



Republic of Iraq

Ministry of Higher Education and Scientific Research

University of Kerbala

College of Engineering

Electrical and Electronic Engineering Department

**Neural Network Based MPC for Tracking
the Self Driving Car**

**A Thesis Submitted to the Council of the College of the
Engineering/University of Kerbala in Partial Fulfilment of the
Requirements for the Degree of Master of Science in Electrical
Engineering**

Written By:

Fatima Muneer Abdulrasul

Supervised By:

Assist. Prof. Dr Ahmed Abdulhadi Ahmed

Assist. Prof. Dr Haider Galil Kamil

April 2023

Ramadan 1444

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

وَأَن نُّنِيسَ لِلإِنسَانِ إِلهًا مَا سَعَى (39) وَأَن سَخِينَهَا سَوْفَ نِيرَى (40)

ثُمَّ نَجْرَاهُ لِحِرَاءَ (41)

صدق الله العلي العظيم

سورة النجم

Examination Committee Certification

We certify that we have read the thesis entitled " Neural Network Based MPC for Tracking the Self Driving Car " and as an examining committee, we examined the student " Fatima Muneer Abdulrasul " in its content and in what is connected with it and that in our opinion it is adequate as a thesis for the degree of Master of Science in Electrical Engineering.

Signature: 

Assist. Prof. Dr. Ahmed Abdulhadi Ahmed
(First Supervisor)

Date: 27 / 4 / 2023

Signature: 

Assist. Prof. Dr. Haider Galil Kamil
(Second Supervisor)

Date: 27 / 4 / 2023

Signature: 


Assist. Prof. Dr. Abdal-Razzak Shehab Hadi
(Member)

Date: 30 / 4 / 2023

Signature: 

Dr. Dhirgaam A. Kadhim
(Member)

Date: 27 / 4 / 2023

Signature: 

Prof. Dr. Mithaq Nama Raheema
(Chairman)

Date: 27 / 4 / 2023

Approval of the Department of Electrical
and Electronic Engineering.

Signature: 

Name: Prof. Dr. Haider Ismael Shahadi

(Head of Electrical and Electronic
Engineering Dept.)

Date: 7 / 5 / 2023

Approval of Deanery of the College
of Engineering/ University of Kerbala.

Signature: 

Name: Prof. Dr. Laith Sh. Rasheed

(The Dean of the College of Engineering)

Date: / / 2023

Undertaking

I certify that the research work entitled " Neural Network Based MPC for Tracking the Self Driving Car " is my own work. The work has not been presented elsewhere for assessment. Where the material has been used from other sources it has been properly acknowledged/referred.

Signature:



Fatima Muneer Abdulrasul

Date: 7/9/2023

Acknowledgements

First and foremost, I would like to express my sincere gratitude and thanks to **Allah** the almighty who blessed us with gave us science and health and helped us to complete this project.

I would like to express my deep gratitude to my supervisor **Asst. Prof. Dr Ahmed Abdulhadi Ahmed** for his advice, guidance, encouragement, and supervision throughout this work.

I would like to thank **Dr Haider Galil Kamil**, for providing valuable assistance in this thesis.

I would like to express my thanks and gratitude to the people and friends for their help, moral support, and nice words during this work.

I would like to offer thanks and appreciation to my lovely family for their help and encouragement, those who have given me love, compassion, safety, and hope.

My God bestows health and happiness upon all of them.

Fatima

DEDICATION

To my father, who supported me

To my lovely mother, who planted love in my heart.

To My Dear Life Partner... My Beloved Husband

To My Beloved Daughter Dima

To My Sincere Brothers and Sisters

To All People Who Love Me Deeply

Abstract

The road traffic environment is highly variable and unpredictable. Autonomous cars operating in such environments may face unexpected critical scenarios, where the risk of an accident rapidly increases compared to normal driving situations. These scenarios may arise due to unforeseen behavior from other road users or obstacles appearing on the road. In such crucial conditions, the primary objective of car motion control is to minimize the danger of an impending accident.

The purpose of this study is to develop a system that can help prevent accidents in unpredictable and variable road traffic environments by addressing the problem of motion planning and control in critical situations for autonomous cars. The system generates optimal paths and control inputs for the car to follow while avoiding obstacles and following the center of the track predictably. To achieve this objective, motion planning technique for self-driving cars are presented.

The motion planning technology utilized in this study is based on the A* and potential field algorithms, with an intelligent controller consisting of prediction supported by neural network technology. The model predictive controller predicts the car's future for a finite time horizon using a mathematical model of the car. The controller utilizes a linearized and discretized kinematic bicycle model as an internal car model. The A* strategy path is used as it is a powerful tool for solving pathfinding problems due to its optimality, efficiency, admissibility, flexibility, and potential function for path planning in an environment with obstacles. The potential function is used due to its simplicity, safety, and low computational cost. The control inputs are the car's steering angle and acceleration.

The model predictive controller solves the optimization problem as a quadratic programming problem that minimizes a cost function while satisfying a set of constraints. The neural network is used to adjust the rate of change of the steering angle value, adjusting the rate of steering angle is an important part of optimizing a self-driving car's performance and ensuring its safety and reliability on the road. Without adjusting the rate of steering angle, the car's ability to make precise turns and stop accurately may be compromised, which can lead to a higher risk of accidents and lower fuel efficiency. The cost function includes a set of objectives, including errors in desired and current states, inputs, and the rate of change of inputs, to guide the self-driving car away from high-cost regions. The decision-making module determines the next course of action for the car. After the subsequent maneuver is selected, a velocity profile and a lane center reference are generated for the self-driving car to track.

