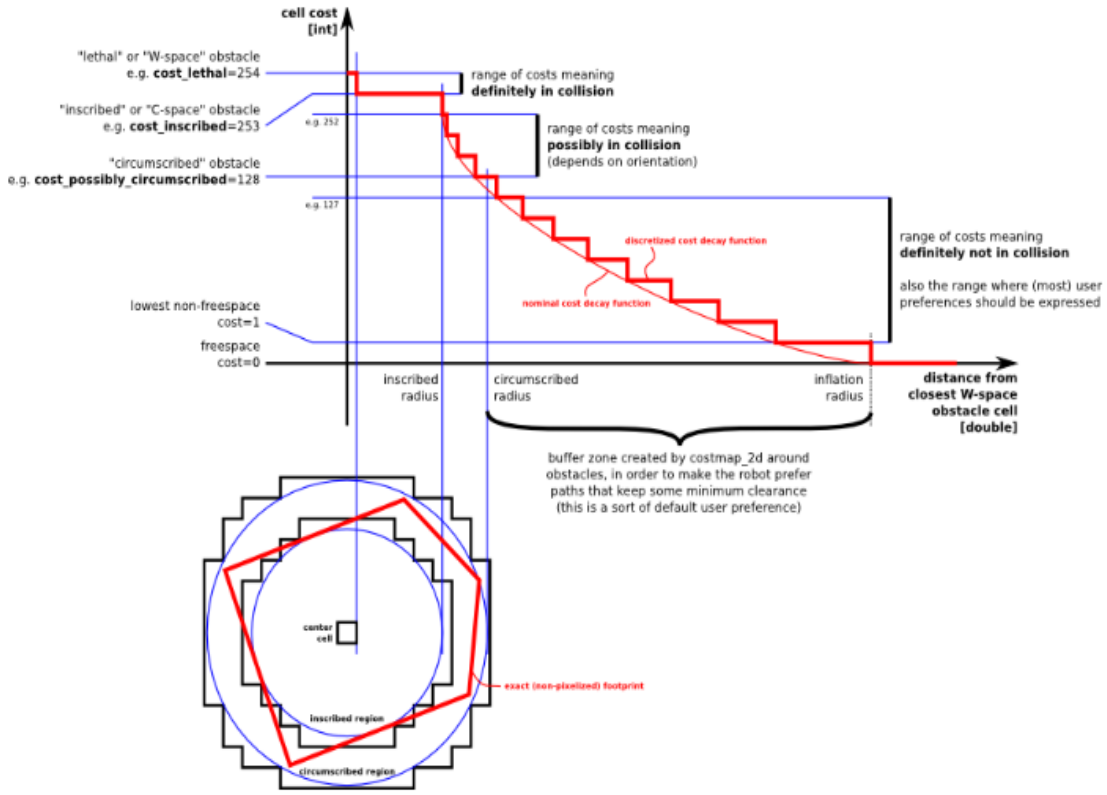


Figure 4.5 Inflation Layer (ref: roswiki)

Configuring the Costmaps

The robot moves through the map using two types of navigation—global and local.

- The global navigation creates paths for a goal in the map or a far-off distance.

- The local navigation creates paths in the close distances and avoids obstacles, for example, a square area of 3 x 3 meters around the mobile robot.

These modules use costmaps to keep every kind of the data on our map. The global costmap gives information about the global navigation and the local costmap for local navigation.

The costmaps have parameters to configure the behaviours, and they have common parameters as well, which are configured in a shared file. Configuration of Costmap basically consists of three files where we can setup different parameters. These files are as mentioned here:

- costmap_common_params.yaml
- global_costmap_params.yaml
- local_costmap_params.yaml

Configuring the common parameters

The obstacle_range and raytrace_range attributes are used to indicate the maximum distance that the sensor is able to read and introduce any other new information in the costmaps provided by navigation stack. The obstacle_range is used for the obstacles. If the robot detects an obstacle around or less than 2.5 meters in our case, it will project the obstacle in the costmap. The raytrace range is used to clean/clear the costmap and keeps on updating the free space in the map as the robot navigates. One thing to be noted is that we can only detect the data of the laser or sonar with the obstacle, we are not able to perceive the entire obstacle or object itself, but these simple approaches are enough to deal with obstacle measurements. This greatly helps to build a complete map and localize the robot.

The footprint attribute indicates to the navigation stack the geometry of the robot. It will be used to keep the right distance between the obstacles and the robot, or to know if the robot can move through a door keeping a safe distance. The inflation_radius attribute defines how much should be the minimal distance between the geometry of the robot and the obstacles.

The below line configures the sensor's frame and the uses of data:

laser_scan_sensor: {sensor_frame: laser_base_link, data_type: LaserScan, topic: /base_scan/scan, marking: true, clearing: true}

In the above code, sensor frame has been defined as "*laser base link*" and the message received from the LiDAR has a certain datatype *LaserScan* which his published on the topic name .*base_scan/scan*.

The laser configured is used to add and clear obstacles detected by LiDAR in the costmap. For example, we could add a sensor with a very wide range to detect obstacles and then another sensor like ultrasonic sensors to navigate and clear the obstacles around the environment. The topic's name has been configured

in the above line. It is important that we configure it the best, because the navigation stack could wait for another topic and all this while, the robot might be moving around which could then crash into a wall or an obstacle.

Configuring the global costmap

The global_frame and the robot_base_frame attributes define the transformation matrix between the map and the robot. This transformation is for the global costmap.

We can configure the frequency of updates for the costmap. In this case, it is 1 Hz. The static_map attribute is established for the global costmap to see a map. The map server is used to start the Costmap with the maps configured. If we aren't using a static map, then this parameter is set to false.

Configuring the local costmap

The update_frequency, global_frame, static_map and robot_base_frame parameters are the same as described in configuration of the global costmap file. The publish_frequency parameter defines the frequency by which the information is published. The rolling_window parameter keeps the costmap centered on the robot when it is navigating in its environment.

The transform_tolerance parameter configures the maximum latency for the transforms, in our case it is 0.2 s. With the help of planner_frequency parameter, we can configure the rate in Hz at which we have to run the planning loop. The planner_ patience parameter configures how long the path planner will wait in an attempt to find a valid plan where no obstacles are found, before space around it is cleared.

The dimension and the resolution of the costmap with the width, height, and resolution parameters are configured in this file.

SLAM Gmapping

Simultaneous localization and mapping, or SLAM for short, helps us to create a map using a mobile robot that navigates through its environment while using the map it creates. SLAM is the algorithm that works for robot mapping or robotic cartography. An area is considered in which the robot is allowed to navigate, but at the same time, it needs to figure out where its own self is located in the place. The process of SLAM considers a complex array of computations, algorithms and sensory inputs to navigate through a previously unexplored environment or to remap a previously known environment. SLAM helps to enable the remote creation of GPS data in areas where the environment is too dangerous or congested for humans to get into.

In a related way, a SLAM robot tries to map an unknown environment while figuring out where it is at. The complexity arrives from doing both these things at once. The robot has to know its position before

answering the question of what the environment looks like. The robot also needs to figure out where it is at without the benefit of already having a map. SLAM (Simultaneous localization and mapping), developed by Hugh Durrant-Whyte and John L. Leonard, is a process of solving this problem using specialized algorithms and techniques.

The basic requirement of SLAM is a range measuring device like SONAR or LiDAR which provides the method for knowing the environment around the robot. A more commonly used form of measurement is a laser scanner such as LiDAR. Laser scanners are quite easy to use and are very accurate. However, they are also extremely costly. There are other options, though. Sonar can also be used, and this device is quite useful for mapping environments under the water bodies but has a lower range than LiDAR. Camera devices can also be used for SLAM. These optical readers come in 2D or even 3D formats. The measurement device used depends on various variables, including preferences, costs, and availability.

Another very important component in the SLAM process is to acquire data about the environmental surroundings of the robot. Just like a human, the robot considers landmarks to determine its current position using its sensors, the laser, sonar, or whichever sensors have been used. A robot uses different landmarks measured using sensors for different environments. However, there are certain conditions for landmarks used in SLAM. Firstly, all these landmarks should be stationary. A robot is not able to determine its own location if a nearby landmark is continuously moving. Additionally, landmarks should be particular and easily distinguishable from the surrounding areas. These landmarks also need to be plentiful and should view from many different angles.

Once the robot has sensed a landmark through the laser scan, it can then determine its own position by extracting the sensory input through LiDAR and then identifying the different landmarks marked previously. A method has to be placed in order for the robot to do this. This landmark extraction is generally completed in a variety of ways from algorithms like Spike extraction to scan-matching. The important factor to remember in our case is that the robot requires a way to identify a landmark. Robots also use information from previously scanned landmarks and match them up with each other in order to determine its location.

The GMapping package is used to create maps while our robot navigates in a given environment. It uses Simultaneous Localization and Mapping(SLAM) to produce a 2D map from laser scan data. The robot_state_publisher publishes the laser scan transformation from laser scan to base_link. The slam gmapping package is used to create a map for the robot. The gmapping node contains a lot of parameters. Important ones to follow are:

- particles- defines the number of particles in the particle filter.
- xmin,ymin,xmax,ymax- defines the map size in metres.

- deltamap- resolution of the map
- base_frame- frame attached to the robot base.
- map_frame- frame attached to the map.
- odom_frame- frame attached to the odometry system.
- srr- odometry error in translation as a function of translation'
- srt- Odometry error in translation as a function of rotation.

AMCL

This package helps the robot to decide its current position and using this current position information the trajectory is planned. Using wheel odometry leads to inaccuracy in wheel spin calculation caused by lack of traction. So longer we run the vehicle, more inaccuracies we get in the pose estimate. So amcl help to compensate the mistakes committed by odometry. AMCL is a variant of the Monte Carlo Localisation (MCL) which is widely used for localization. MCL uses particles to localise the robot pose. It has several advantages over using Extended Kalman Filters (EKF) such as uses raw measurements (i.e. from lasers), is not reliant on gaussian noise, is memory and time efficient, and can perform global localisation. This node generally works with laser scans and laser maps. The AMCL package adaptively changes the number of particles used based on the robot motion, which has the advantage of reducing the computational overhead required. Each sample stores a position and orientation data which represents robot's current pose. Particles are all sampled randomly initially. When the robot moves, particles are resampled based on their current state as well as robot's action using recursive Bayesian estimation.

Topics subscribed:

- /scan- contains the laser scan information
- /tf- transforms incoming laser scans to the odometry frame.
- /initialpose- Mean and covariance with which to (re-)initialize the particle filter. This describes the parameters with which localization estimate amcl initially starts running with.
- /map- amcl package use this topic to get the map for laser-based localization.

Topics published :

- amcl_pose- gives the robot's estimated pose in the map with covariance.
- particlecloud- set of pose estimates being maintained by the filter.
- tf- publishes the transform from odom to map

We have observed that if some different pose estimate is given to the robot, it localizes itself at that location but is not able to get the correct estimate. We need to move the robot around to get to know about the landmarks detected in the map. Since the laser scan just provides data about the presence or

absence of obstacle. Similar occupancy grid might cause confusion about the current position of the robot. Besides, AMCL dynamically adjusts the number of particles over a period of time. This provides a computational advantage over the traditional algorithm. The AMCL has a node that will define its behaviour in RVIZ and parameters that will determine how effectively it can localize itself. In this stage, topics are remapped to fit the AMCL specifications and the truth map is published to the RVIZ program.

Parameters used in AMCL package are basically related to filter, laser and odometry.

Filter parameters: The first two parameters that relate to the filter are min_particles and max_particles. The maximum number of particles starts off at the initial and then AMCL dynamically decreases the number of particles toward the preset minimum. One important aspect of these parameters is that if the maximum is too high, it might be too computationally extensive and lead to a laggard system. Update _min_a and Update_min_d are defined as the translational movement required before performing a filter update and the rotation movement required before performing a filter update, respectively. Lowering these values would result in more updates and thus more iterations which will have the effect of increasing accuracy, while also increasing computation.

Laser parameters: Parameters for the laser can be changed to increase the amount of incoming data from this sensor. The parameter laser max beams determine the amount of beams in each scan of laser to be used when updating the filter. The laser max range parameter describes the maximum scan range of the laser. The laser likelihood max dist parameter determines the maximum distance to do obstacle inflation on map. Two other important parameters used to increase the accuracy of localization are the laser z hit and the laser z rand which are weights for the z hit and z rand part of the model.

Odometry parameters: Odometry parameters describe the movement of the robot and provide the AMCL information about this movement. The odom_model_type is the diff_corrected type. There are also 4 odom alpha parameters. Each (in order) specifies the expected noise in odometry's rotation estimate from the rotation component of the robot's motion, the expected noise in odometry's rotation estimate from the translational component of the robot's motion, the expected noise in odometry's translation estimate from the translation component of the robot's motion, and the expected noise in odometry's translation estimate from the rotation component of the robot's motion, respectively.

In amcl localization, the transform is published between the global frame and the odometry frame and thus accounting for the drift error that occur using dead reckoning. If the localization is done through odometry, there is a possibility of lateral drift error which can give wrong pose estimate of the robot as the error gets added cumulatively.

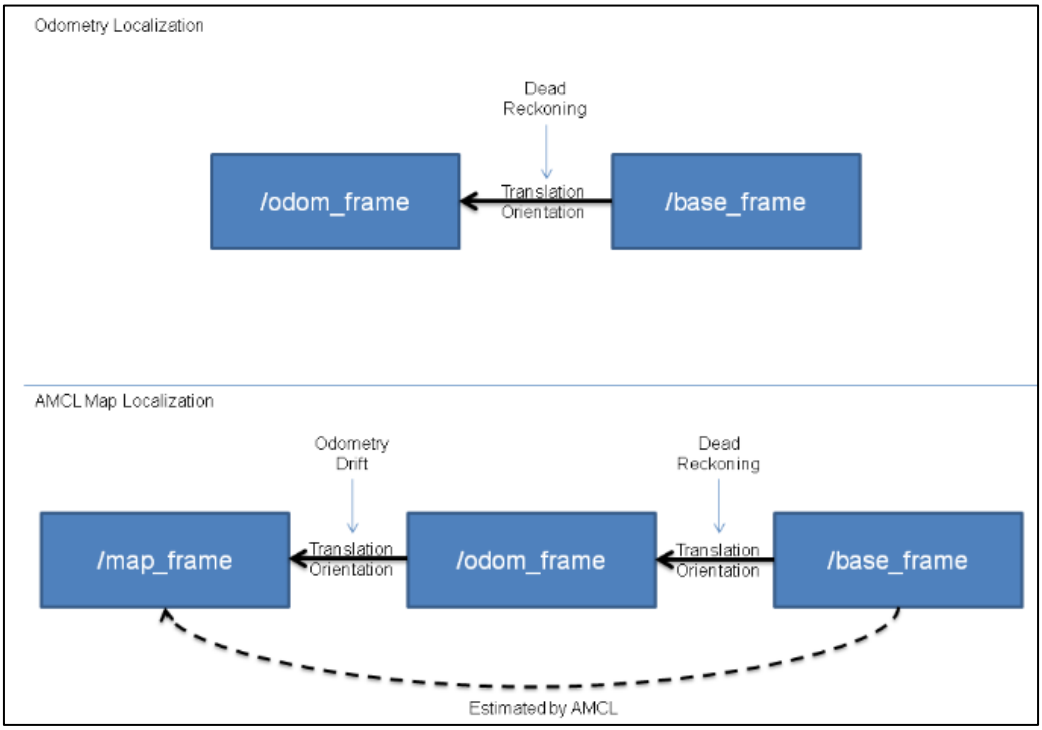


Figure 4.6 AMCL localization structure

Local Planner: Timed Elastic Band

The TEB primarily provides the time-optimal solution. The teb_local_planner package is a plugin to the base_local_planner of the 2D navigation stack. The underlying method, Timed Elastic Band locally optimizes the robot's planned path with respect to separation from obstacles, trajectory execution time and compliance with kino-dynamic constraints at runtime. The optimal trajectory is readily computed by solving a sparse scalarized multi-objective optimization problem. We provide weights to the optimization variables in order to know the behaviour in case of ant type of dubious objectives. This package subscribes topics odom to get the odometry information to plan out the velocities so that desired trajectory is followed. The topic obstacle provides custom obstacles as point-, line- or polygon-shaped one. It publishes the global plan which it is trying to follow currently. It also helps to visualize both the plans in rviz.

The teb_local_planner package helps the user to set parameters in order to customize the behaviour. These parameters are classified into several classes: robot configuration, goal tolerance, trajectory configuration, obstacles, optimization, planning in distinctive topologies and miscellaneous parameters.

Robot configuration parameters: It consists of velocity limits, acceleration limits, footprint model and turning radius as parameters. These parameters determine the vehicle velocity commands when a local path is planned. Generally the y-direction is neglected and is put to a value of zero. Here we also determine the footprint model to be used for optimizing the vehicle motion. The footprint model is quite

important as it determined the computation required for moving the vehicle. This package allows various types of footprints like point, circular, line, two circles and polygon.

Goal tolerance parameters: It consists of parameters that allow a certain level of discrepancies while reaching the goal. It is usual that the robot will show some errors while reaching a goal point. It can't be exactly move at the same point where the user desires it to be. If the tolerances are kept small, there is a possibility that the robot will keep on showing some motion even if we see that it has reached its goal. The xy goal tolerance is the position tolerance for the controller when achieving a goal. This can be lowered to increase the system accuracy, but will undoubtedly increase the time to reach the goal destination. the yaw goal tolerance is the tolerance in orientation when achieving a goal. Again, this can be lowered to increase system accuracy, but will undoubtedly increase the time to reach goal destination. We generally keep free_goal_vel as false so that the robot can arrive at the goal with maximum speed.

Trajectory configuration parameters: These parameters take care of the trajectory planning. max_global_plan_lookahead_dist defines the maximum length (cumulative Euclidean distances) of the entire set of the global plan considered to be optimized. Then we determine the actual length by the logical conjunction of the local costmap size and this maximum bound. We set this distance to zero or negative in order to deactivate this limitation. It decides what kind of line or arc has to be taken while deciding the path. It allows the planner to shrink the horizon temporary (50%) in case of automatically detected issues.

Obstacle parameters: It gives a clear idea of obstacle detection methods. min_obstacle_dist specifies minimum desired separation from obstacles. costmap_obstacles_behind_robot_dist limits the occupied local costmap obstacles taken into account for planning behind the robot. inflation_dist specifies buffer zone around obstacles with non-zero penalty costs.

Optimization parameters: It gives an optimization weightage for maximum allowed translational velocity, maximum allowed angular velocity, maximum allowed translational acceleration and maximum allowed angular acceleration. weight_obstacle defines optimization weight for keeping a minimum distance from obstacles, weight_inflation defines optimization weight for the inflation penalty, weight_kinematics_forward_drive provides optimization weight for forcing the robot to choose only forward directions, and weight_kinematics_nh provides optimization weight for satisfying the non-holonomic kinematics.

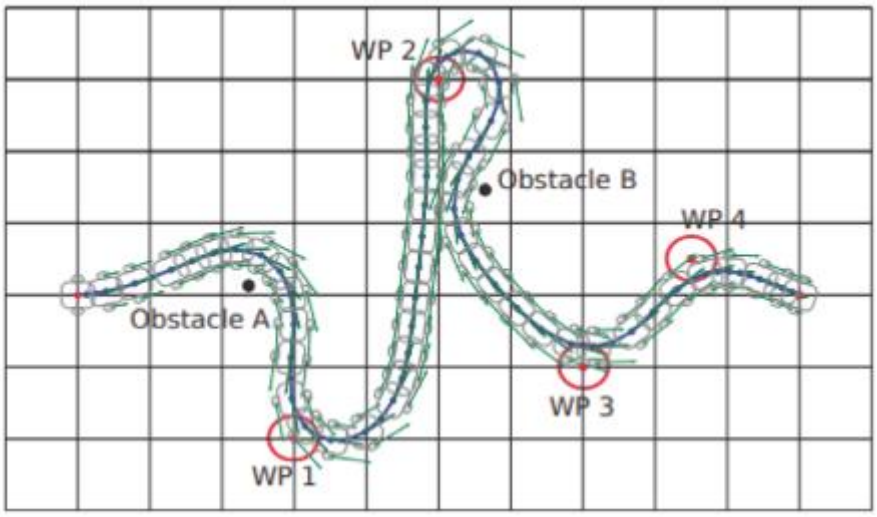


Figure 4.9 Timed Elastic Band

Local Planner: Dynamic Window Approach

The dwa_local_planner package allows a specific controller that drives a mobile base in the plane. This controller's job is to connect the path planner to the robot. The planner creates a kinematic trajectory for the robot using a map to get from a start to a goal location. Along the way, the planner keeps on creating, at least locally around the robot, a value function, represented in a grid map. This value function calculates the costs of traversing through the grid cells. The controller's job is to make use of this value function to determine dx,dy,dtheta velocities to send to the robot.

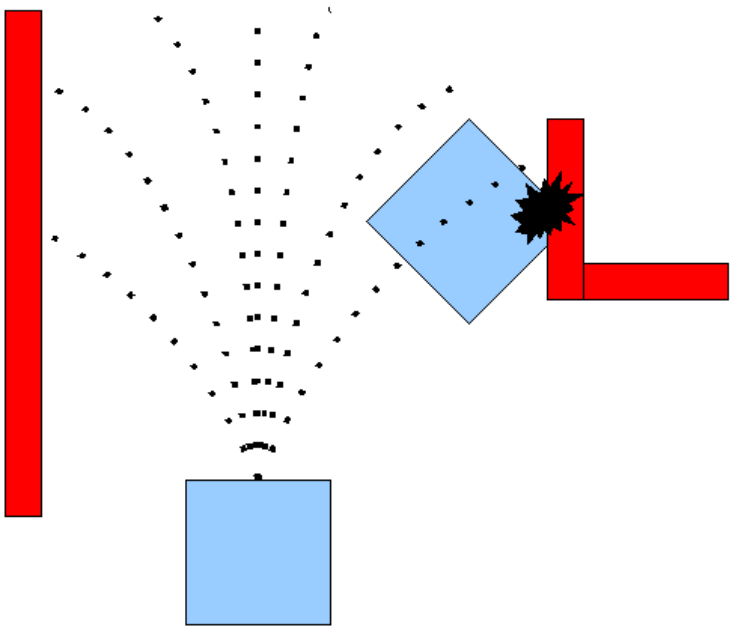


Figure 4.8. DWA planner trajectory

Velocity samples: vx sample, vy sample determine number of translational velocity samples to be taken in x, y direction for prediction. *vth sample* controls the number of rotational velocities samples. In most cases we prefer to set vth samples to be higher than translational velocity samples, because turning is generally a more complicated condition than moving straight ahead.

Trajectory Scoring: DWA Local Planner maximizes an objective function to obtain optimal velocity pairs. In implementation, the value of this objective function relies on three components: progress to goal, clearance from obstacles and forward velocity. The objective is to get the lowest cost. *path distance bias* is the measure for how much the local planner should stay close to the global path. A high value of this parameter makes the local planner prefer trajectories on global path. *goal distance bias* is the measure for how much the robot should attempt to reach the local goal, with whatever path. Experiments show that increasing this parameter makes the robot to be less attached to the global path and more to local. *occdist scale* is the measure for how much the robot should try to avoid obstacles. A high value for this parameter results in indecisive robot that stuck in place. Currently, we set path distance bias to 32.0, goal distance bias to 20.0, occdist scale to 0.02. They work well in simulation.

Goal distance tolerance:

- yaw goal tolerance (double, default: 0.05): This defines tolerance in radians for the controller in yaw/rotation when achieving its goal.
- xy goal tolerance (double, default: 0.10): This defines tolerance in meters for the controller in the x & y distance when achieving a goal.
- latch xy goal tolerance (bool, default: false) If goal tolerance is latched, if the robot ever reaches the goal xy location it will simply start rotating on the spot, even if it ends up outside the goal tolerance while it is performing the navigation part.

Global Planner: Navfn

This package implements a fast, interpolated navigation function that is used to create efficient plans for a wheeled mobile base through the navfn::NavFn class. It also provides a ROS Wrapper for this class via the navfn::NavfnROS object that adheres to the nav_core::BaseGlobalPlanner interface specified in the nav_core package. The navfn::NavfnROS object is also very helpful as a global planner plugin for the move_base node.

Parameters in this package are:

allow_unknown: This specifies whether or not to allow navfn to create plans that goes in unknown space.

planner_window_x: Specifies the x size of an optional window to restrict the planner to. This can be greatly used for restricting NavFn to work in a small window of a large costmap.

planner_window_y: Specifies the y size of an optional window to restrict the planner to. This can be greatly used for restricting NavFn to work in a small window of a large costmap.

default_tolerance: A tolerance on the goal point for the planner. NavFn tries to create a plan that is as close to the specified goal as possible but no further than default_tolerance away.

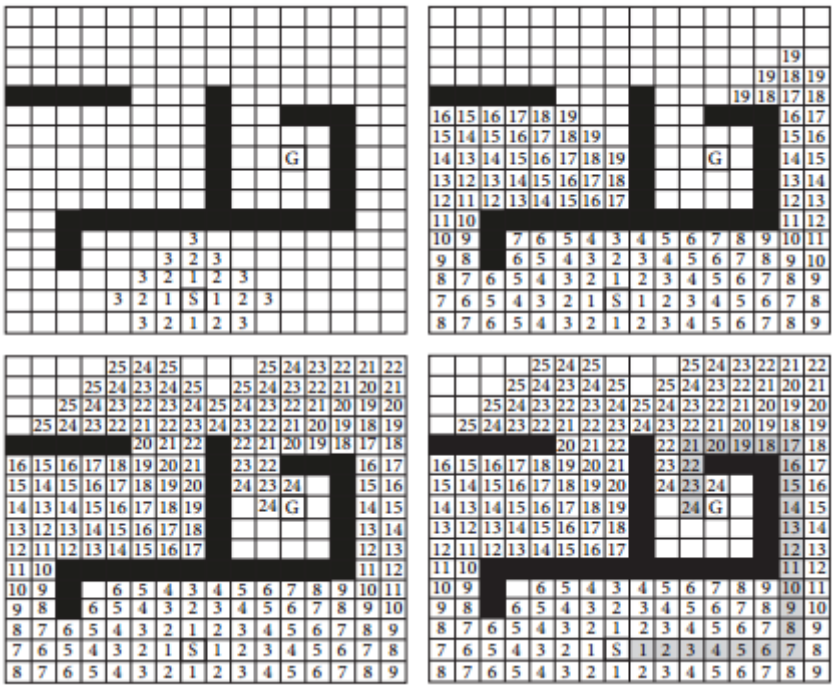


Figure 4.9 Djisktra algorithm

4.3 Dynamic Window Approach (DWA)

Dynamic Window Approach searches for commands controlling the robot by creating a search space for velocities. In our case these inputs are rotational and translational velocity and these can be considered as a velocity pair (v,w). Dynamics of the vehicle is included to search for just those velocities which are under dynamic constraints and thus reducing the search space.

Search Space- The possible paths of robot are uniquely defined by velocity pairs. Each path is having curvature given by $c = \frac{v}{w}$. this resulting trajectory should not intersect with the obstacle.

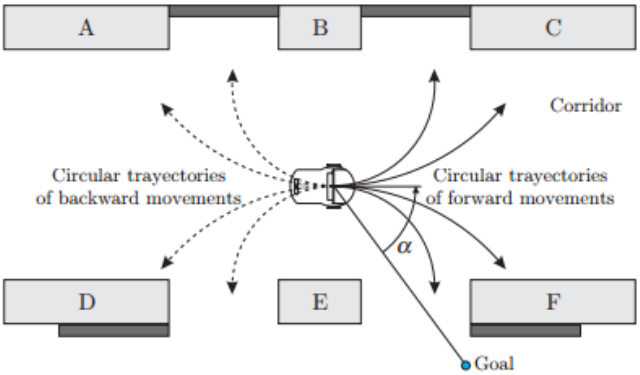


Figure 4.10 Illustration of a robot navigation environment using DWA

The complete set of admissible velocities (Va) is computed by a function Dist (v, w) that computes the distance to the nearest obstacle for a given trajectory.

Dist(v,w)= min(dist(v,w,obs))

$$\boldsymbol{V_a} = \left\{(\boldsymbol{v},\boldsymbol{w})\middle|\begin{matrix}\boldsymbol{v} \le \sqrt{\boldsymbol{2} * \boldsymbol{Dist}(\boldsymbol{v},\boldsymbol{w}) * \dot{\boldsymbol{v}}_{\boldsymbol{max}}} \\ \boldsymbol{w} \le \sqrt{\boldsymbol{2} * \boldsymbol{Dist}(\boldsymbol{v},\boldsymbol{w}) * \dot{\boldsymbol{w}}_{\boldsymbol{max}}/\boldsymbol{c}}\end{matrix}\right\} \tag{4.1}$$

where v_max and w_max are the maximum linear and rotational accelerations respectively. Vp is the whole space of all possible velocities for the robot. This is expressed as:

$$\boldsymbol{V_p} = \left\{(\boldsymbol{v},\boldsymbol{\omega})\middle|\begin{matrix}\boldsymbol{v} \in [\boldsymbol{0},\boldsymbol{v}_{\boldsymbol{max}}] \\ \boldsymbol{\omega} \in [-\boldsymbol{\omega}_{\boldsymbol{max}},\boldsymbol{\omega}_{\boldsymbol{max}}]\end{matrix}\right\} \tag{4.2}$$

Due to dynamic constraints of body, there is a set of reachable velocity given by dynamic window V_d:

$$\boldsymbol{V_d} = \left\{(\boldsymbol{v},\boldsymbol{\omega})\middle|\begin{matrix}\boldsymbol{v} = [\boldsymbol{v_c} - \dot{\boldsymbol{v}} * \boldsymbol{dt}, \boldsymbol{v_c} + \dot{\boldsymbol{v}} * \boldsymbol{dt}] \\ \boldsymbol{\omega} = [\boldsymbol{\omega_c} - \dot{\boldsymbol{\omega}} * \boldsymbol{dt}, \boldsymbol{\omega_c} + \dot{\boldsymbol{\omega}} * \boldsymbol{dt}]\end{matrix}\right\} \tag{4.3}$$

The resultant velocity search space is given by

$$V_r = V_p \cap V_a \cap V_d$$

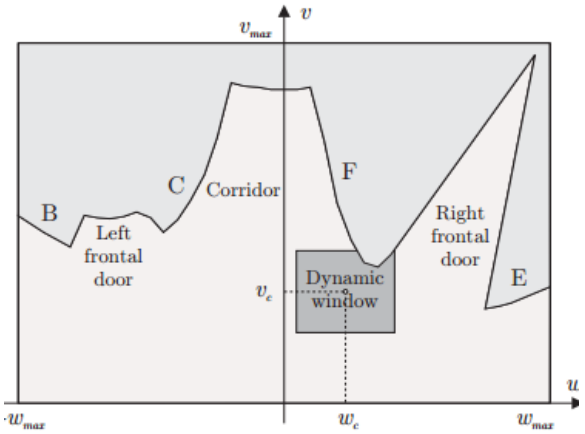


Figure 4.11 Velocity map

From this resultant velocity search space, the controller chooses a velocity pair that maximizes our objective function. The objective function can be written as a function of dependent parameters which brings the vehicle close to its target.

$$\boldsymbol{G(v,\omega) = \sigma * (\alpha * Dist(v,w) + \beta * speed(v,\omega) + \gamma * heading(v,\omega))}$$

(4.4)

Here σ is the scaling factor and α, β & γ are the tuning parameters.

The Dist function represents the distance to the nearest obstacle over a circular trajectory with a curvature given by the velocities (v, w). From equation 1, we can reverse calculate the dist function.

If the path is clear, then the vehicle should move at high speed so as to minimise time to reach the target. This is taken care by speed function. Though it has disadvantage like no orientation parameter is involved.

Speed(v)= v/v_{max}

The orientation parameter takes care of the anamoly caused by the speed function. It gives a measure of how much the vehicle is oriented towards the goal target.

Heading(w)= $1 - |c - \omega * dt|/\pi$

We can define some other function which can be suited for our platform. Another objective function can be defined as such:

$$G(v,\omega) = \mu_1 * \left(1 - \frac{|v - v_i|}{2 * v_{\max}}\right) + \mu_2 * \left(1 - \frac{|\omega - \omega_i|}{2 * \omega_{\max}}\right) + \mu_3 * dist(v,\omega)$$

(4.5)

4.4 KALMAN FILTER

There are two basic position-estimation methods widely used in navigation system, i.e. absolute and relative positioning. Absolute positioning uses navigation beacons, active or passive landmark, map matching, or satellite-based navigation signal, where absolute positioning sensors interact with dynamic environment. Relative positioning is usually based on odometry sensors, or inertial sensors.

There are two kinds of sensors: internal and external sensors. Internal sensor utilises physical variables that can be measured on the vehicle. Typical examples are gyroscopes, accelerometers, compasses, encoders. Except for compasses, internal sensors present a typical drift that affects long term estimates. However, short period measurements are quite accurate moreover, internal sensors give immediate responses.

External sensors keeps track of relationships between the robot and some natural or artificial reference objects: if some characteristics of the reference objects (for instance, their position in space) are known, it is possible, by means of proper computations, to estimate position and orientation of the robot with respect to its environment. The computation step makes an external sensor not continuous; nevertheless, estimation errors do not present drift, since precision doesn't depend on mission duration, but on position.

The type of internal sensors that are mostly used in navigation is odometry sensor. They are mounted on the robot's wheel shafts and register angular movements of the wheels. These angular rotations are then converted into linear movements. But this process has a very limited accuracy, for example, if slip occurred on the wheel, then the odometry will register the movement, but in reality, the vehicle may stay on its own position due to the lateral drift phenomena. In long run, due to the incremental motion of odometry, this error will keep on accumulating while processing the positioning of the robot. However one advantage of using odometry is that the data is continuously available.

GPS (Global Positioning System) is one of the external sensors which is used to give absolute position of the robot but is not consistently available. At the same time its frequency of publishing data is less than that of the odometry and IMU sensor.

Kalman filter uses the idea of using multi-sensor fusion for trajectory estimation by taking advantage of both internal and external sensors: in particular, position and orientation are estimated, in short period, by

internal sensors and their increasing errors are periodically limited by using external sensors. This also involves weighting both internal and external estimates.

The robot external kinematics has been earlier defined in eq. 2.7 :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos(\varphi(t)) & 0 \\ \sin(\varphi(t)) & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$$

4.4.1. Kalman filter algorithm

A kalman filter simply calculates state prediction given by mathematical model and measurement update from the sensors over and over again.

First we have a state variable given by $x = [\, x\ y\ \theta]^T$ which is given an initial condition. In our case the initial positional coordinated x,y is given by GPS and θ is given by IMU.

An uncertainty is given for the initial state by the covariance matrix **P**. In the one-dimensional case, the variance is defined as a vector, but now is matrix of uncertainty for all states.

$$P = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

This matrix is most likely to be altered during the filter passes. This gets changed in both the predicted and corrected steps. The Matrices needs to be initialized on the basis of the sensor accuracy. If the sensor is very accurate, we use small values. If the sensor is relatively inaccurate, large values needs to be used to allow the filter to converge relatively quickly.

The core of the filter is the dynamics matrix **A** which we should set up with great understanding of the physical context. We have previously defined in chapter 2 (eq. 2.11)

$$A = \begin{bmatrix} v(k) * T_s \cos\left(\varphi(k) + \omega(k) * \frac{T_s}{2}\right) \\ v(k) * T_s \sin\left(\varphi(k) + \omega(k) * \frac{T_s}{2}\right) \\ \omega(k) * T_s \end{bmatrix}$$

As the movement of the can also be disturbed, we introduce the process noise co-variance matrix. The filter information is obtained by this matrix, and how the system state can “jump” from one step to the next. If an acceleration command tries to affect the system state, then the physical dependence for it is described in Q. This Q matrix is a co-variance matrix having following elements:

$$Q = \begin{bmatrix} {\sigma_x}^2 & {\sigma_{xy}}^2 & {\sigma_{x\theta}}^2 \\ {\sigma_{yx}}^2 & {\sigma_y}^2 & {\sigma_{y\theta}}^2 \\ {\sigma_{\theta x}}^2 & {\sigma_{\theta y}}^2 & {\sigma_\theta}^2(0) \end{bmatrix}$$

The filter also need to be told what is to be measured and how it can relate to the state vector. This is done by computing matrix **H.**

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

If the sensors measure in a different steps or the size by detours, we need to map the relationships of the measuring matrix in a formula.