The quadratic programming problem is solved using convex optimization in the optimization tool of python, with a sample time of 0.1 seconds. The control parameters, including the cost function weights and the length of the horizon, are adjusted to make the lane safe and comfortable. The selected horizon length ranges from 8 m to 12 m, within which the control unit ensures that the car follows the intended path while avoiding obstacles.

All the results were achieved within the constraints of the specified car, with a maximum speed of 15 m/s, maximum reverse speed of 5 m/s, maximum steering angle of 45° , maximum steering rate of 30° , maximum deceleration of 6 m/s^2 and a maximum for acceleration 2.5 m/s^2 .

Table of Contents

Abstract.....	I
List of Contents	II
List of Symbols	v
Abbreviations	v
List of Tables	v
List of Figures.....	v
Chapter One: Introduction	1
1.1 Overview	1
1.2 Thesis Background	3
1.3 Problem Formulation.....	5
1.4 Objectives.....	6
1.5 Thesis Organization.....	6
Chapter Two: Literature Review	7
2.1 Introduction	7
2.2 Related Work.....	7
Chapter Three: System Modelling	19
3.1 Introduction	19
3.2 Self-Driving Car Structure	19
3.2.1 Route Planning	20
3.2.2 Behavioral Decision Making	20
3.2.3 Motion Planning	21
3.2.4 Control System	22

3.3 System Modelling	23
3.3.1 Path Planning.....	23
3.3.2 Controller Design	37
Chapter Four: The Proposed Work.....	55
4.1 Python Implementation	55
Chapter Five: Results and Discussion.....	63
5.1 Overview	63
5.2 Motion Planning Algorithm Results	63
5.2.1 Dijkstra Algorithm.....	63
5.2.2 A* Algorithm.....	65
5.2.3 D* Algorithm.....	70
5.2.4 Potential Field Algorithm	72
5.3 Using a Neural Network with Predictive Control	78
5.4 Obstacle Avoidance in a Constrained Environment Results.....	81
Chapter Six: Conclusions and Recommendations.....	114
6.1 Conclusions	114
6.2 Future Work	115
References	116
الخلاصة.....	124

List of Symbols

Symbol	Definition
ζ	Attractive potential coefficient
η	Repulsive potential coefficient
Q	State error weightage matrix
R1	Input weighted matrix
R2	Rate of input change weightage matrix
D	Final state weighted matrix

Abbreviations

MPC	Model Predictive Control
NN	Neural Network
N-MPC	Neural Network-based Model Predictive Control
NNPM	Neural Network Plant Model

List of Tables

Table 3-1 Most Common Types of Heuristic Functions Used in Path Planning Algorithms.....	29
Table 3-2 Mathematical equations for activation function	49

List of Figures

Fig. 1.1 levels of autonomous vehicles	3
Fig. 3.1 An illustration of the decision-making process hierarchy[11].....	20
Fig. 3.2 Dijkstra example [38]	26
Fig. 3.3 Flowchart illustrates the Dijkstra algorithm, which is used in path planning.....	27
Fig. 3.4 A* example	28
Fig. 3.5 Path planning flowchart using the A* algorithm	30
Fig. 3.6 State lattice planner when the agent's visibility is 1 unit	33

Fig. 3.7 State lattice planner when the agent's visibility is 7 unit	34
Fig. 3.8 State lattice planner when the agent's visibility is 15 unit	34
Fig. 3.9 MPC strategy[49]	39
Fig. 3.10 Basic structure of MPC [51]	41
Fig. 3.11 The structure of the neural network plant model [59].....	48
Fig. 3.12 Convex optimization (one point on the convex hull, is closest to the minimum)	54
Fig. 4.1 Diagram of N-MPC	60
Fig. 4.2 Neural network flowchart.....	61
Fig. 4.3 System implementation flowchart.....	62
Fig. 5.1 Dijkstra algorithm path planning from (20,25) to (20,0)	64
Fig. 5.2 Dijkstra algorithm path planning from (45,45) to (0,0)	64
Fig. 5.3 Dijkstra algorithm path planning from (5,185) to (140,0)	65
Fig. 5.4 A* algorithm path planning from (-30,-40) to (20,20),.....	67
Fig. 5.5 A* algorithm path planning from (40,0) to (-20,20),.....	67
Fig. 5.6 A* algorithm path planning from (20,-20) to (-20,20),.....	68
Fig. 5.7 A* algorithm path planning from (20,25) to (20,0)	68
Fig. 5.8 A* algorithm path planning from (45,45) to (0,0)	69
Fig. 5.9 A* algorithm path planning from (5,185) to (140,0)	69
Fig. 5.10 D* algorithm path planning from (20,25) to (20,0)	70
Fig. 5.11 D* algorithm path planning from (45,45) to (0,0)	70
Fig. 5.12 D* algorithm path planning from (5,185) to (140,0)	71
Fig. 5.13 The problem of local minima potential filed	73
Fig. 5.14 System response where $\zeta = 1, \eta = 1$	74
Fig. 5.15 System response where $\zeta = 1, \eta = 100$	75
Fig. 5.16 System response where $\zeta = 100, \eta = 1$	76
Fig. 5.17 System response where $\zeta = 100, \eta = 100$	77
Fig. 5.18 System steering angle response with and without use neural network	78
Fig. 5.19 System acceleration response with and without use neural network.....	79
Fig. 5.20 System velocity response with and without use neural network	79
Fig. 5.21 A* path planning from starting point (-20,20) to the target point (-20, -40).....	82
Fig. 5.22 Avoid path obstacle by using the potential field in local planne	82
Fig. 5.23 Result static obstacle simulation path following from (-20,20) to (-20, -40).....	83
Fig. 5.24 Result dynamic obstacle simulation path following from (-20,20) to (-20, -40)	84
Fig. 5.25 Input cost weights low, path from (-30,40) to (20,20).....	86