The measurement uncertainty measured by measurement noise covariance matrix **R** indicates how much one trusts the measured values of the sensors. If the sensor is very accurate, we use small values. For higher inaccurate sensors, large values should be used here.

$$R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

After initialization of the state vector with a position and velocity, the dynamics can be used to make an optimal prediction about the current location of the robot.

$$x_{t+1} = x_t + A$$

The co-variance also needs to be recalculated. In the predicted step uncertainty about the state of the system increase, as we have seen in the one dimension case. In the multidimensional case, the measurement uncertainty gets added up, so the uncertainty becomes more and more.

From the sensors we get current measurement values, with which an innovation factor (y) is obtained by using the measurements, the state vector with the measuring matrix.

$$S = H * P * H' + R$$

This determines the Kalman gain. This defines whether the readings or system dynamics should be more familiar.

$$K = P * H' * S$$

The Kalman Gain reduces if the readings (measurements) match the predicted system state. If the measured values are different, the elements of matrix K become larger.

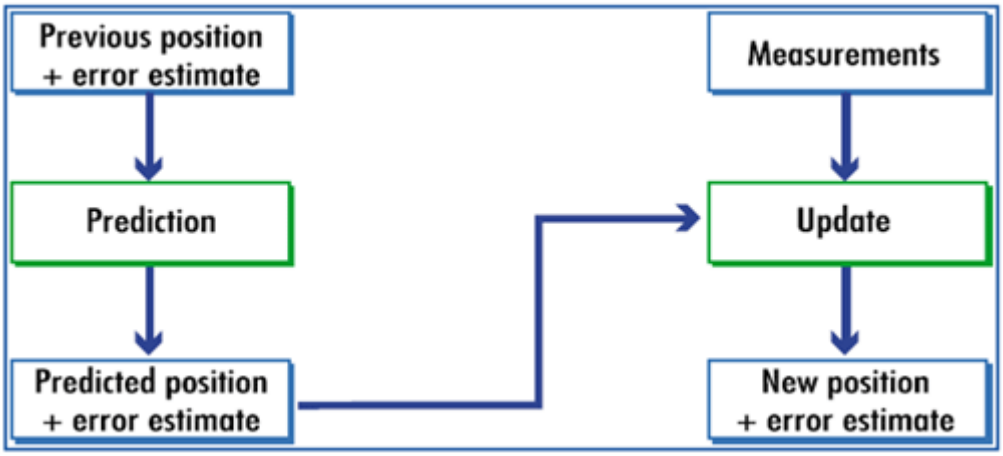


Figure 4.12 Kalman Filter basic working layout

Chapter 5

RESULTS AND DISCUSSIONS

As previously discussed the project has been completed on two phases: one of the part deals with the vehicle modelling and motion controller implementation while the other part deals with the development of autonomous vehicle and trajectory planning of the robot.

5.1 Vehicle modelling

The simulation considering various vehicle parameters in the mathematical model has been done. The model has been already formulated in chapter 2 and the simulation has been carried out in MATLAB.

Generally the autonomous vehicle moves at a very low speed considering the safety of the surrounding and the payload it carries. Besides, there can be cases which can't be predicted earlier in the simulation. So I have limited our speed up to 40 km/hr in the simulation and then have seen the changes.

5.1.1 Steering actuator consideration

There have been lot of papers published regarding the mathematical modelling of the vehicle where the steering actuator dynamics was ignored. But there are known errors due to the exclusion of the steering actuator. In real world, we don't change the vehicle steering angle by applying force to the wheel, rather we give voltage commands which turn the vehicle wheels by desired steering.

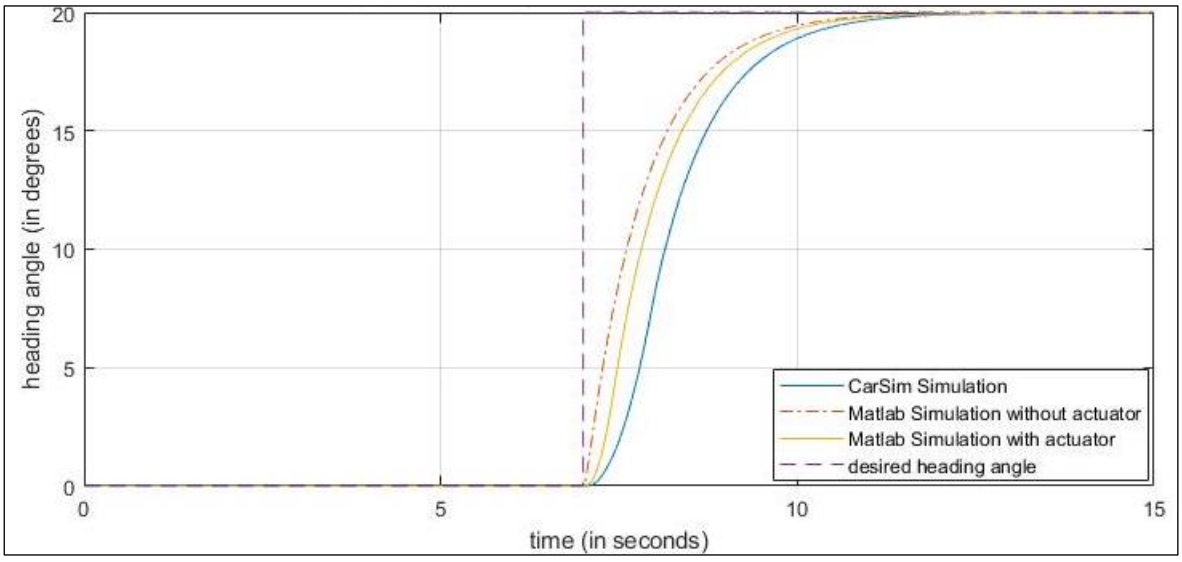


Figure 5.1 Comparison for 20˚ heading angle input when vehicle is moving at 3.2 m/s

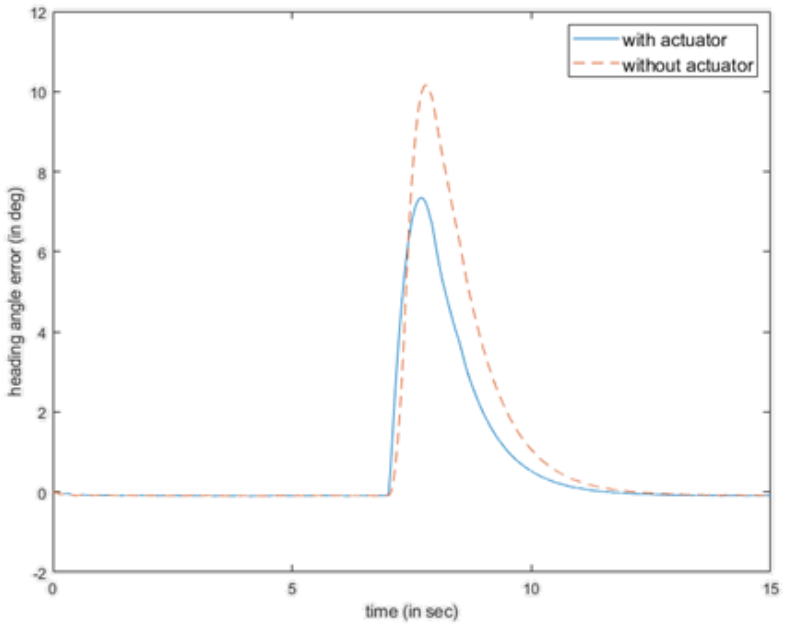


Figure 5.2 Heading angle error without actuator dynamics and considering actuator dynamics when the vehicle is moving at 3.2 m/s to achieve a desired heading angle of 20 degrees

We simulated the vehicle with and without taking the steering actuator dynamics into account and then compared the results with CarSim. It has been found out that the test vehicle takes 7 seconds to reach a constant velocity of 3.2 m/s. So, we gave the desired heading input to the vehicle starting from 7 seconds and then simulated the outputs.

In each test, the vehicle is allowed to run with a constant velocity of 3.2 m/s in a straight path and after 7 seconds, a desired step input of 20˚ heading angle was given to the vehicle. The results have been performed in CarSim and MATLAB. In MATLAB, we took two conditions. In first we considered the role of steering actuator dynamics and in other we neglected its presence. There has been significant deviation due to this assumption. With steering actuator in consideration, we have less values of settling time and less steady state errors. Without steering actuator, the settling time reaches to 4.5 seconds as compared to 2.5 seconds by taking steering dynamics into account as can be seen in figure 5.1.

In figure 5.1, we also observe a significant difference between the simulation results of CarSim and MATLAB. We observe when the vehicle is simulated in CarSim, we have high settling time which is comparable to a condition without actuator. These deviations can be attributed to the real life environments provided by CarSim. The overshoot and steady state errors seem to be reduced in the MATLAB simulation but in CarSim simulation, we have steady state error of 1.2% but there is no significant overshoot. In figure 5.2, we can see the overall picture. It has been obtained by subtracting the simulated and carsim results. We see that the heading error reaches to more than 10˚ when we don't take the steering dynamics into account. With steering actuator the error reduces to a value less than 7˚.

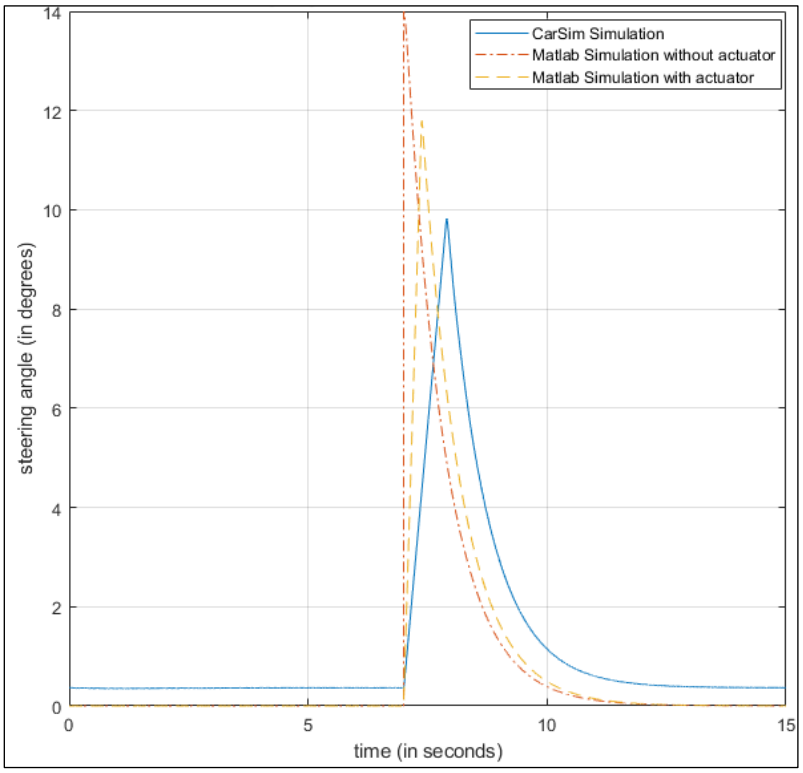


Figure 5.3 Steering angle comparison when 20˚ heading angle step input is given for vehicle running at 3.2 m/s.

In Fig.5.3, steering angle for the CarSim and MATLAB simulations has been compared. We observe that the simulation considering the actuator shows a maximum steering value of 20˚, while the CarSim simulation suggests a lower value, 9.5˚. By considering actuator we also come to find that it takes the vehicle an almost 2 seconds to reach the steering value of 20˚. This is relatable to real-life condition because we can't get an instantaneous output. If some input is provided, it takes some amount of time to give the output.

Table 5.1 Results for vehicle running at 3.2 m/s with 20˚ step input heading angle response

Scenario	**Heading angle response (in ˚)**		**Steering angle response (in ˚)**	
	RMS error	Steady state error	RMS error	Maximum steering angle
With actuator	1.1075	0.049	1.2215	11.4
Without actuator	1.7769	0.065	1.8984	14.05

5.1.2 Sensitive Analysis of Vehicle Parameters

Considering a J-turn manoeuvre for the vehicle, a sensitive analysis on cornering stiffness has been done. Generally there are three conditions when a vehicle performs a continuous rotation motion: oversteer, neutral steer and understeer.

Oversteer is what occurs when a car turns (steers) by more than the amount commanded by the driver. Conversely, understeer is what occurs when a car steers less than the amount commanded by the driver.

Since for different situation on real road cornering stiffness can change a lot, we have to model the vehicle such that it can sustain those situations.

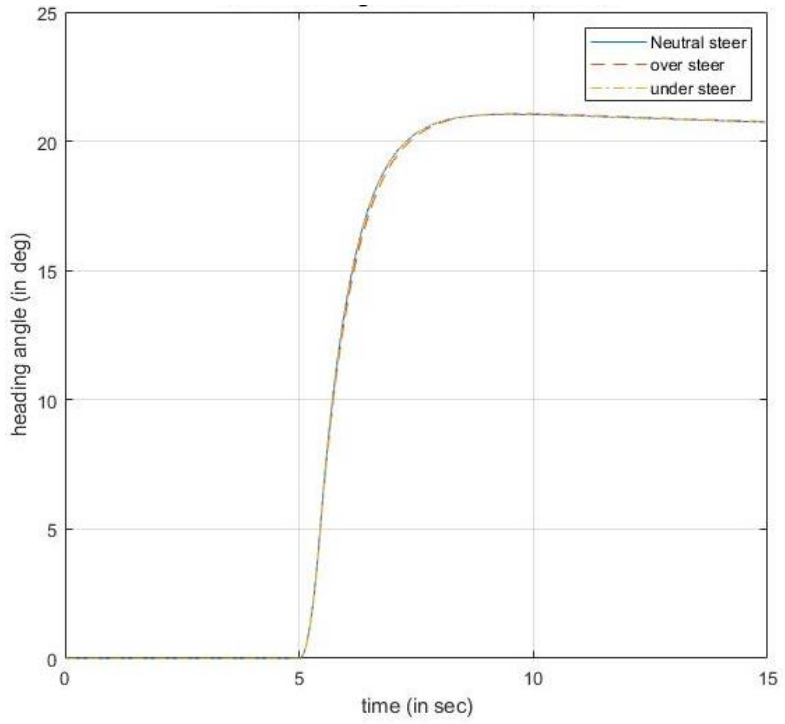


Figure 5.4 Effect of Cornering Stiffness on the vehicle

We observe that there is no such drastic change when we change the cornering stiffness value to incorporate these driving conditions. The graphs seem to follow the same lines. This is something which is actually predictable at lower velocities since the dynamics doesn't come into play at these velocities.

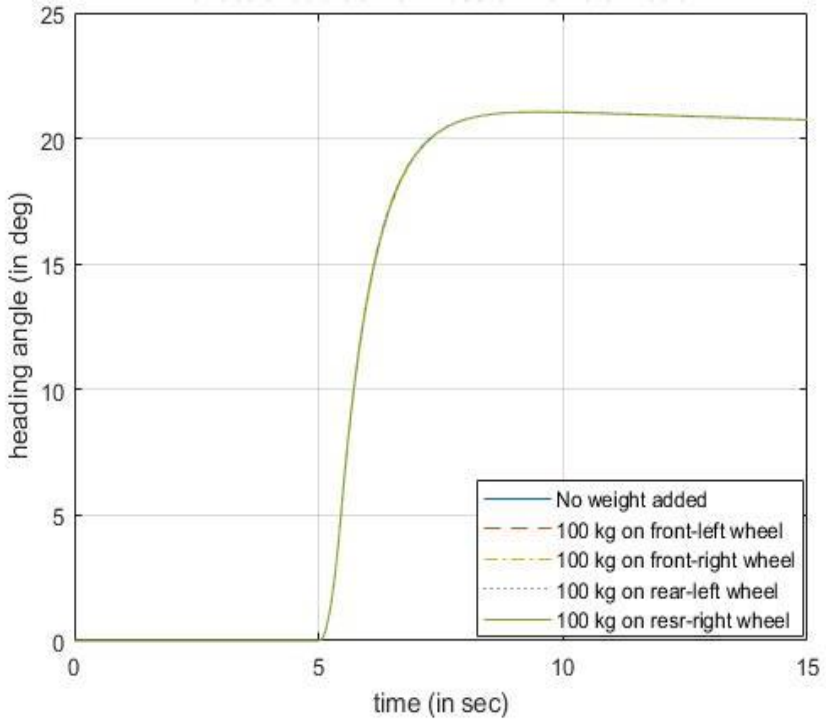


Figure 5.5 Effect of mass on vehicle performance

From the figure 5.5, I have simulated the vehicle model by adding 100kg on each wheel separately. It is being observed that there is no such effect on the vehicle even if we add more mass to it. The graph seems to follow the same as before.

5.1.3 Motion Controllers

I have implemented two motion control schemes on the robot and checked the performance and robustness of the vehicle. The simulations have been performed by taking various dynamic and kinematic vehicle parameters into account. The vehicle is moved with five different velocities so that we can design a controller which can track the vehicle path at every velocity. The controller should be such that it can handle the vehicle at different road and driving conditions. Meanwhile, we also have to take care that the tuning parameters should not be changed to make the vehicle performance better ,i.e., the vehicle should adapt to the condition it is facing.

First of all I have tuned the controllers to the get the best possible results. This has been done for both controllers for velocities 5, 10, 20, 30 and 40 km/hr. In addition we have also included the steering actuator in our basic model.

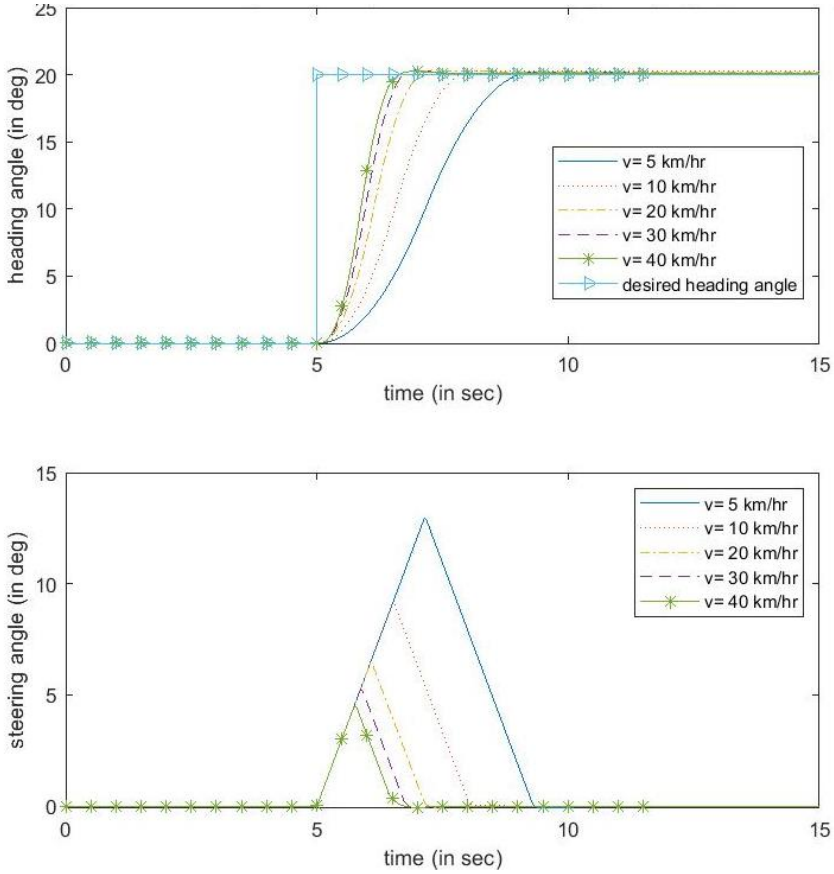


Figure 5.6 PID controller best tuned with varying gains and speeds

Figure 5.6 shows the vehicle heading and steering angle change when I am manually tuning the parameters of the PID controller. It is observed that as the velocities increase, settling time reduces and

the steady state remains within the desired limits. Also the gain values are recorded and tabulated for reference. The maximum settling time is 4.3 seconds for velocity = 5 km/hr while the minimum settling time is 1.8 seconds for velocity= 40 km/hr. The steering angle is also lower for higher velocities and higher for lower velocities. For v=5 km/hr, maximum steering angle is obtained as 13.47˚ and for v=40 km/hr maximum steering angle is just 4.88˚. The steady state error in case of PI controller is less than 2% for all the velocities.

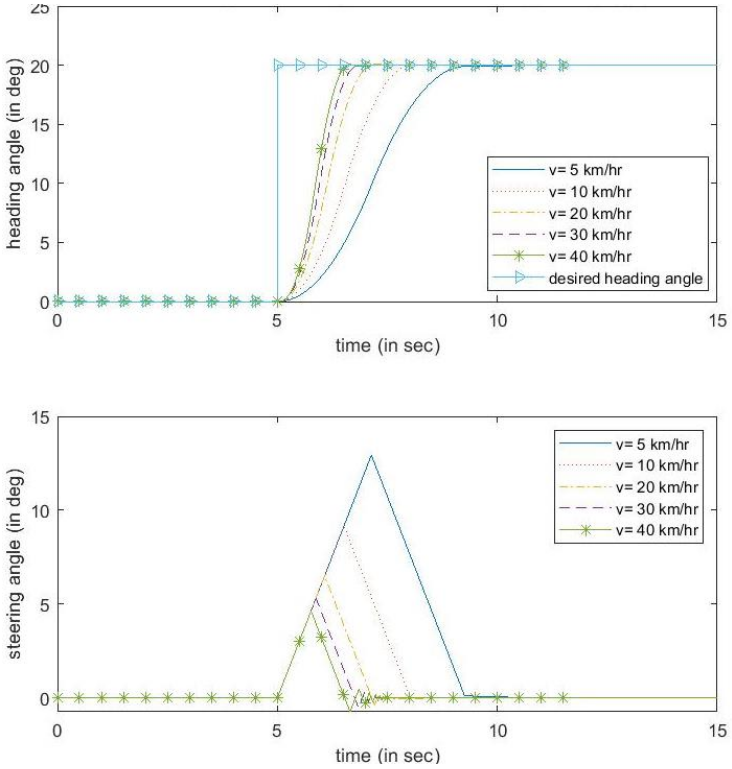


Figure 5.7 SMC controller best tuned with varying speeds and gains

Figure 5.7 shows the vehicle performance when SMC controller has been implemented. I have kept the same velocity range and the other vehicle parameters. The trend observed in settling time and steady state error is same of the PID but it shows a better performance in those respects. This can be clearly seen from the table 5.2.

Table 5.2 Comparison of best tuned SMC and PID controller

Speed (in km/hr)	**PI**		**SMC**		**PI gain values**		**SMC gain values**	
	settling time (in s)	steady state error (in %)	settling time (in s)	steady state error (in %)	kp	ki	ks	a
5	4.47	1.57	4.22	0.029	0.66	0.002	1.6	0.9
10	3.25	1.32	3	0.029	0.45	0.004	1.7	0.99
20	2.36	1.06	2.19	0.029	0.3	0.003	1.9	1.6
30	2.21	0.43	2.01	0	0.23	0.001	2.3	3.6
40	2.09	0.38	1.79	0	0.182	0.0009	2.9	9.24

Generally, it is observed in the vehicles that the gain values can't be changed as the vehicle changes its speed. The change is so instantaneous that such a controller should be chosen so that could provide a very slight deviation from the previously well-tuned gain values.

To incorporate this study, the simulation has been performed by keeping the gain values constant and varying the velocities. First of all, both controllers were best tuned for velocity 5 km/hr and then the simulation has been performed keeping the gain values constant.

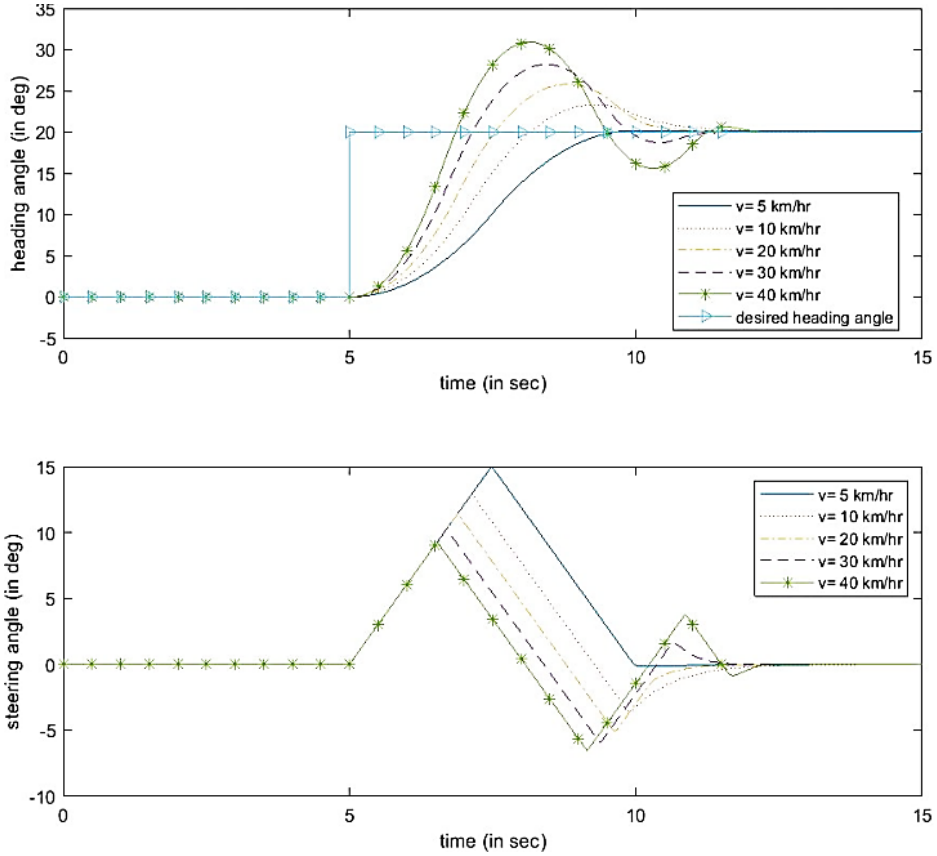


Figure 5.8 PI controller behaviour keeping gain constant with best tuned at 5 km/hr

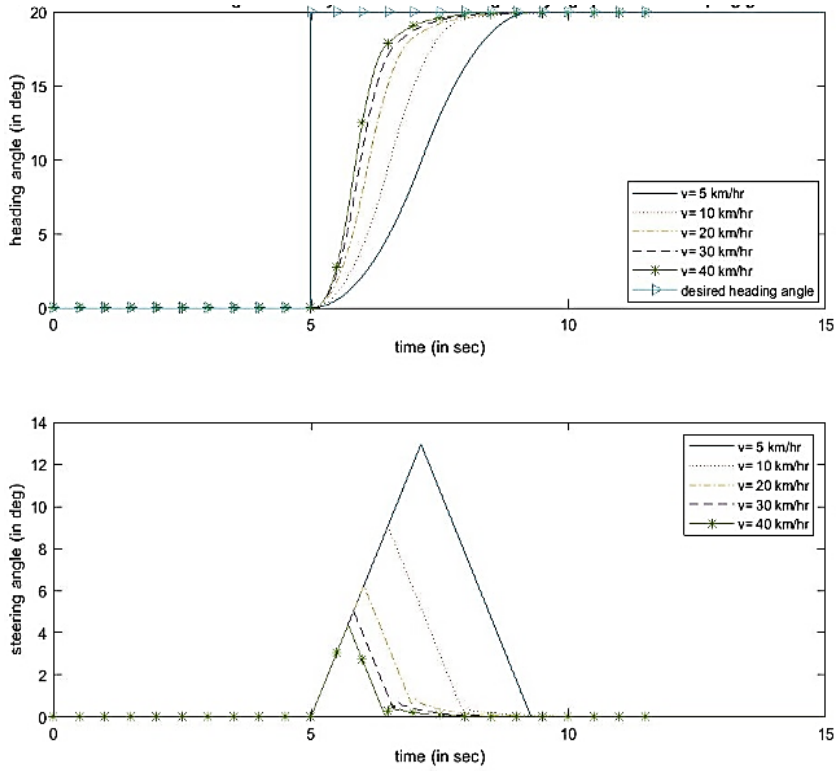


Figure 5.9 SMC controller behaviour keeping gain constant with best tuned at 5 km/hr

From the figure 5.8 we observe that as the best tuned PI controller for 5 km/hr velocity is applied to higher velocities there is an increase in overshoot as the velocity increases. Also, the rise time decreases with increase in velocity. The total time to achieve steady state significantly increase from 5 km/hr to 40 km/hr. This is because as the velocity increases, the overshoot increases significantly and thus the vehicle keeps on trying to follow the path but is unable to reach the desired trajectory quickly and oscillates around it.

This suggests that controller parameters tuned at 5 km/hr cannot be reliably used for controller at 40 km/hr. But controller tuned at 5 km/hr can be used for 10 km/hr. Therefore, the best tuned controller at a particular velocity can be practically used in a small range of velocities as there is lesser deviation from the performance of the best tuned one. This can be considered a disadvantage of PID controller as there are a lot of undesirable situations on road like bumps or some sudden jerk which increases the vehicle velocity significantly. In those cases the controller fails to give the desired control on the vehicle.

From the figure 5.9 we observe that as the best tuned SMC at 5 km/hr is applied for higher velocity the settling time almost remains same. There was no sign of overshoot as the velocity increases.

SMC performs significantly better for higher velocities once tuned for lower velocity than the similar tuned PI controller. This can be clearly observed from the table 5.3. In case of SMC I don't get any overshoot and even the settling time reduces with increasing speeds. So it can be said that this controller doesn't need to be tuned for a certain range of velocities and thus is able to handle uncertainties better than the previous discussed controller PID.

Table 5.3 Comparison of PID and SMC when best tuned at 5km/hr with increasing velocity and constant gain

speed (in km/hr)	**PID**		**SMC**	
	settling time (in s)	Overshoot (in %)	settling time (in s)	Overshoot (in %)
5	4.47	0	4.22	0
10	5.89	11.3	4.03	0
20	6.07	21.06	3.82	0
30	6.78	35.6	3.67	0
40	7.41	48.7	3.33	0

5.1.4 Linear and nonlinear model

As discussed in chapter 2, I have formulated two mathematical models for the vehicle. One includes the nonlinearities found in the model while the other one is linearized model. Here, I am going to compare both the models by implementing SMC controller and steering actuator dynamics whose results have been discussed above. Also these simulation results are compared with CarSim results for validation. Finally a double change manoeuvre has been implemented in the simulation to get idea of the real path tracking error.

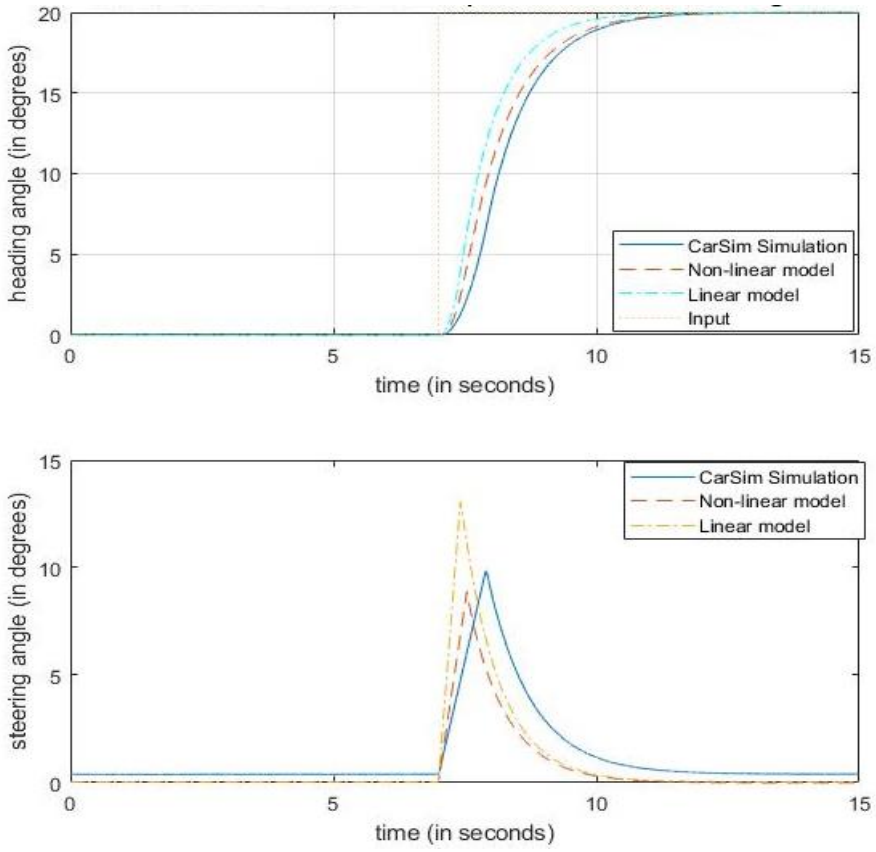


Figure 5.10 Linear and Nonlinear model comparison at 3.2 m/s for 20˚ turn

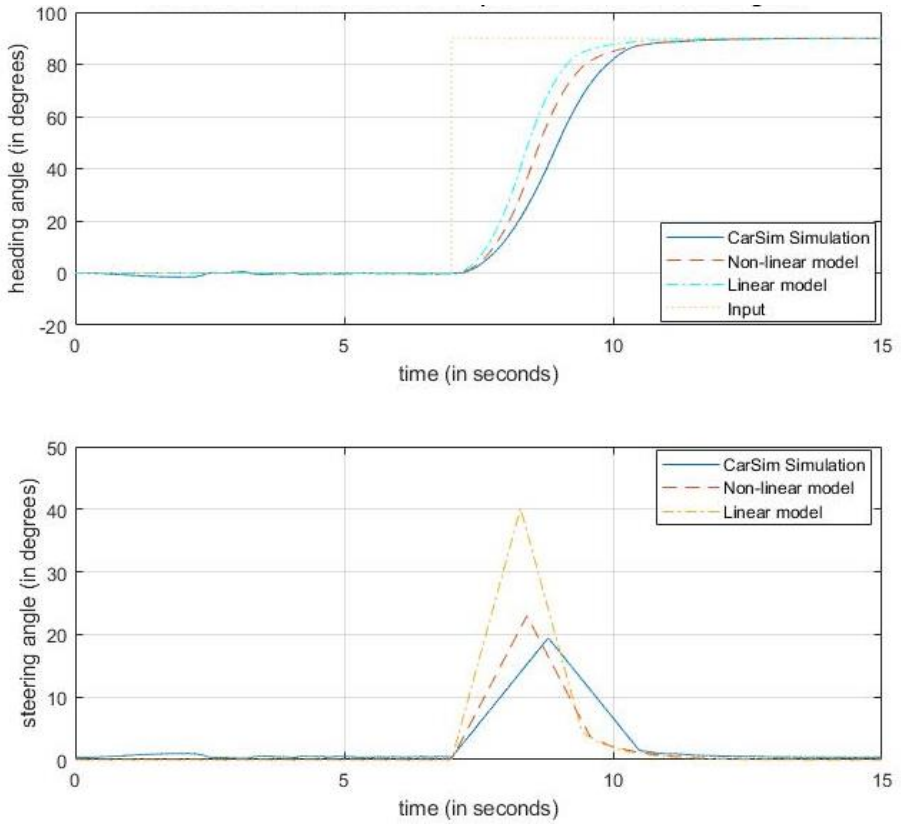


Figure 5.11 Linear and Nonlinear model comparison at 3.2 m/s for 90˚ turn

The comparisons have been made for two angles turn, 20˚ (lower angle) and 90˚ (higher angle). Generally the vehicle is run up to 5 km/hr. That's why the simulation is carried at a test speed of 3.2 m/s. From the figures 5.10 and *5.11, it can be seen that the vehicle can be better tracked by the non-linear model. There is a significant deviation the linearized mathematical model from the CarSim results. This needs to be avoided so that the trajectory tracking becomes consistent when the vehicle is moves on a real road. In case of lower angle turn, the error in trajectory tracking by linearized model may be considered within limits but for the right angle turn, the error is significantly large and can't be ignored.

In the table 5.4 below, the results have been tabulated.

Table 5.4 Nonlinear and Linear model comparison with SMC at 3.2 m/s

Simulated Model	**Heading Angle RMS error (in deg)**		**Steering Angle RMS error (in deg)**	
	20˚	90˚	20˚	90˚
Linear	1.3485	7.523	1.434	5.6318
Non-linear	0.661	1.7653	1.03	1.2897

After performing all the above simulation, we finally implemented the desirable controller and best predicted model in the vehicle simulation and a double lane change (DLC) manoeuvre has been implemented to find out the path tracking error.