Fig. 5.26 Input cost weights low, path from (-20,20) to (40,0).....	87
Fig. 5.27 Input cost weights high, path from (-30,40) to (20,20)	88
Fig. 5.28 Input cost weights high, path from (-20,20) to (40,0)	89
Fig. 5.29 State error cost weights low, path from (-30,40) to (20,20).....	91
Fig. 5.30 State error cost weights low, path from (20,20) to (40,0)	92
Fig. 5.31 State error cost weights high, path from (-30,-40) to (20,20).....	93
Fig. 5.32 State error cost weights high, path from (-20,20) to (40,0)	94
Fig. 5.33 Rate input cost weights low, path from (-30,-40) to (20,20)	96
Fig. 5.34 Rate input cost weights low, path from (-20,20) to (40,0)	97
Fig. 5.35 Rate input cost weights high, path from (-30,-40) to (20,20)	98
Fig. 5.36 Rate input cost weights high, path from (-20,20) to (40,0).....	99
Fig. 5.37 Terminal cost weights low, path from (-20,20) to (40,0)	101
Fig. 5.38 Terminal cost weights low, path from (-30,-40) to (20,20)	102
Fig. 5.39 Terminal cost weights high, path from (-20,20) to (40,0).....	103
Fig. 5.40 Terminal cost weights high, path from (-30,-40) to (20,20).....	104
Fig. 5.41 Low Horizon length effect, path from (-35,0) to (20, -20)	106
Fig. 5.42 Low horizon length effect, path from (20, -20) to (-20,20)	107
Fig. 5.43 High horizon length effect, path from (-35,0) to (20,-20)	108
Fig. 5.44 High horizon length effect, path from (20, -20) to (-20,20)	109
Fig. 5.45 Suitable horizon length effect, path from (-35,0) to (20, -20).....	110
Fig. 5.46 Suitable horizon length effect, path from (20, -20) to (-20,20).....	111

Chapter One: Introduction

1.1 Overview

When a person takes on the role of the driver in the morning, they are typically confident in their ability to safely reach their destination. However, this is becoming increasingly inaccurate. There are many reasons for a car accident: Driver negligence, bad weather, poor road conditions, and third-party carelessness which can cause accidents that lead to deaths or serious injuries.

In reference to [1], the statistical data reveals the following:

- Around 1.36 million people die every year in road accidents; An average of 3,700 people is killed every day. In other words, one person is killed every 25 seconds.
- An additional 20-50 million people are injured but do not die, frequently leading to long-term impairment.
- Car crashes are the primary source of death among youngsters between the ages of 5-29. Young people aged 15-44 make up more than half of all deaths.
- Road accidents may cost countries between 2-8% of their gross domestic product.
- In direct medical expenditures, traffic accidents cost the United States more than \$380 million.

In the United States, car accidents are the greatest cause of death for children aged 1-3 years. One of the most heart-breaking realities regarding car accidents is that the majority of them are avoidable. According to a 2016 "National Highway Automobile Safety Administration (NHTSA)" research, human error accounts for 94% to 96% of all traffic accidents.

These human errors include [2]:

- High speed
- Aggressive/Reckless driving
- Distracted driving
- Drunk driver
- Sleepy driving

Moreover, in the era of huge technological progress in which we live, technologies like cars are becoming more and more affordable to the point that almost every family owns at least one car. Thus, the number of accidents increased exponentially. As a result, there is a hole in the market for self-driving car. In simple language, self-driving or autonomous cars can be called mobile robot. This car can sense the environment, understand the surrounding scene, and make decisions without human interaction from the road to the destination. Self-driving cars have gone from "likely possible" to "definitely possible" to "inevitable". The Society of Automotive Engineers (SAE) has created a "Levels of Driving Automation standard that defines the six levels of driving automation, from Level 0 (no automation) to Level 5 (fully autonomous)" [3], as shown in Fig. 1.1.