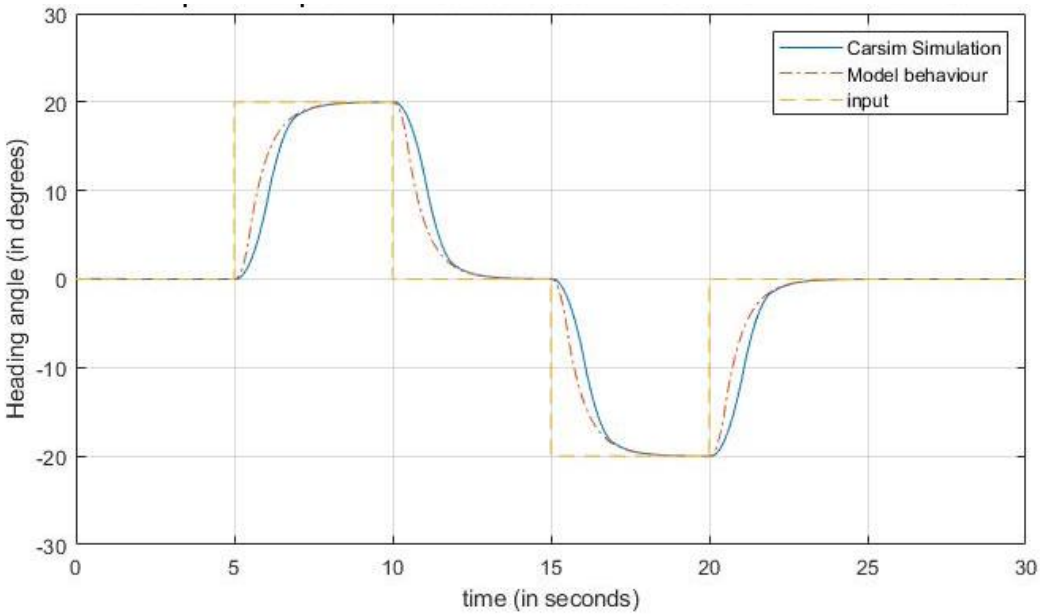


Figure 5.12 Comparison of predicted model with CarSim simulation at 3.2 m/s for DLC manoeuvre

In figure 5.12, I observe that matlab simulation results are quite close to desired heading input. It takes almost 3.2 seconds to reach a constant input of 20˚. Meanwhile in CarSim result, we observe that it takes almost the same time to reach the desired input. The results seem to follow the model behaviour quite closely. I have also performed the simulation for two more speeds 2 m/s and 5.55 m/s. The trend comes out to be the same for those two as well. In those cases, I have kept the SMC gain values same as before so as to take the real world situation into account. Finally the results have been tabulated in table ***.

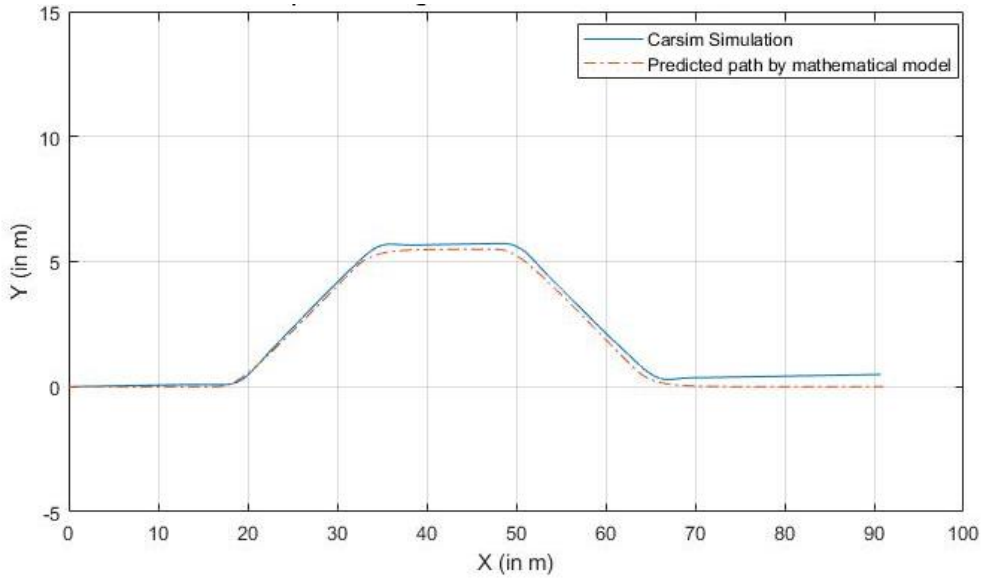


Figure 5.13 Path tracking for DLC manoeuvre at 3.2 m/s

Figure 5.13 shows the path tracking comparison between the formulated model and the CarSim results.it is observed that the actual car (CarSim model) seems to be a little deviated from the desired path. This deviation has been quantitatively found out for all the three speeds. These errors increase as we increase the speed. This can be actually expected on a real vehicle since there is always some drift error when a vehicle is moved straight. These drift errors can be attributed to the tire slip caused due to non-linearities in tire models. The below table 5.5 shows the path tracking errors for different speeds.

Table 5.5 Predicted model errors for different speeds in DLC manoeuvre

Vehicle velocity (in m/s)	**Heading Angle comparison**			**Path tracking RMS error (in m)**
	RMS error (in deg)	Steady state error (in deg)	Settling time (in sec)	
2	1.073	0.02	4.32	0.4512
3.2	0.687	2.56*e^-3	3.83	0.5123
5.55	3.657	1.09*e^-3	2.18	0.542

5.2 To find and reduce odometry error on the experimental vehicle

The experiments have been performed on *0xDelta* series robot, a four wheeled differential-drive robot. The controller codes have been implemented using ROS platform. The basic objective of this part is to find the real problems when we move a robot in real world. As we know there are certain limitations for every sensor so sensor fusion techniques using kalman filter has also been applied.

There are four sensors which have been put into use: IMU, wheel encoder and integrated GPS-INS. Integrated GPS-INS provides a reference for the actual path and orientation taken by the robot. My objective is to use wheel encoder and IMU data to get the corrected pose estimates of the robot in outdoor. But the robot is also designed to perform indoor navigation where we can't rely on GPS. So my basic idea is to fuse odometry and IMU data to get the correct pose estimation in indoor environment. Even the sensors publish data at different rates. So to incorporate those anomalies sensor fusion becomes quite necessary.

5.2.1 Odometry drift calculation

Firstly the odometry drift has been quantitatively studied by moving the robot in a straight path with two speeds. It is observed that as the robot moves forward, the drift gets accumulated. This should be avoided as the odometry is publishing data in the robot as if it is moving in a straight path but in reality the robot is having some lateral drift. This gives the wrong pose estimated to the controller and there is likely for

the robot to hit the walls or some obstacles. So, this makes it clear that the odometry data is not sufficient to give the corrected pose estimate of the robot.

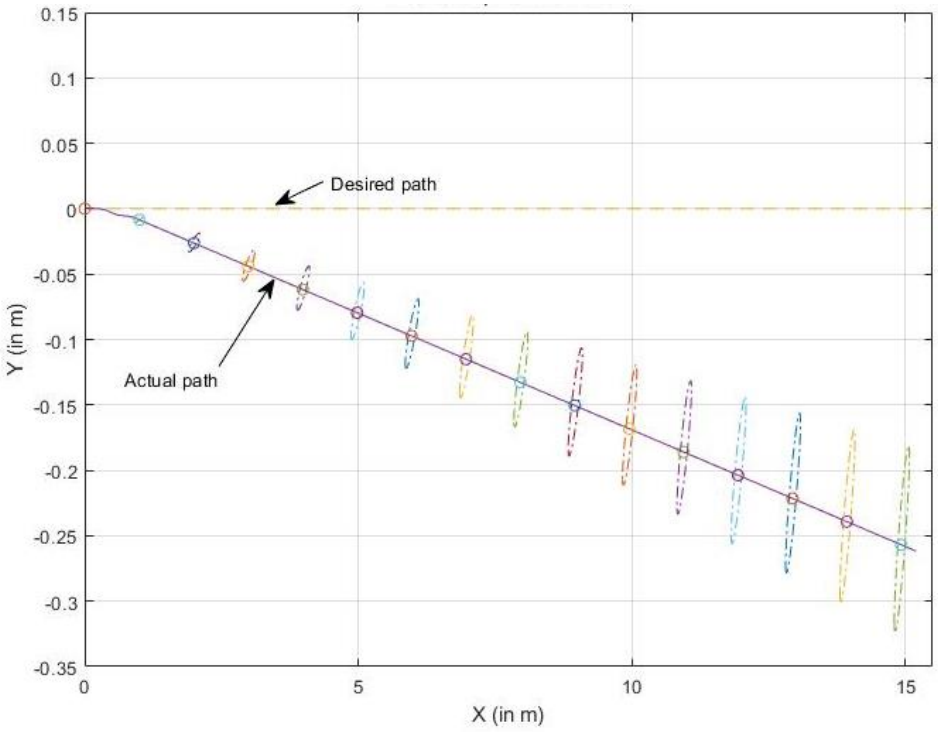


Figure 5.14 Robot running at 0.4 m/s for 15 m

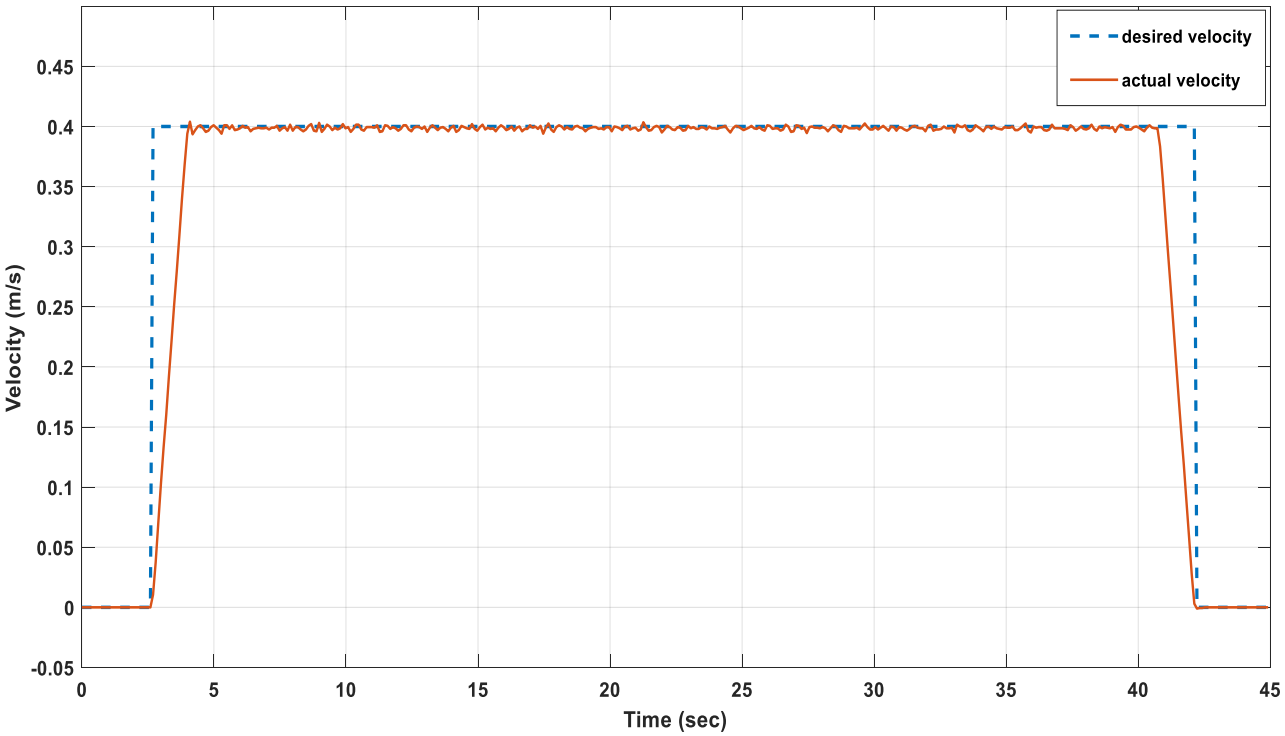


Figure 5.15 Velocity analysis at 0.4 m/s for 15 m

Table 5.6 Path tracking errors when the robot is moved straight

Case	Desired linear velocity (m/s)	Desired distance (m)	Actual distance covered (m)	RMS Error in distance (m)	Linear velocity RMS error (m/s)	Lateral Drift (m)
1	0.2	10	9.89	0.1380	0.0174	0.15
2	0.4	10	9.81	0.2684	0.0622	0.18
3	0.2	15	14.86	0.1819	0.0156	0.253
4	0.4	15	14.78	0.3260	0.0584	0.287

From the above experimental results, it can be observed that the lateral drift increases as the robot covers more distance. This is because the error in odometry keeps on accumulating over the time and gets added each time. This deviation is expected at every kind of surface. In this experiment the robot is moved on tiles which are very smooth as compared to roads. As the surface becomes different, it is highly likely that the robot will not show the same results as before. Also with increasing velocity, the lateral drift increases due to increase in lateral tire forces. These tire forces are solely responsible for lateral drifts. Therefore increasing the speed of the vehicle causes more deviation from the actual path.

5.2.2 Running robot in outdoor environment and Kalman filter

In the next set of experiments the robot is allowed to run in the outdoor environment where the data is collected using the sensors mounted on the robot. Firstly the sensors have been transformed to the robot base coordinates. So the pose estimation has been done in the robot frame. In this case the basic idea is to know about the trajectory followed by the robot when it is given some velocity commands.

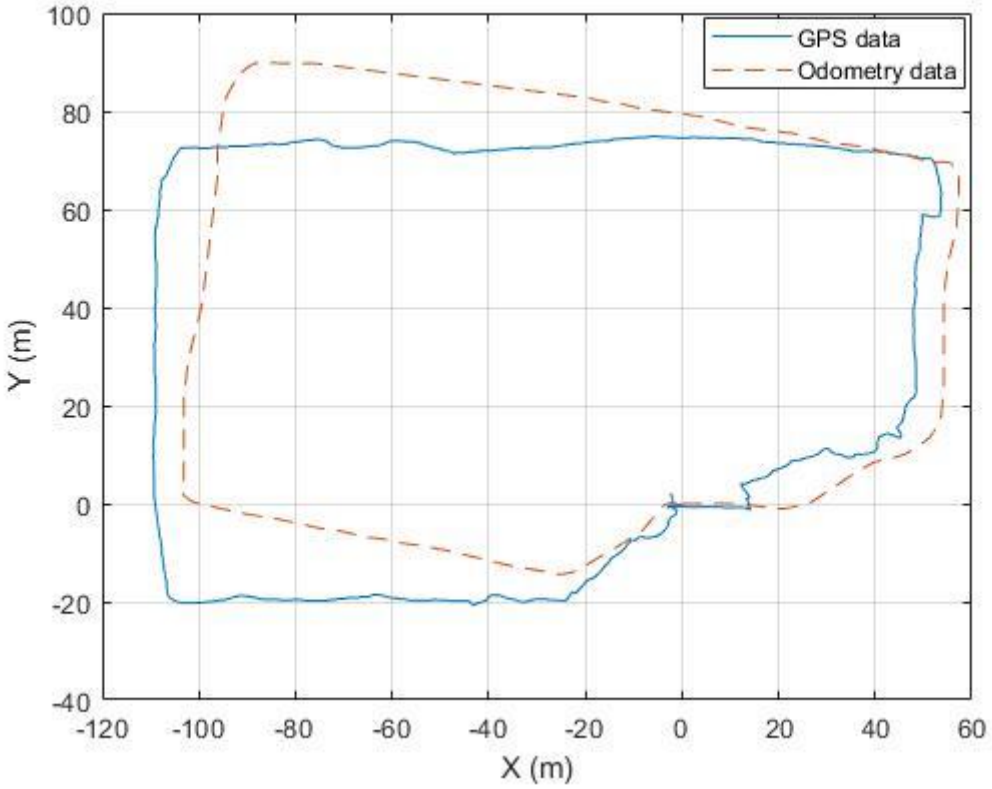


Figure 5.15 Representing GPS and raw wheel odometry data when the robot moved in a closed path

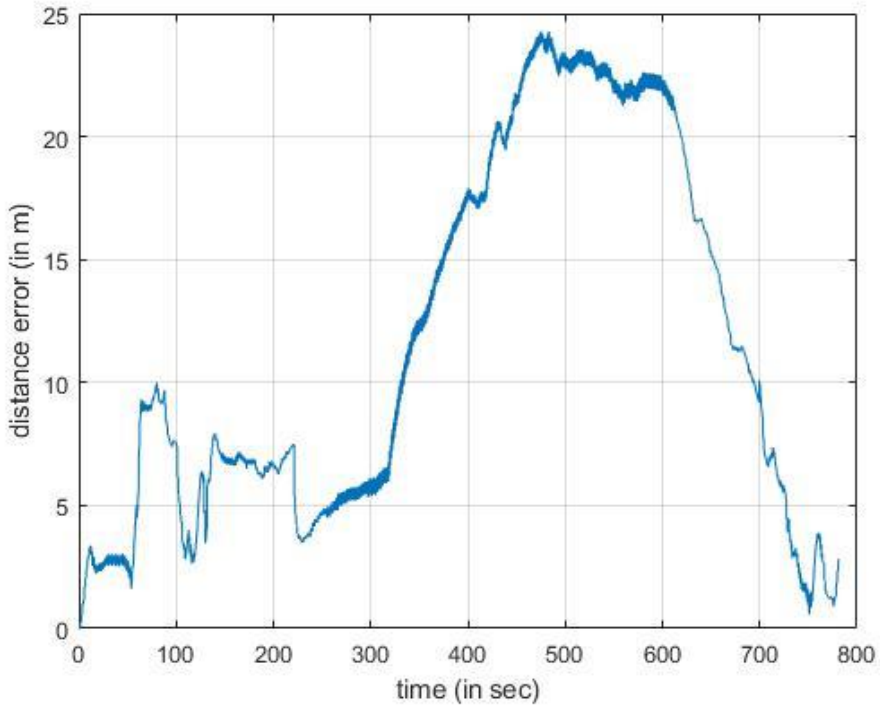


Figure 5.16 Path tracking errors

In the figure 5.15 the robot is moved to form a closed loop. It can be easily observed that the odometry drift got accumulated shows very different results from the actual GPS data received. This should be definitely avoided otherwise accidents are likely to occur due to wrong pose estimated by wheel odometry. The distance RMS error calculated from these data has been found out to be around 11.50

metres. This error is quite large and is clearly not acceptable. In figure 5.16, the maximum error is found out to be 24.56 m.

Data publishing rates of different sensors:

- GPS (Garmin):1.02 Hz
- IMU (Xsens): 97.8 Hz
- Wheel Odometry: 10 Hz

The kalman filter used in our case use a filter time parameter which publishes data with a frequency of 20 Hz. The odometry data is fused with IMU data to get the best possible position estimates.

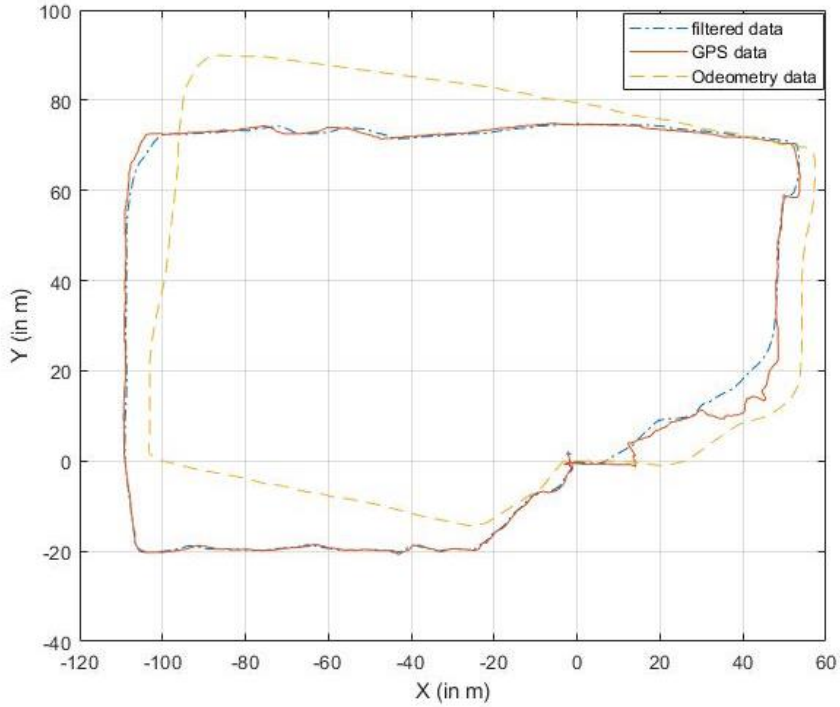


Figure 5.17 comparison GPS data and Kalman filter data obtained from the fusion of odometry and IMU

From the figure 5.17, we can clearly see that the filtered data follows the GPS data quite accurately. The RMS error is found out to be less than a metre (0.89m) and the maximum error is 2.45m. These errors are within the acceptable range. Thus we can see that there is no need to rely only on GPS data rather we can implement sensor fusion techniques which can predict a close behaviour. The kalman filter uses weight parameters for different sensors. These parameters have been tuned to get the best possible result. If we include GPS data into this sensor fusion the results are further better. We know that each sensor has certain limitations: IMU gives just the orientation and acceleration data, Odometry gives drift errors and GPS works well in outdoor but in certain places we can't have a good GPS (for example, tunnels). So it is good to use all the three data to get the best possible position estimates of the robot. This is a lot needed when the localisation part is done on the robot. For performing autonomous navigation it is important that

the robot should know its current position to the best accuracy because if wrong pose estimates will lead to incorrect map building and the navigation might fail.

5.3 Autonomous Navigation in Indoor Environment

Finally after setting up the robot and calculating all sort of errors I implemented the navigation algorithm for obstacle detection and obstacle avoidance. This is done in two phases: first phase deals with the localisation part, i.e., map-building using SLAM and second phase deals with implementation of path planning algorithms based on the map created and the laser scan received from the LiDAR.

5.3.1 Localisation

If one could attach an accurate GPS (global positioning system) sensor to a mobile robot, much of the localization problem would be obviated. The existing GPS network provides accuracy to within several meters, which is unacceptable for localizing mobile robots. Furthermore, GPS technologies cannot function indoors or in obstructed areas and are thus limited in their workspace.

That's why a map based approach is considered the best for indoor navigation of the mobile robots. It includes both localization and cognition modules. In map-based navigation, the robot explicitly attempts to localize by collecting sensor data, then updating some belief about its position with respect to a map of the environment.

The robot constructs a two-dimensional geometric representation of its environment using the laser scanner. It utilizes a combination of this geometric data and odometry information supplied through the wheel encoders to determine its current location. It generates a point cloud where each point corresponds to a location where it believes it could be based on available data. As the robot moves, it rules out possible locations and the number of points in the cloud decreases. In this way, its number of belief states rapidly converges to its true location. Thus, the robot achieves localization through probabilistic inference. The below figure 5.18 depicts the map building in a closed room.

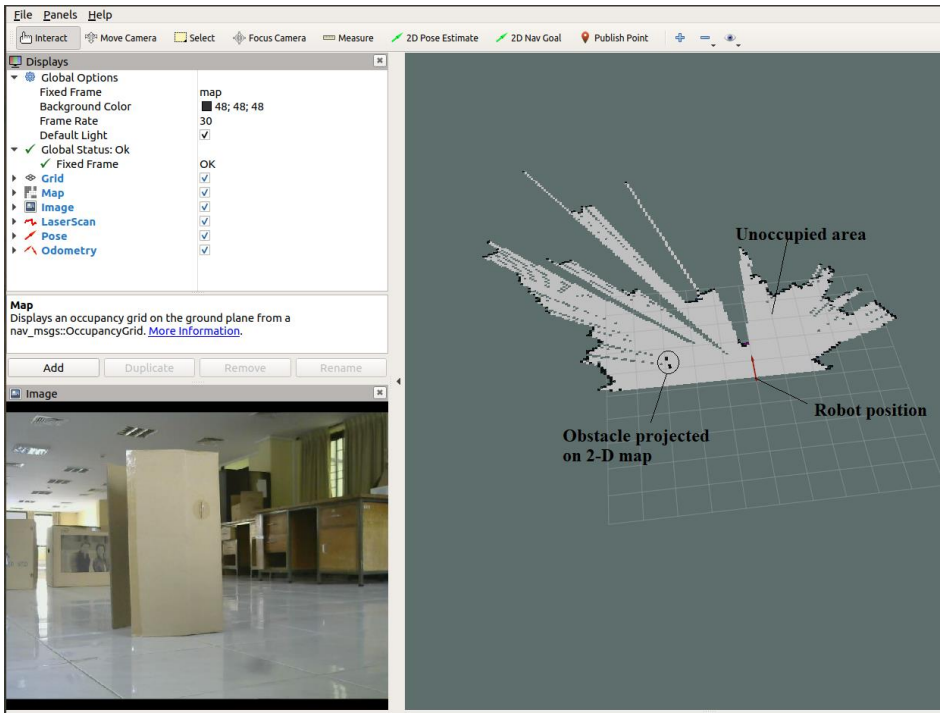


Figure 5.18 Map building environment in ROS

Here the robot is tele-operated to move all around the room and scan all kind of obstacle. The sensor used to scan the environment in the above case is a 2-D LiDAR. This creates the map in a 2-D plane by dividing them into grids called occupancy grids. On the basis of presence of any obstacle these occupancy grids are assigned cost values to them. As discussed previously, there three markings to these grids: Occupied, free and unknown. In the above figure, the dark black lines are the obstacles and the whitish grey is free/unoccupied area and the rest is unknown space. After moving the robot in the entire room, the final map created is shown in figure 5.19.

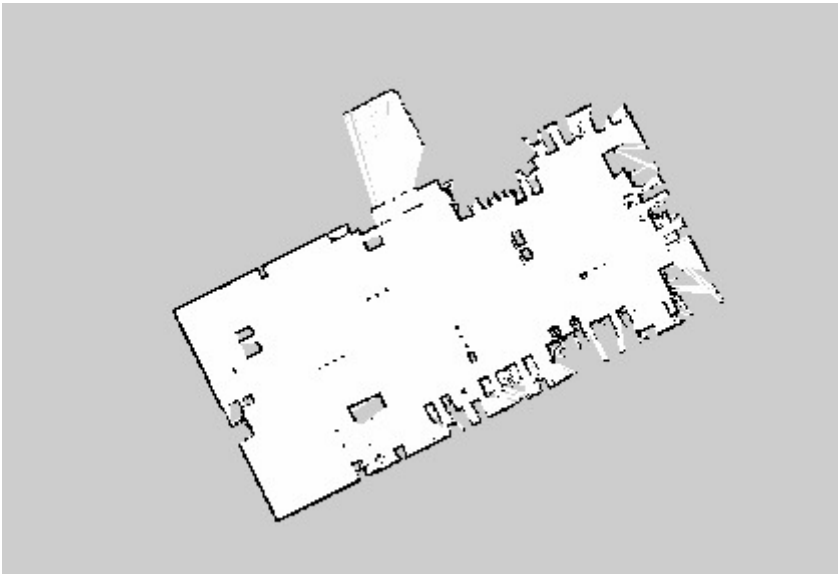


Figure 5.19 Complete map of the room

The robot must not only create a map but it must do so while moving and localizing to explore the environment. This is often called the simultaneous localization and mapping (SLAM) problem. To get the position estimates at every moment the robot is moving, adaptive monte carlo localization (AMCL) is used. This particle filter based on past coordinates and velocity commands predicts the current position of the robot in the form of particles. This also uses the laser scan and based on the landmarks observed predict the current position of the robot.

5.3.2 Path Planning Algorithm Results

Once there is map in place the mobile robot tries to navigate through the map area keeping map as a reference. While navigating the robot tries to follow global plan to plot a path to those desired coordinates. There are various global planners for navigating a robot across a map area to name a few: Dijkstra's algorithm, A* algorithm. Once a global plan has been generated, the local planner translates this path into velocity commands for the robot's motors. It does this by creating a value function around the robot, sampling and simulating trajectories within this space, scoring each simulated trajectory based on its expected outcome, sending the highest-scoring trajectory as a velocity command to the robot, and repeating until the goal has been reached avoiding all the obstacle in the way.

Global path planning

In the experiments I have used Djisktra algorithm as the global planner and two local planners: Timed Elastic Band and Dynamic Window Approach. Each has their own problems and their own advantage. . Once the robot has localized successfully, it can be supplied with destination coordinates and uses a global planner to plot a path to those coordinates. The global planner uses the static map created using the LiDAR sensor and plans the most optimal path. The local planner takes care of the reactive obstacle avoidance problem. If any sudden obstacle is detected by the laser scanner, it overwrites the global path and plans a new path. It also checks that if there is any path possible or not. In case no valid path is found, it sends no velocity commands to the vehicle and displays the message that no valid paths could be found.

Effect of footprint model

Footprint is the contour of the mobile base. In ROS, it is represented by a two dimensional array of the form [[x0, y0], [x1, y1], [x2, y2], ...], no need to repeat the first coordinate. This footprint will be used to compute the radius of inscribed circle and circumscribed circle, which are used to inflate obstacles in a way that fits this robot. Usually for safety, we want to have the footprint to be slightly larger than the robot's real contour.

To determine the footprint of a robot, the most straightforward way is to refer to the drawings of your robot. Then we determine the centre of the mass of the robot by considering that the mass is uniformly

distributed. We consider this as our centre of the robot which becomes our *base_link* frame point. The vertices of the robot are determined with respect to the centre of the robot.

Generally the robots are treated as circular objects. For a robot with circular footprint, path planning is done by considering the robot as point robot and the obstacles are inflated by the robot's radius. At the same time it can be done other way round as well. But all the robots are not circular, which has to be taken into account. In our case the robot has a rectangular shape with a semi-circular top. When operating in cluttered spaces it therefore becomes important to evaluate the footprint of these robots against a cost map. This evaluation is one of the major computational burdens in planning for robots whose footprints can't be assumed to be circular.

The footprint models used in the experimentation are:

- *"point"*: This footprint is considered useful when the robot has a circular shape. By considering this footprint model, we observe that the robot takes path quite close to the obstacle. There are certain regions which are thought to be bit crowded but the robot still planned those paths and ultimately has to change its course during reaching its goal. The main advantage with the point footprint model is the computation time. It shows really less computation power and time required as compared to other footprint model. Thus decision making becomes really fast.
- *"circular"*: The circular type parameter represents the robot as a circular with the perimeter circumscribing the entire robot. It is necessary to consider the centre of robot in this case as the radius is decided on this basis. This footprint model provides an advantage by removing the need for inflation radius. Meanwhile, it takes out the risk for the robot's orientation in planning the path. I observed that the velocities taken during this footprint model was not aggressive as compared to the point footprint model.
- *"line"*: This type of footprint comes handy when we are working with a robot which has length greater along one direction than the other. This is particularly the case in two-wheeled differential robot. This is the case when we considered when the inflation radius was taking care of the obstacle avoidance. But if we reduced the inflation reduce to provide low cost paths, we found that the robot was hitting the obstacle (boxes) while achieving its goal. This situation could be only avoided when the inflation radius also takes the robot dimension into account. This increases risk for high computation power.
- *"two_circles"*: This type of footprint increases the computation complexity since the obstacle distance has to be computed from both circles. Though this works well considering the safety of the vehicle. The object avoidance is good since it is not hitting any obstacle coming in its way.
- "*polygon"*: This footprint model can closely determine the robot size. However the problem can come in case of robot's orientation. There is always a chance that in case of on-spot rotation in an

obstacle surrounded area, it might not perform well. There have been occasions that it hit into something while performing even the recovery behaviour. To avoid this, a proper value of inflation radius has to be provided. With proper inflation, the robot behaviour has been more efficient in the context that it could plan the shortest path. Though at the same time it takes the highest compute.

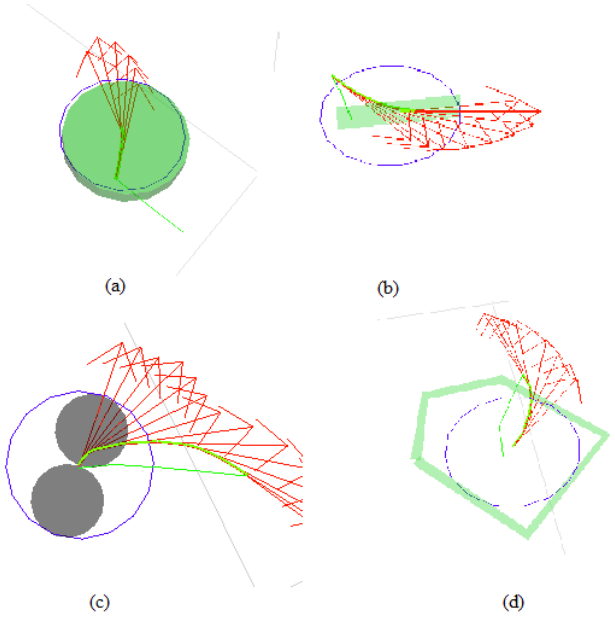


Figure 5.20 Footprint models [(a) circular (b) line (c) two_circles (d) polygon]

Timed Elastic Band (TEB)

Based on the velocity and acceleration limits of the vehicle, the security distance of the obstacles and the geometric, and kinematic and dynamic constraints of the vehicle, it generates velocity command for the vehicle. This planner generates a quite complex trajectory for reaching the goal.

In the experiment a goal position has been given to the vehicle in *rviz* window. Based on the map provided to the robot and the laser scan received it generates a velocity command to build a global path and local path on which the vehicle is supposed to move. The landmarks scanned by laser gives the robot an idea about the current position of the robot. Figure 5.21 depicts the path followed by the robot to achieve the goal. The RMS error in the trajectory tracking is found out to be 0.535 m.

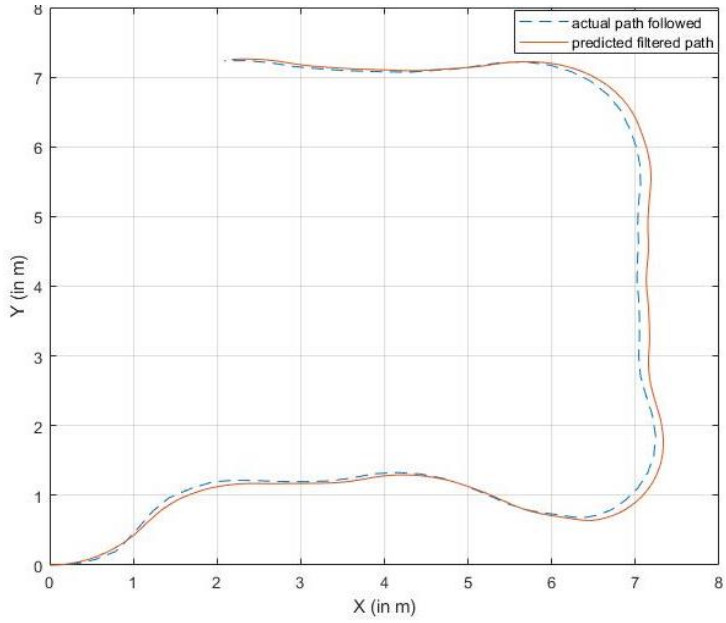


Figure 5.21 TEB local planner path behaviour

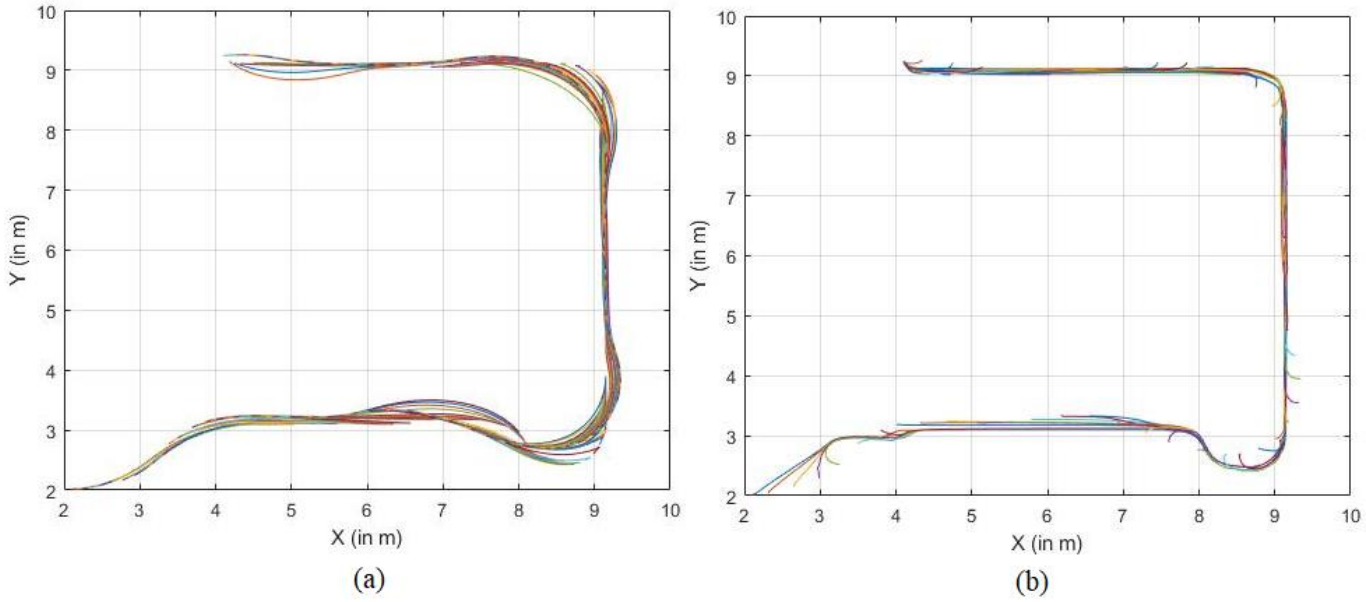


Figure 5.22 Global path (b) and Local path (a) followed by the vehicle using TEB

Figure 5.22 (a) depicts the local path followed by the TEB local planner to complete the goal given. Based on the laser scans received it avoids the obstacle and plans an optimal path around the obstacle. Figure 5.22 (b) depicts the global path planned by the robot considering the map data which has been fed to the robot. If there is no obstacle present or inflated cost around the object, the robot is supposed to follow just the global path. It is seen that the global path takes a lot of time in computing the path and publishes the data at a lesser frequency than the local planner. It is important as the global path should take care of the best optimal path based on map while the local planner is concerned for the instantaneous obstacle coming in front of the robot.