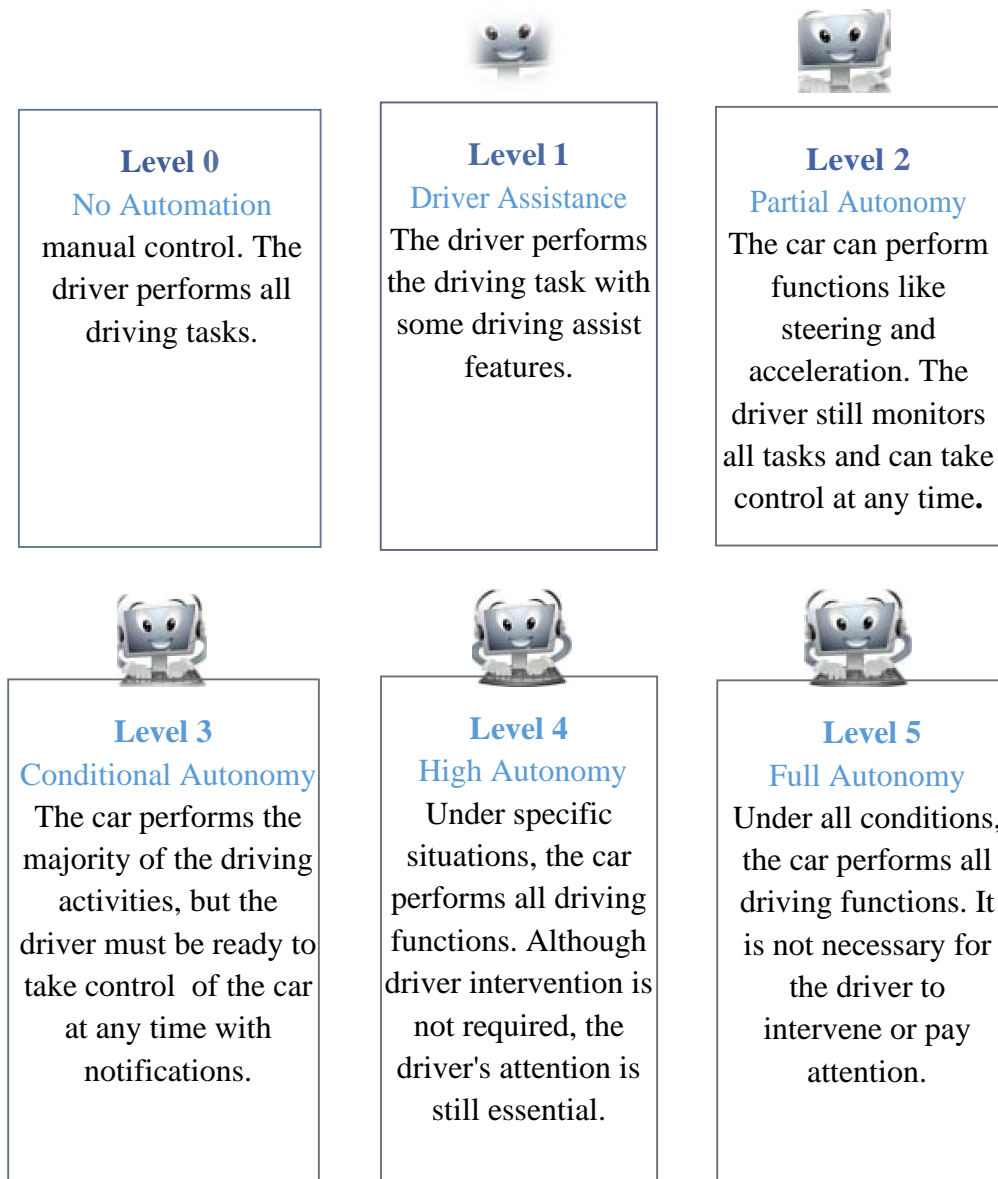


Fig. 1.1 levels of autonomous cars

1.2 Thesis Background

The topic of navigation is one of the most critical in robotics research. Mobility is required for all autonomous mobile robots in order to execute, locate, plan movement, and direct. In this context, navigation is defined as the act of planning a moving robot's path from its current location to the desired

target location, following the intended path and avoiding any obstacles encountered along the way[4].

To ensure the safety and practicality of navigation, a number of parameters must be met by the needed routes. In addition, pathways may be described in terms of requirements; for example, in highly dynamic situations, rapid or smooth paths are often preferable over long and curvy paths[5].

The navigation challenge requires interaction with changes in the environment model in addition to route planning. The robots must travel fast to the goal while avoiding static or dynamic impediments detected by their sensors, which requires excellent trajectory planning and obstacle avoidance. Despite extensive research on these areas, there is still no conclusive answer to the difficulty of navigation in busy, dynamic surroundings [6].

Many navigation technologies from mobile robots have been adapted to suit the problems of road networks and driving restrictions. These planning approaches are classified into four classes based on their applicability in autonomous driving: graph searching, sampling, interpolation, and numerical optimization [7].

Intelligent control is becoming increasingly crucial in our society as route planning improves and information technology advances. Compact devices have tiny sizes, minimal power consumption, and strong functionalities, among other characteristics, this field is poised to have a diverse array of applications, including automobile electronics, aircraft, and smart homes. If any of these technologies are merged, it will lead to more intelligent applications.

The self-driving car was selected as the research platform, and the predictive control model was used as the central control unit. The intelligent car can move independently with intelligent control while following the path.

Intellectual activities such as executing motion planning algorithm, steering drive direction and brake commands can be incorporated in the intelligent car application. The applications of the self-driving car technology are used in:

1. Autonomous Robotic Systems.
2. Auto-Pilot in the Airplane.
3. Probe used in space exploration.
4. Transfer robot
5. Agricultural robot

1.3 Problem Formulation

After mentioning the proportion of human-caused accidents, it can be concluded that most car accidents can be avoided by keeping the humans away from the driving process.

The overall objective of this thesis is to create an autonomous car system capable of comfortably and safely navigating with minimal jerks by finding a solution to the problem of path planning in environments containing road obstacles and to design an intelligent controller for an autonomous car capable of tracking certain paths. After determining what is required for the performance of an autonomous car, it is possible to formulate the problem addressed in this thesis:

- The car needs longitudinal control to maintain acceleration and speed.
- The car needs lateral control to steer the car along the desired path.
- The car must maintain safety distances, speed limits and acceleration limits.
- Supporting the controller with intelligent techniques such as neural networks.

1.4 Objectives

The aims of this thesis are:

- Develop a path planning by using global and local planners to achieve computational efficiency in dealing with changing traffic environments in different road scenarios.
- Design an intelligent controller that manipulates motion planning.
- Integrate the controller with a path planning algorithm to minimize the tracking error as a result of continuous updating of the path with the control.

1.5 Thesis Organization

This thesis consists of five chapters, which are briefly introduced as follows:

1. The first chapter explained the background of self-driving cars and their applications.
2. The second chapter discusses previous studies and researches that have been conducted related to the self-driving car.
3. The third chapter presents the general structure of the self-driving car model, path planning algorithms (Dijkstra, A*, D*, state lattice, potential function) and the control design.
4. The fourth chapter presents the proposed work.
5. The fifth chapter explains the simulation results and their discussion.
6. The sixth chapter shows the conclusions of the current study and the suggested recommendations for future studies.

Chapter Two: Literature Review

2.1 Introduction

An autonomous car transfers a manual driving car to autonomous driving using different sensors and actuators that decide on driving based on other criteria. In order to understand the evolution of research on self-driving in recent years, it is important to conduct a literature review to understand the different application areas from which self-driving has developed as well as to identify research gaps. Therefore, in the following sections, a review of the literature is presented.

2.2 Related Work

Several researchers presented a variety of studies to construct a self-driving automobile system that incorporates a perception system and a decision-making system. The perception system is separated into multiple subsystems that are in charge of self-driving car localisation, mapping of stationary objects, detection and tracking of moving obstacles, road mapping, and traffic sign detection and identification. On the other hand the decision-making system is separated into many subsystems that are in charge of path planning, behaviour selection, action planning, and control. Some research from different decision-making systems has been reviewed as follows:

P Falcone et al. (2008) [8] suggested a control strategy that combines MPC with steering control devices as well as two model predictive controllers. The first was used in the all-wheel drive model, which changed the steering angle and brake torque to follow the intended trajectory. A modified bike model with fewer inputs was used to create the second MPC

controller. The findings reveal that the first microcontroller performs well in both low and high-speed monitoring of the reference route, however calculation is time consuming. The second controller, on the other hand, performed poorly at high speeds due to the simplicity of the automobile model but suitable for real-time execution.

V T Minh (2016) [9] presented an approach for controlling and planning the path of autonomous robots using Nonlinear Model Predictive Control (NMPC) and Feasible Path Planning (FPP) techniques. The approach considers the dynamics and constraints of the robot, as well as environmental obstacles, in order to plan feasible paths and generate control signals that enable the robot to follow these paths while avoiding collisions. The NMPC algorithm optimizes the robot's control inputs over a finite time horizon, while the FPP algorithm plans the robot's path based on the current environment and the robot's constraints. The proposed approach has been validated through simulations and experiments on a mobile robot platform, demonstrating its effectiveness in controlling and navigating the robot in dynamic environments.

A Koga et al. (2016) [10] implemented the lateral and longitudinal control subsystems using the MPC technique, which predicts autonomous vehicle motion using the standard bike model. The proposed autonomous driving system was tested on a small-scale experimental track at a speed of 20[km/h] with seven different parameter settings. The results showed that the system was capable of following a reference path with small deviations and smooth operation. While the overall driving performance of the model predictive controller was inferior to that of human drivers, the system was able to produce a range of different driving characteristics by putting different

weights on tracking precision and steering smoothness. The performance of the controller could potentially be improved by setting more accurate values for the physical parameters of the vehicle using system identification techniques.

B Paden et al. (2016) [11] provided a survey of driverless vehicle decision-making problems with a focus on motion planning and feedback control. The breakdown of decision-making into individual problems has allowed for the use of well-developed solution techniques from a variety of research areas. However, tailoring and integrating these methods so that their interactions are semantically valid remains a challenge. Additionally, the computational burden of the entire system is an issue that needs to be resolved. Nonetheless, these issues do not limit the potential of driverless vehicles as a means of personal mobility.

C Götte et al. (2016) [12] proposed a model predictive control (MPC) approach for guiding a vehicle laterally. The aim of this approach is to generate real-time steering commands for a vehicle that are safe, smooth, and comfortable for passengers. The authors first present the mathematical model of the vehicle and describe how it can be used to generate reference trajectories for the vehicle to follow. Next, the authors introduce the MPC approach, which involves solving an optimization problem at each time step to generate the optimal steering command. Finally, the authors evaluate the performance of their approach through simulations and experiments on a test track. In the first scenario, the system successfully avoids a collision with two static obstacles by performing a double lane change maneuver. The planned trajectory is adjusted as soon as the first obstacle appears within the prediction horizon, and the system waits until the last-point-to-steer is detected before

following the reference trajectory. The resulting maneuver reaches the limits of driving physics, but remains stable and collision-free. In the second scenario, the system is tested with a dynamic obstacle, and it is able to adjust the planned trajectory at an early stage to avoid a collision. The moving obstacle is taken into account, and the resulting maneuver is stable and collision-free.

M Bojarski et al. (2016) [13] discussed a deep neural network approach to autonomous driving, specifically for lane and road following. The authors demonstrate that their convolutional neural network (CNN) can learn to perform this task without the need for manual decomposition into sub-tasks such as road or lane marking detection, semantic abstraction, path planning, and control. They evaluate the network's performance through simulation tests and on-road tests, and visualize the internal state of the CNN to show how it learns to detect useful road features on its own. The authors conclude that their approach is promising, but more work is needed to improve the network's robustness.

Y Nishio et al. (2017) [4] proposed a method for obstacle avoidance that combines the fuzzy potential method and model predictive control. While the fuzzy potential method can handle the shape and attitude of the robot and achieve obstacle avoidance through translational and rotational motion, it cannot explicitly include the dynamics of the robot and the motion of obstacles. On the other hand, model predictive control considers the dynamics of the robot and predicts its motion while handling the constraint explicitly through an index function. By considering the mobility range of obstacles as constraints, it guarantees obstacle avoidance. The proposed method was

verified through numerical simulations and was found to be effective even in complex situations where conventional methods fail.

J Rios-Torres et al. (2017) [14] stated that reservations were one of the ways used to address the various initiatives in the literature to coordinate "Connected and Automated Vehicles" (CAVs) to enhance traffic flow and safety in certain transportation sectors. The biggest difficulty with this technique is the high level of communication required and the possibility of barriers. They reported that the most prevalent issue was minimising travel time. Alternative formulations, on the other hand, include reducing compound interference at the intersection region. Also, they investigated multi-objective optimisation factors such as acceleration, speed tracking error, and collision risk. Furthermore, they used traffic flow modelling to create control inputs that guarantee traffic flow at the intersection remains stable.