Errors involved in TEB local planner: As some obstacle comes in between the goal and robot position it takes a very complex trajectory to avoid it. This reduces the robustness of the vehicle and due to such trajectory generation, shortest path is not taken by the robot. There are a lot of clear paths which are avoided by the robot. This can be seen in the below figure 5.24.

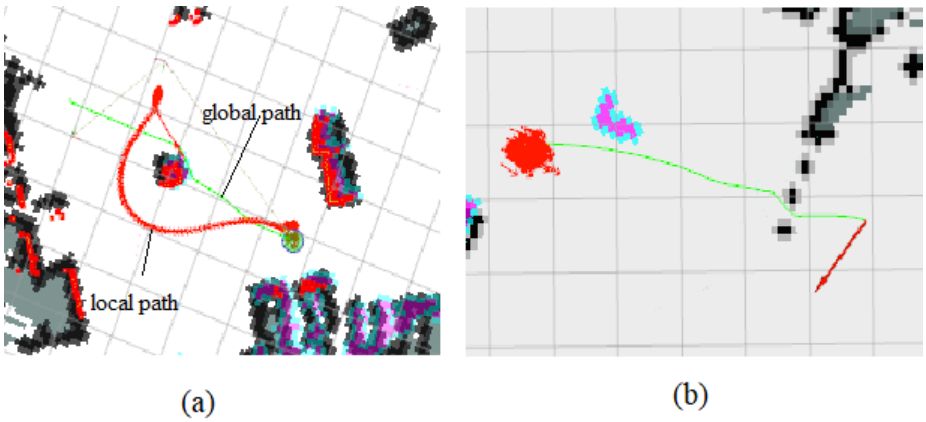


Figure 5.23 TEB local planner errors

In the figure 5.23 (a), the trajectory generation is very complicated while in (b) the trajectory generated passes between the obstacles which is incorrect to proceed as the robot is likely to hit the obstacle.

Apart from that due to generation of backward velocities, it also hits the obstacle if kept at its rear side where laser scan is not available. This can create an issue for the vehicle and passenger safety.

Dynamic Window Approach (DWA)

Based on the velocity search space and the cost values provided to the algorithm, this planner generates the velocity commands. The main advantage of this planner over TEB is that it doesn't generate any backward velocities and thus it improves the obstacle avoidance issues caused due to TEB.

Various situations have been tested using this planner. It is observed that the robot moves very close to the obstacle. In that case footprint padding needs to be increased. Besides the cost weight parameters need to be tuned for getting the best performance of the planner. Generally *path_distance_bias* has to be considered as this parameter decides if the robot's local planner will overwrite the path planned by the global planner or not. *goal_diatance bias* parameter has to be increased in order to make the vehicle reach to the correct goal position.

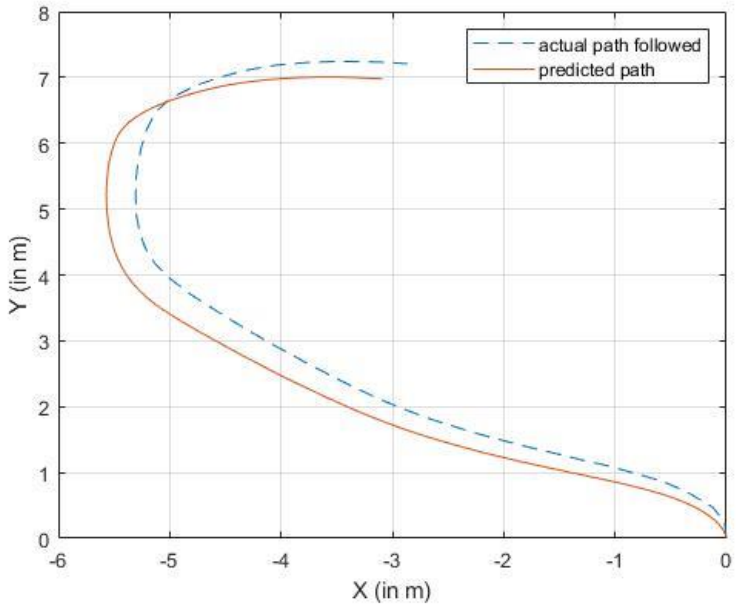


Figure 5.24 DWA local planner path behaviour

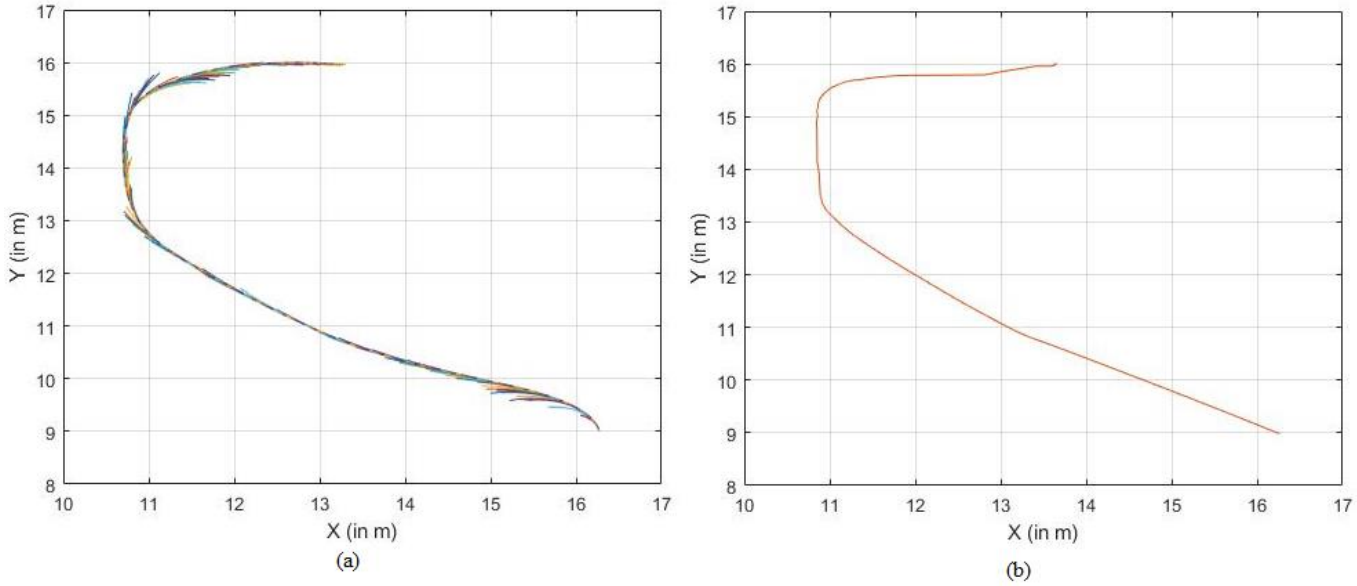


Figure 5.25 Local path (a) and Global path (b) followed by vehicle using DWA

Figure 5.25 depicts the path tracking when DWA algorithm is implemented on the robot. It can be seen the path tracking is not efficient as compared to the TEB planner. The RMS error in path tracking has been found out to be 0.86m which is 53% higher than the previous implemented planner. To be consistent with the path tracking error, the vehicle dynamics need to be considered as in case of DWA there is on spot rotation of the vehicle which increases the chance of vehicle slip. This vehicle slip leads to more lateral drift which keeps on accumulating and thus the error gets added as the vehicle moves forward towards reaching the goal.

An advantage of this planner can be clearly seen in figure 5.26 where the local planner plans a very simple trajectory and reduces the computation power and time. Even there is just one set of global planner required considering there is no obstacle around and the local planner follows it perfectly.

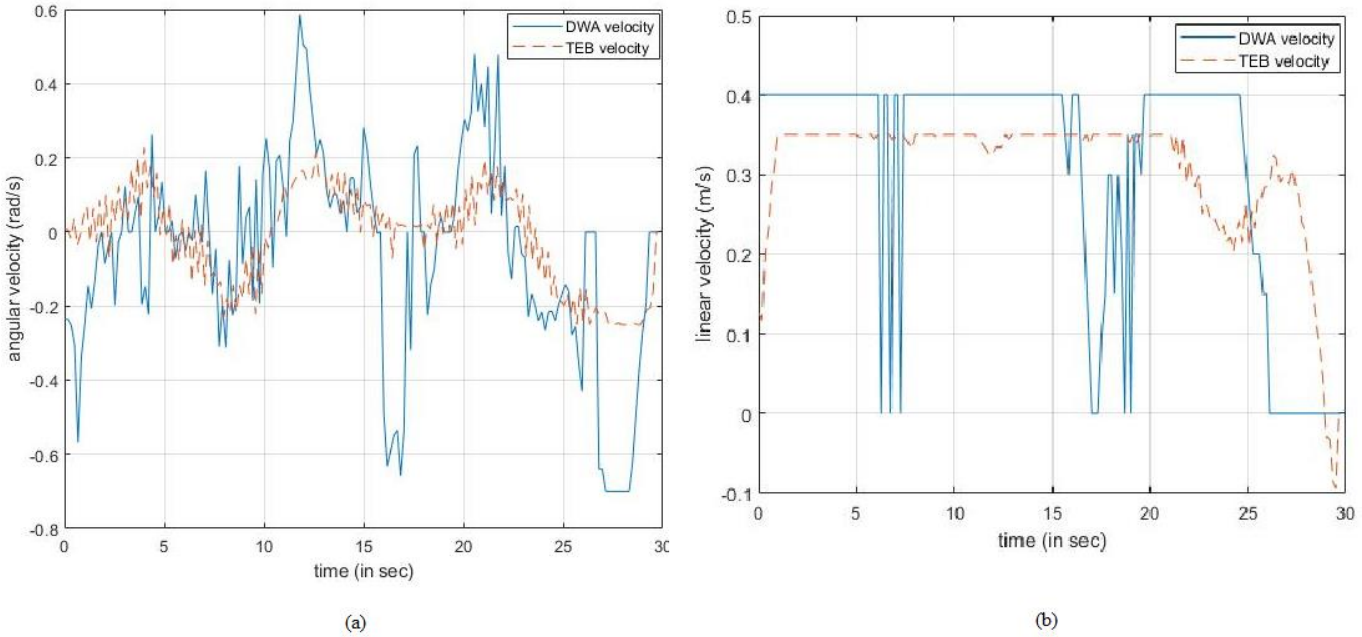


Figure 5.26 Velocity comparison of TEB and DWA local planner

Above figure 5.26 depicts the velocity comparison of both the local planner. Though the distribution might seem random but it is clear that the TEB planner follows more regressive approach. It can be seen in the linear velocity commands where a simple step velocity is given in DWA but it's not the case in the other planner.

Obstacle Avoidance

Using the above discussed algorithms, obstacle avoidance tests have been conducted on the robot. The robot is given commands to go from one goal position to another. Static as well as dynamic obstacles have been put in between the trajectory made by the robot. In all the cases, the robot seems to avoid these obstacles successfully and plans out a safe path. If there are no paths available, the robot doesn't make any path and no velocity commands are executed.

We can also determine how closely the robot can avoid the obstacle by changing either *footprint padding* or *inflation radius*. It is advisable to determine these two parameters carefully because if there is less critical distance then the robot can hot the obstacles while if it has large values then some of low cost paths are avoided which can reduce the efficiency of the vehicle.

Costmap Prohibition Layer

This layer has been plugged into the global costmap and marks the areas within the coordinates as prohibited areas. This works successfully in our platform. This layer comes into use when it is undesirable for the vehicle to go into some areas. For example, suppose in the room an area is occupied for a short period of time and the obstacle doesn't have height based dimension. In that case if the robot tries to take the path around those regions, it can become troublesome.

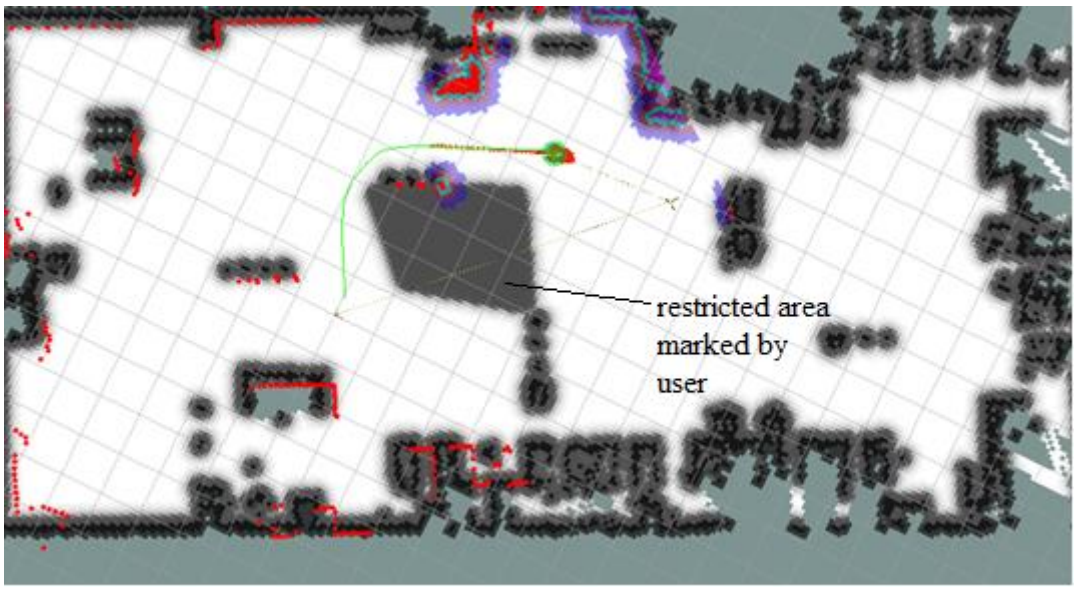


Figure 5.27 Costmap Prohibition layer

In the above figure 5.27, it can be clearly seen though the robot has a short path available to it, it's taking a longer the path as it has been marked as occupied by the prohibition layer. This comes quite handy when there is a temporary unavailability of certain routes in the map. I have also included inflation around the perimeter which has been used to mark the entire area as occupied.

Chapter 6

Conclusion

Vehicle modelling and autonomous navigation on a wheeled mobile robot has been completed in this project. Simulation and experiments have been performed to get the idea of vehicle behaviour under different condition. Moreover some of the challenges faced in implementing autonomous navigation in real world have been solved.

A mathematical model which closely predicts the behaviour of a real vehicle has been formulated and thus eliminating the need for actual experimentation on the vehicle. This helped to reduce the cost and efforts put for the real experimentation. A non-linear mathematical model with steering actuator dynamics has been tested and a sensitive analysis of various vehicle parameters affecting its performance has been done. It has been found out that a non-linear mathematical model gives better results in terms of settling time and steady state error than the linearized model. Especially, when the vehicle is taking large angle turn, for example a right angle turn, the linearized model can't handle the driving condition of the vehicle whereas the non-linear model shows behaviour quite close to a real car model.

Two motion controllers, PID and SMC have been implemented on the vehicle model and their results have been compared. It has been seen that the PID controller shows good results but when the vehicle velocity changes it is not able to track the vehicle path as desired by the user. Since in real world driving tuning parameters take time as well as high computation, SMC controller is preferred which can handle vehicle uncertainties in different road conditions. To avoid chattering phenomena an adaptive high order SMC with sigmoid control law has been implemented. It is found out that this controller handled the vehicle parameters efficiently than the PID. There has been no overshoot when the vehicle velocity is increased.

After the simulation results of mathematical model, experiments have been performed on a differential-drive robot. The mathematical model for the same has been done as well and then the lateral drift has been calculated by making the robot move in a closed path. The robot has been moved in outdoor environment and GPS/INS data is taken as a reference for the paths followed by it. The wheel odometry and IMU data is fused to get position estimates of the robot. This filtered data is compared with the reference data received by GPS/INS. It is found that the pose estimation done with implementation of kalman filter to fuse IMU and wheel odometry data follows the GPS data closely. So it is concluded that rather than using a high cost GPS/INS sensor for localization of robot, a cost-effective way by sensor fusion technique is possible. Apart from that GPS has a lot of limitations like it can't work in indoor environment or in road tunnels. So this fused data can be used in those areas to localize the robot.

The map building has been done using the *gmapping* technique which takes the laser scan reading and build the map. This map is divided into small parts called the occupancy grid. These occupancy grids are given the cost values on the basis of free or occupied area. An inflation layer is also added around the obstacle so that the vehicle should not follow a path very close to the obstacle.

Two local planners have been tested on the robotic platform: TEB and DWA.TEB is very good for obstacle avoidance but it has limitation because of the complex trajectory generation while DWA performs the obstacle avoidance correctly and also it gives simple velocity commands to the controller. In a crowded area, TEB behaviour is not acceptable due to generation of negative velocities because it can hit the obstacle at the rear side of the vehicle (if there is no sensor mounted on the rear side). DWA does an on spot rotation to visualize if the path is clear or not. After that it plans out the trajectory to be taken to achieve its goal. A new layer called *costmap prohibition layer* has also been tested on the vehicle. In this layer, a region has been marked as an occupied area. This is done because there are situations when we don't want the vehicle to plan its path around certain region even though that may be the shortest path. After implementing this layer the robot is found to be taking the longer path as expected.