G Bresson et al. (2017) [15] suggested a field survey of simultaneous localisation and mapping. They began by discussing the limits of traditional autonomous driving systems before moving on to the requirements required for this sort of application. Then, they examined how the highlighted difficulties are being addressed. It focused on techniques to creating and reusing long-range maps under various settings (weather, season, etc.) and finished by providing an overview of the numerous, large-scale experiments done to date as well as outlining remaining obstacles and future perspectives.

C Bila et al. (2017) [16] Provides an overview of research on support and services offered by information and communication technology for the safety of future linked cars. Vehicle detection, route detection, , pedestrian detection, collision avoidance, and drowsiness detection are the primary

classifications and a brief summary of the areas of concentration for research and development in this approach. These applications assist drivers and reduce the chance of an accident.

C Liu et al. (2017) [17] suggested a unified method to route planning based on a predictive control model (MPC), with the aim of automatically determining manoeuvre placement while ensuring safety. To achieve this, neighbouring cars were represented as polygons and an MPC constraint was created to enforce collision avoidance between the ego vehicle and surrounding vehicles. A lane-related potential field was also integrated into the MPC's objective function to ensure safe and smooth manoeuvres. The MPC path planner was evaluated through simulations in three scenarios: normal highway driving, ramp merging, and intersection crossing. In the normal highway driving scenario, the path planner successfully planned a path for the ego vehicle to maintain a safe distance from surrounding vehicles and make a lane change when feasible. The simulation displayed the trajectory of the ego vehicle and surrounding vehicles, with the speed of each vehicle shown in subplots. In the ramp merging scenario, the path planner generated a safe longitudinal path for the ego vehicle to merge into the merging lane between two surrounding vehicles. The ego vehicle first accelerated to catch up with the gap and then decelerated to keep a comfortable distance from the car in front before successfully merging into the lane. In the intersection crossing scenario, the path planner planned a path for the ego vehicle to approach a stop sign, stop there, and remain in the "stop" state until it became safe to cross the intersection.

R Guidolini et al. (2017) [18] suggested a "Neural Model Predictive Control (N-MPC)" technique to overcome delays in the "Intelligent and Autonomous Robotic Automobile (IARA)". Due to the intricacy of the dynamics of IARA's reaction to stimuli, they attempted to create a neural network in N-MPC utilising this neural model. The experimental findings revealed that "N-MPC outperformed PID control" by eliminating the influence of IARA guidance station delays, allowing IARA autonomous running at speeds up to 37 km/h with a 48% improvement in maximum constant speed.

D Cairano et al. (2018) [19] provided a high-level description of a real-time optimisation issue for automotive and aerospace applications, with a focus on the MPC controller. The cost function and system limitations were used to define the optimum control issue. Also covered were numerical algorithms and their implementations on an embedded computer platform.

G Williams et al. (2018) [20] provided an information-theoretical approach to optimize random control issues that utilised to develop broad sampling-based optimisation strategies. This novel mathematical technique was utilised to create a sampling-based model-based predictive control algorithm. They assessed the performance of the Information Theoretical Model Predictive Control Scheme (IT-MPC) to a typical predictive control version of the entropy technique on a demanding autonomous driving job over an earthy test track.

C M Martinez et al. (2018) [21] suggested an important contribution by adding aspects impacting driving style and classification methodologies for intelligent automobile control applications, as well as implementation

restrictions. The intricacy of driving style is examined while assessing current interpretations of safety and fuel economy through the use of various algorithms. The continual advancement of Advanced Driver Assistance Systems and vehicle autonomous driving capabilities necessitates a more in-depth examination of driving style and the incorporation of drivers in the systems. This has prompted the development of data-driven algorithms capable of processing larger amounts of data, as well as the deployment of machine learning algorithms capable of adapting to individual drivers.

M V Smolyakov et al. (2018) [22] described the use of a car simulator to generate data for training neural networks to predict the steering angle of a car. The authors developed two different neural network architectures, one with convolutional layers and one with additional regularization and batch normalization layers. They trained these networks using data generated from the car simulator, which included images from the left, center, and right cameras and the corresponding steering angle. The results showed that the second architecture with regularization and batch normalization layers performed better and had fewer parameters. The authors suggest that this architecture may be suitable for testing on embedded systems. They also suggest adding recurrent layers to the network in future work to better predict the steering angle based on the data sequence. The authors conclude that using a simulator to generate data is efficient and avoids the need for expensive or resource-intensive data collection from the real world.

J Wang et al. (2019) [23] explored networking and communication technologies for autonomous driving to improve the perception and planning ability of autonomous vehicles. The study covered intra-vehicle and inter-vehicle networks, discussing various technologies suitable for autonomous

vehicles, including data bus wired interconnection, Ethernet, power-line communication, and wireless interconnection. The inter-vehicle network was further discussed, with a focus on low power technologies, 802.11 family technologies, base station driven technologies, and other auxiliary technologies. New trends in communication technologies, such as 5G, computing technologies, SWIPT, VLC, and deep learning, were also introduced. Verification methods, challenges, and open issues were summarized, highlighting the need for joint efforts between academia and industry to advance networking and communications for autonomous driving.

Y Wang et al. (2019) [24] discussed the use of deep learning models in (Intelligent Transportation Systems and their applicability in different tasks such as computer vision, time series prediction, classification, and optimization. It was explained that deep learning models could be applied if the problem could be formulated as a classification, regression, or Markov Decision Process problem, and sufficient training data and Graphics Processing Unit resources were available. The advantages of deep learning models, such as achieving state-of-the-art performance in various classification and prediction tasks, were also highlighted. However, the limitations of deep learning models were acknowledged, including their reliance on specific amounts of data and computing resources, the difficulty in parameter tuning, and the lack of interpretability in their black-box representations.

J Gwak et al. (2019) [25] development of commercial autonomous driving research was reviewed. Several companies have developed self-driving technologies and vehicles using their own systems and algorithms. The technologies used by these companies were compared based on the

sensors used in self-driving vehicles. Tesla developed its self-driving technology called "Autopilot," which uses 8 cameras, 12 ultrasonic sensors, and a radar sensor. Google's self-driving technology, Waymo, uses a vision system, lidar system, radar system, and supplemental sensors. Uber also developed self-driving technology using lidar, cameras, radar, GPS, a self-driving computer, telematics, ultrasonic sensors, and a vehicle interface module. General Motors (GM) developed self-driving technology for its Volt EV vehicles using multiple cameras, lidar sensors, a radar sensor, and 4G LTE Connected.

A Reda et al. (2020) [26] explored MPC and adaptive MPC controller implementations to operate an autonomous vehicle steering system. The implementations were carried out for systems with both constant and variable dynamics. The results demonstrated that the MPC controller gives adequate control for a constant dynamics system, but it cannot manage changing operating circumstances, while adaptive MPC provides adequate control for changing dynamics systems.

K Muhammad et al. (2020) [27] identified the primary advantages of safe learning techniques and assessed existing approaches for safe autonomous driving that encompass significant accomplishments and limits. Furthermore, they identified the primary embodiments of the self-sustaining driving pipeline, which are measurement, analysis, implementation (also known as control processes), and evaluation of the performance of deep learning methods for various safety-related tasks such as road, vehicle, track, drowsiness, pedestrian, traffic light detection and collision avoidance.

Y Xu et al. (2021) [28] proposed a path-tracking strategy that incorporates predictive model control (MPC) and "preview follower theory (PFT)", as well as a reference generation unit and an MPC controller. The reference generating unit can use PFT to determine the lateral reference acceleration at the sample point and create the reference diffraction rate at each prediction point. PFT improves the accuracy of the diffraction rate computation as the sample range rises. The MPC controller can achieve optimal reference route tracking with physical constraints. To create an online predictive model from nonlinear to continuous linear vehicle dynamics, the MPC issue was written as a "linear time-varying (LTV)".

S Kolachalama et al. (2022) [29] introduced a novel driving mode called "Intelligent Vehicle Driving Mode (IVDM)", which improves the vehicle's engine performance in real-time without increasing flight time under normal driving situations. When running, IVDM engages adaptive cruise control (ACC); hence, longitudinal acceleration (LOT) was automatically calculated by the ACC, and the parameters of lateral acceleration [LAT] and yaw rate [YAR] were estimated using particular mathematical models that assumed idealised steering behaviour (ISB). They created an Autonomous External Input Regression Network (NARX) for deep learning models.

J L Vazquez et al. (2022) [30] suggested resolving the movement prediction issue as a policy learning problem in a novel approach. The policy is taught by model-based simulated learning, which, in conjunction with the Interactive Multi-Agent Prediction Policy (IMAP), enables us to comprehend a highly interactive prediction model/policy. Based on the optimal response frequencies, the two interactive motion planners offered based on this model. One is inspired by the leader-follower structure, while the other is derived

from the Nash equilibrium. In genuine driving scenarios, simulation results are visible. The prediction architecture and interactive deep motion planning can handle difficult lane changes as well as hostile activities such as halting another vehicle. Each of their offered approaches are capable of planning the challenge of interacting motions and accurately predicting the effect of a certain action on other factors.

These works of literature review are the most related papers in the field of decision systems involving path planning and control in the self-driving car, controlled with different control techniques, including predictive, adaptive, neural network, deep learning, and hybrid control systems. The literature has proven the effectiveness of the proposed motion planning and control systems used in the self-driving car system.

Chapter Three: System Modelling

3.1 Introduction

The autonomous car uses the perception module to see its environment and the planning module to make decisions and create paths. The controller is responsible for controlling car moving by generating steering wheel angle (lateral control) and acceleration (longitudinal control).

In this chapter, the general structure of the self-driving car model is presented in Section 3.2. Section 3.3, presented a theoretical model in detail. The model mainly consists of two parts, the first part is about path planning algorithms and the second part is the car control and it has four sections, the first section is the use of the predictive model, the second section is the use of the neural network, the third section is the mathematical model, and the fourth section about convex optimization.

3.2 Self-Driving Car Structure

One common approach to create a self-driving car system is to organise sensor perception and decision-making in a hierarchical structure as shown in Fig. 3.1. The decision-making unit of a self-driving automobile is represented by four components: route planning, behavioural layer, motion planning, and control system [11].