Future Scope

There are still a lot of challenges that need to be addressed in autonomous navigation of mobile robots. The further work is towards building a 3-D occupancy grid map so that the vehicle can sense the environment in all the directions. This is important when the robot is moving on a road and a bump or a ditch comes in its way, it should sense the environment depth and avoid such obstacles. Apart from that many non-linear factors like aerodynamic drag, dynamic road friction has been avoided in the mathematical modelling of the vehicle. This can be included to get a complete picture of a real vehicle performance. Roll over dynamics is also to be included to have a full 3-D stable model of the vehicle.

References

[1] Keshav Bimbraw, A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology, DOI: 10.5220/0005540501910198 In Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO-2015), pages 191-198 ISBN: 978-989-758-122-9.

[2] Juan Rosenzweig ,Michael Bartl, A Review and Analysis of Literature on Autonomous Driving E Journal: Making-of Innovation,THE MAKING-OF INNOVATION, E-JOURNAL OCTOBER 2015

[3] Mayur Rukhadbhai Hadiya, A Review paper on Development of Autonomous Vehicle,International Research Journal of Engineering and Technology (IRJET), Volume: 06 Issue: 01 | Jan 2019

[4] Zhiqiang Li, Lu Xiong, Dequan Zeng, Peizhi Zhang, Zhiqiang Fu, Jie Yao and Yi Zhou, Predictable Trajectory Planner in Time-domain and Hierarchical Motion Controller for Intelligent Vehicles in Structured Road, 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France. June 9-12, 2019.

[5] M.Bergerman, Omead Amidi, J.R.Miller, N.Vallidis, T.Dudek. "Cascaded Position and Heading Control of a Robotic Helicopter." IEEE/RSJ International Conference on Intelligent Robots (IRS) and Systems, Sandiego, CA, USA, Nov 2007.

[6] Horwimanporn Suppachai , Chaiyaporn Silawatchananai, Manukid Parnichkun, Chairit Wuthishuwong, Double Loop Controller Design for the Vehicle's Heading Control, Proceedings of the 2009 IEEE International Conference on Robotics and Biomimetics December 19 -23, 2009, Guilin, China.

[7] Y. S. Son, W. Kim, S.-H. Lee, and C. C. Chung, "Robust multirate control scheme with predictive virtual lanes for lane-keeping system of autonomous highway driving," IEEE Trans. Veh. Technol., vol. 64, no. 8,pp. 3378–3391, Aug. 2015.

[8] A. Merah, K. Hartani, and A. Draou, "A new shared control for lane keeping and road departure prevention," Vehicle Syst. Dyn., vol. 54, no. 1, pp. 86–101, 2016.

[9] Gilles Tagne, Reine Talj, Ali Charara. Higher-Order Sliding Mode Control for Lateral Dynamics of Autonomous Vehicles, with Experimental Validation. IEEE Intelligent Vehicles Symposium (IV 2013), Jun 2013, Gold Coast, Australia. pp.678-683. ffhal-00858299

[10] Alcala, E., Sellart, L., Puig, V., Quevedo, J., Saludes, J., Vazquez, D., & Lopez, A. (2016). Comparison of two non-linear model-based control strategies for autonomous vehicles. 2016 24th Mediterranean Conference on Control and Automation (MED). doi:10.1109/med.2016.7535921

[11] Anil Kunnappillil Madhusudhanan, Matteo Corno & Edward Holweg (2015): Sliding mode-based lateral vehicle dynamics control using tyre force measurements, Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility, DOI: 10.1080/00423114.2015.1066018.

[12] Kanghyun Nam, Sehoon Oh, Hiroshi Fujimoto, and Yoichi Hori. Design of Adaptive Sliding Mode Controller for Robust Yaw Stabilization of In-wheel-motor-driven Electric Vehicles, World Electric Vehicle Journal Vol. 5 - ISSN 2032-6653 - © 2012 WEVA

[13] Tabatabaei, S. H., Zahedi, A., & Khodayari, A. (2012). The effects of the Cornering Stiffness variation on Articulated Heavy Vehicle stability, 2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012). doi:10.1109/icves.2012.6294280.

[14] S Sahoo, SC Subramanian and S Srivastava, "Sensitivity Analysis of Vehicle Parameters for Heading Angle Control of an Unmanned Ground Vehicle" In Proceedings of ASME International Mechanical Engineering Congress and Exposition, Montreal, Quebec, Canada, November 14–20, 2014.

[15] Sahoo, S., Subramanian, S. C., & Srivastava, S. (2012). Design and implementation of a controller for navigating an autonomous ground vehicle. 2012 2nd International Conference on Power, Control and Embedded Systems. doi:10.1109/icpces.2012.6508073

[16] Zhengrong Chu, Christine Wu and Nariman Sepehri, Automated steering controller design for vehicle lane keeping combining linear active disturbance rejection control and quantitative feedback theory, Proc IMechE Part I: J Systems and Control Engineering 1–12 IMechE 2018.

[17] D. Nakhaeinia, S. H. Tang, S. B. Mohd Noor and O. Motlagh. A review of control architectures for autonomous navigation of mobile robots, International Journal of the Physical Sciences Vol. 6(2), pp. 169-174, 18 January, 2011.

[18] Chia-Feng Juang, Min-Ge Lai, and Wan-Ting Zeng. Evolutionary Fuzzy Control and Navigation for Two Wheeled Robots Cooperatively Carrying an Object in Unknown Environments, IEEE TRANSACTIONS ON CYBERNETICS, VOL. 45, NO. 9, SEPTEMBER 2015.

[19] Ghazi, I., ul Haq, I., Maqbool, M. R., & Saud, S. (2016). GPS based autonomous vehicle navigation and control system. 2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST). doi:10.1109/ibcast.2016.7429883 .

[20] Tzafestas, S. G. (2018). Mobile Robot Control and Navigation: A Global Overview. Journal of Intelligent & Robotic Systems, 91(1), 35–58. doi:10.1007/s10846-018-0805-9.

[21] Andreas, S., Ina, S., Jan Becker, C., & Walter, S. (2000). Navigation and Control of an Autonomous Vehicle. IFAC Proceedings Volumes, 33(9), 449–458. doi:10.1016/s1474-6670(17)38185-5.

[22] Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1), 23–33. doi:10.1109/100.580977.

[23] Guan, M., Wen, C., Wei, Z., Ng, C.-L., & Zou, Y. (2018). A Dynamic Window Approach with Collision Suppression Cone for Avoidance of Moving Obstacles. 2018 IEEE 16th International Conference on Industrial Informatics (INDIN). doi:10.1109/indin.2018.8472029.

[24] Garcia, F., Martin, D., de la Escalera, A., & Armingol, J. M. (2017). Sensor Fusion Methodology for Vehicle Detection. IEEE Intelligent Transportation Systems Magazine, 9(1), 123–133. doi:10.1109/mits.2016.2620398.

[25] Deshpande, Pawan. "Road Safety and Accident Prevention in India: A review." Int J Adv Engg Tech/Vol. V/Issue II/April- June 64 (2014): 68.

[26] Xsens MTI100 series: https://www.xsens.com/products/mti-100-series

[27] Hewson P. Method of estimating tyre cornering stiffness from basic tyre information. Proc IMechE Part D: J Automobile Engineering 2005; 219(12): 1407–1412.

Appendix

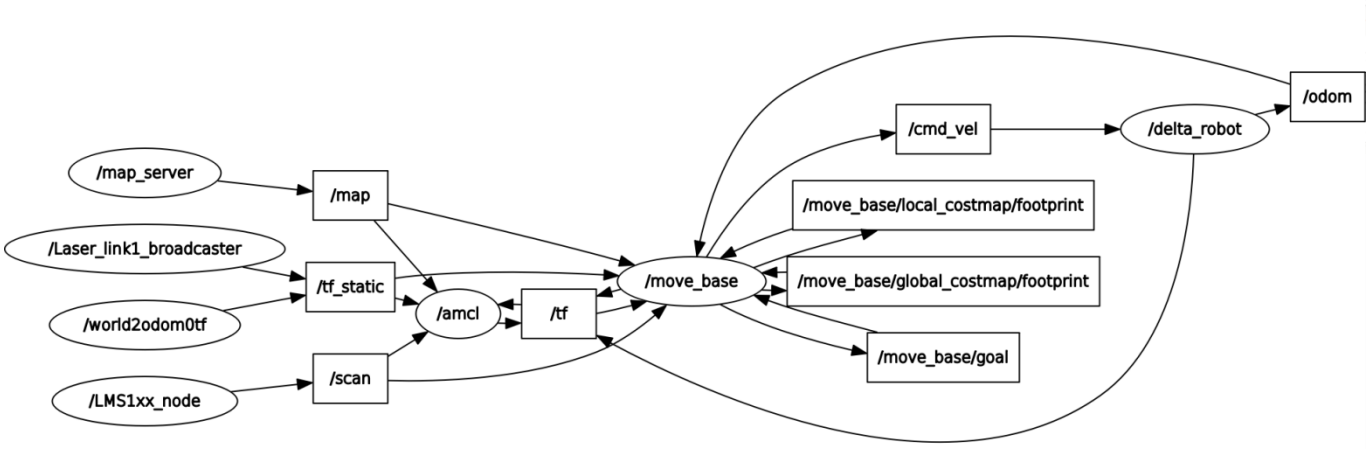
1. Odometry Code

```
  #include <string>
  #include <ros/ros.h>
#include <sensor_msgs/JointState.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>
int main(int argc, char** argv) {
ros::init(argc, argv, "state_publisher");
ros::NodeHandle n;
ros::Publisher odom_pub = n.advertise<nav_msgs::Odometry>("odom",
10);
// initial position
double x = 0.0;
double y = 0.0;
double th = 0;
// velocity
double vx = 0.4;
double vy = 0.0;
double vth = 0.4;
ros::Time current_time;
ros::Time last_time;
current_time = ros::Time::now();
last_time = ros::Time::now();
tf::TransformBroadcaster broadcaster;
ros::Rate loop_rate(20);
const double degree = M_PI/180;
// message declarations
geometry_msgs::TransformStamped odom_trans;
odom_trans.header.frame_id = "odom";
odom_trans.child_frame_id = "base_footprint";
while (ros::ok()) {current_time = ros::Time::now();
double dt = (current_time - last_time).toSec();
double delta_x = (vx * cos(th) - vy * sin(th)) * dt;
double delta_y = (vx * sin(th) + vy * cos(th)) * dt;
double delta_th = vth * dt;
x += delta_x;
y += delta_y;
th += delta_th;
geometry_msgs::Quaternion odom_quat;
odom_quat = tf::createQuaternionMsgFromRollPitchYaw(0,0,th);
// update transform
odom_trans.header.stamp = current_time;
odom_trans.transform.translation.x = x;
odom_trans.transform.translation.y = y;
odom_trans.transform.translation.z = 0.0;
odom_trans.transform.rotation = tf::createQuaternionMsgFromYa
w(th);
```

```
//filling the odometry
nav_msgs::Odometry odom;
odom.header.stamp = current_time;
odom.header.frame_id = "odom";
odom.child_frame_id = "base_footprint";
// position
odom.pose.pose.position.x = x;
odom.pose.pose.position.y = y;
odom.pose.pose.position.z = 0.0;
odom.pose.pose.orientation = odom_quat;
// velocity
odom.twist.twist.linear.x = vx;
odom.twist.twist.linear.y = vy;
odom.twist.twist.linear.z = 0.0;
odom.twist.twist.angular.x = 0.0;
odom.twist.twist.angular.y = 0.0;
odom.twist.twist.angular.z = vth;
last_time = current_time;
// publishing the odometry and the new tf

broadcaster.sendTransform(odom_trans);
odom_pub.publish(odom);
loop_rate.sleep();
}
return 0;
}
```

2. **Rqt_graph: nodes communicating with each other**



3. **Controller codes**

```
#include <ros/ros.h>
#include <sensor_msgs/JointState.h >
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>
```

```
#include <iostream>
using namespace std;
double width_robot = 0.1;
double vl = 0.0;
double vr = 0.0;
ros::Time last_time;
double right_enc = 0.0;
double left_enc = 0.0;
double right_enc_old = 0.0;
double left_enc_old = 0.0;
double distance_left = 0.0;
double distance_right = 0.0;
double ticks_per_meter = 100;
double x = 0.0;
double y = 0.0;
double th = 0.0;
geometry_msgs::Quaternion odom_qua t;
void cmd_velCallback(const geometry_msgs::Twist &twist_aux)
{
geometry_msgs::Twist twist = twist_aux;
double vel_x = twist_aux.linear.x;
double vel_th = twist_aux.angular.z;
double right_vel = 0.0;
double left_vel = 0.0;
if(vel_x == 0){
// turning
right_vel = vel_th * width_robot / 2.0;
left_vel = (-1) * right_vel;
}else if(vel_th == 0){
// forward / backward
left_vel = right_vel = vel_x;
}else{
// moving doing arcs
left_vel = vel_x - vel_th * width_robot / 2.0;
right_vel = vel_x + vel_th * width_robot / 2.0;
}
vl = left_vel;
vr = right_vel;
}
int main(int argc, char** argv){
ros::init(argc, argv, "base_controller");
ros::NodeHandle n;
ros::Subscriber cmd_vel_sub = n.subscribe("cmd_vel", 10, cmd_
velCallback);
ros::Rate loop_rate(10);
while(ros::ok())
{
double dxy = 0.0;
double dth = 0.0;
ros::Time current_time = ros::Time::now();
double dt;
```

```
double velxy = dxy / dt;
double velth = dth / dt;
ros::spinOnce();
dt = (current_time - last_time).toSec();;
last_time = current_time;
// calculate odomety
if(right_enc == 0.0){
distance_left = 0.0;
distance_right = 0.0;
}else{
distance_left = (left_enc - left_enc_old) / ticks_per_meter;
distance_right = (right_enc - right_enc_old) / ticks_per__
meter;
}
left_enc_old = left_enc;
right_enc_old = right_enc;
dxy = (distance_left + distance_right) / 2.0;
dth = (distance_right - distance_left) / width_robot;
if(dxy != 0){
x += dxy * cosf(dth);
y += dxy * sinf(dth);
}
if(dth != 0){
th += dth;
}
odom_quat = tf::createQuaternionMsgFromRollPitchYaw(0,0,th);
loop_rate.sleep();
}
}
```

4. Kalman filter matlab code

```
close all;
clear all;
clc;
r2d=180/pi;

gps=load('gps.txt');

gpst=gps(:,1)-gps(1,1);
gpsx=gps(:,2);
gpsy=gps(:,3);
gpshx=gps(:,4);
gpshy=gps(:,5);

odom=load('wheelodom.txt');
odomt=odom(:,1)-odom(1,1);
odomx=odom(:,2);
odomy=odom(:,3);
odomhx=odom(:,6);
odomhy=odom(:,7);
odomthetaerror=odom(:,8);
```

```
wheel_OdomZERO = zeros (length(odom));
q1=[wheel_OdomZERO(:,1) wheel_OdomZERO(:,2) odom(:,4) odom(:,5)];



wheelrot_data = quat2eul(q1);
wheelodom_pitch=wheelrot_data(:,2);
wheelodom_roll=wheelrot_data(:,1);
wheelodom_yaw=wheelrot_data(:,3);

imu=load('imu.txt');
imut=imu(:,1)-imu(1,1);
q=[imu(:,2) imu(:,3) imu(:,4) imu(:,5)];
r= quat2eul(q);
imu_pitch=r(:,2);
imu_roll=r(:,1);
imu_yaw=r(:,3);
imuthetaerror=0.1*odomthetaerror;

ofset = [gpsx(1);gpsy(1);0];
x_plot = [0];
y_plot = [0];
o_plot = [0];
prev = [0 0 0];
E_upd = eye(3).*0.01;
i = 1;
j = 1;
k = 1;

j=1;
for i=2:6635
   while odomt(i)>=imut(j) & j<length(imut)
      j=j+1;
      euang1=r(j-1,3);
   end;
   euang2=wheelrot_data(i,3);
   if(abs(euang1-euang2)<0.5)
      euang=euang1;
   else
      euang=euang2;
   end;
    %system variables
   del_trans = sqrt((odomx(i)-prev(1))^2+(odomy(i)-prev(2))^2);
   del_rot1 = atan2((odomy(i)-prev(2)),(odomx(i)-prev(1)))-prev(3);
   del_rot2 = euang-prev(3)-del_rot1;
   %state prediction
   x_pred = [x_plot(end) ; y_plot(end) ; o_plot(end)]+[(del_trans*cos(o_plot(end)+del_rot2)) ;
(del_trans*sin(o_plot(end)+del_rot2)) ; (del_rot1+del_rot2)];
   %state matrix
   A = [1 0 -del_trans*sin(o_plot(end)+del_rot2); 0 1 del_trans*cos(o_plot(end)+del_rot2); 0 0 1];
   prev = [odomx(i) odomy(i) euang];
   %covariance of system
   P = [odomhx(i) 0 0; 0 odomhy(i) 0; 0 0 odomthetaerror(i)];
```

```
    %covariance of prediction
    E_pred = A*E_upd*(A.') + P;
    x_upd = x_pred;
    E_upd = E_pred;
     if odomt(i)<=gpst(k+1)
        %covariance of measurement model
        Q = [gpshx(k) 0 0; 0 gpshy(k) 0; 0 0 5];
        %measurement model
        H = [1 0 0; 0 1 0; 0 0 1];
        %Kalman gain estimation
        K = E_pred*(H.')/(H*E_pred*(H.')+Q);
        %sensor values
        euang = r(j-1,3);
        z = [gpsx(k);gpsy(k);euang]-ofset;
%         gpsx = cat(2, gpsx, z(1));
%         gpsy = cat(2, gpsy, z(2));
        %state updation
        x_upd = x_pred + K*(z - x_pred);
        E_upd = (eye(3)- K*H)*E_pred;
        if(k<781)
            k = k + 1;
        end;
     end
    x(i)=x_upd(1);y(i)=x_upd(2);
    x_plot = cat(2, x_plot, x_upd(1));
    y_plot = cat(2, y_plot, x_upd(2));
    o_plot = cat(2, o_plot, x_upd(3));
end;

figure
plot(x_plot,y_plot,'-.',gpsx-ofset(1),gpsy-ofset(2),odomx,odomy,'--')
```

5. Rqt transformation tree