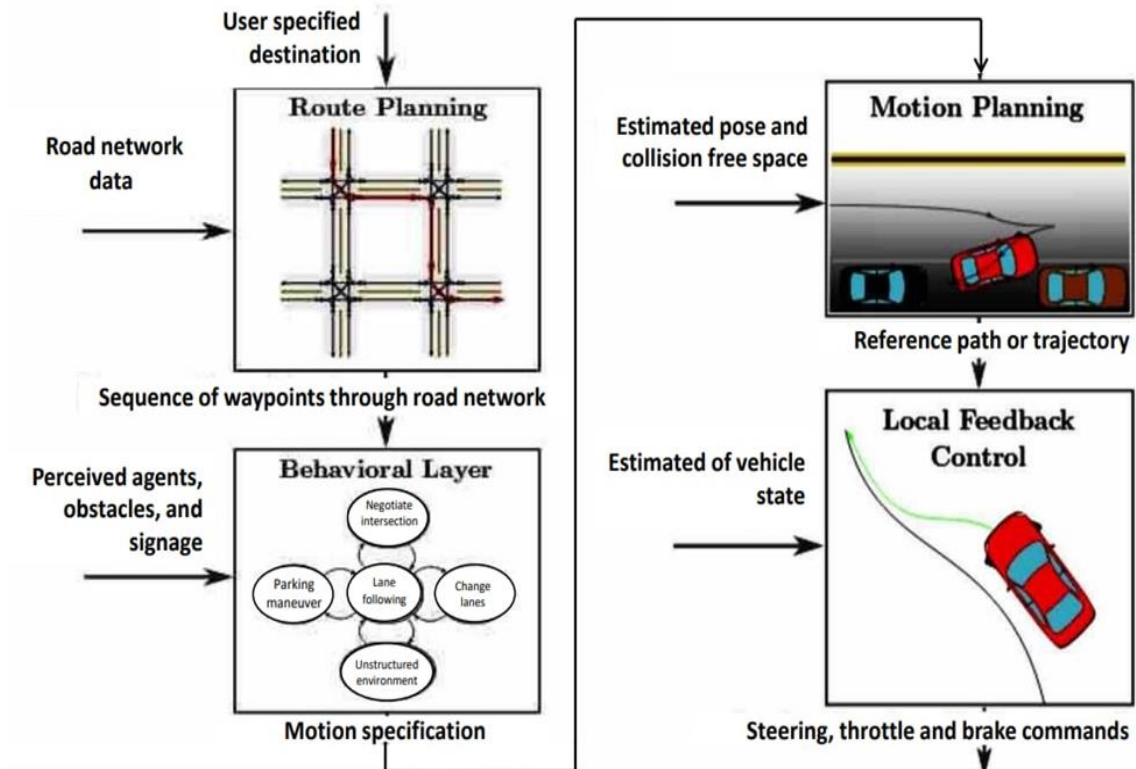


Fig. 3.1 An illustration of the decision-making process hierarchy[11]

3.2.1 Route Planning

A car's decision-making system must determine a path via the road network from its current location to the intended destination at the highest level. By expressing the road network as a directed graph with edge weights proportional to the cost of traversing a segment of the road, such a path may be defined as the problem of finding the path with the lowest cost on the road network graph [11].

3.2.2 Behavioral Decision Making

Following the discovery of the route plan, the autonomous car must be able to traverse the allocated path and communicate with other traffic participants while following to driving rules and road laws. Given a series of

road segments that define a given lane, the behavioural layer is responsible for choosing the appropriate driving behaviour (follow the lane, change lane, turn right, etc.) at any moment based on assessed behaviour of other traffic participants, road conditions, and infrastructure signals. One of the most recent developments in this field is an artificial intelligence approach to modelling this step in decision-making [11].

3.2.3 Motion Planning

The requested behavior must be translated into a path that the low-level feedback controller may trace when the behavioral class determines which command behavior to conduct in the present environment. The resultant trajectory should be dynamically possible for the car, passenger-friendly, and avoid accidents with impediments identified by on-board sensors. The motion planning system is in charge of locating such a path [11].

A great deal of navigation technology has been taken from mobile robots and modified to meet the challenges of road networks and driving rules. According to their application in automated driving, these planning techniques are categorized into four groups: graph search, sampling, interpolation, and numerical optimization. The motion planning layer is responsible for the dynamic computation of a safe, convenient, and viable path from the current car configuration to the target configuration provided by the behavioral layer of the decision-making hierarchy.

The motion diagram output is often forwarded to the local feedback control layer. The feedback controllers create an input signal to operate the car in accordance with the action plan. The purpose of motion planning for autonomous driving is to identify a suitable set of control inputs that will move a car from its beginning condition to its target state while remaining within

environmental and physical restrictions. Due to the high speed of autonomous driving compared to mobile robots, safety and driving comfort should be prioritized. To improve computing efficiency in dealing with changing traffic conditions in different road scenarios, the traffic planning issue for autonomous driving is frequently simplified to a global reference road planning level and a local traffic planning level [31].

3.2.4 Control System

In control system, the determination of the appropriate actuator inputs to execute the planned motion and correct tracking issues during the execution of the motion plan is accomplished by a feedback controller. Intelligent control is utilized in this task, whereby a control objective is synthesized and reasonable methods to achieve it are identified through a general information process that operates independently or in a human-machine mode. The motivation and knowledge used in this process include information about the environment and its internal state. Currently, the aggregation of a control objective is achieved through human-machine interaction in car control [32].

Intelligent control systems simulate biological intelligence to solve problems, and they seek to replace humans in performing tasks or borrow ideas from natural systems to solve control problems. For example, neural networks can be used for control. Intelligence and control are closely related, and the phrase "intelligent control systems" emphasizes the control component of an intelligent system [33].

To achieve their objectives, intelligent control systems must identify and employ targets, and control must direct the system towards those objectives. Any intelligent system is a control system because control is a key component of every intelligent system. However, intelligence is essential to

ensure that systems work as expected under changing situations, and a high degree of independence is required in the control system [34]

3.3 System Modelling

3.3.1 Path Planning

Path planning is an essential aspect of car detection. It is defined as establishing a geometrical path from the car's present position to a destination point while avoiding obstacles. It must be permissible to cross the car and ideal in at least one variable to be considered an acceptable path. For certain goal distance conditions, the shortest, smoothest, or fastest path that the car may follow can be the base path. In other words, the optimum path is determined using these factors. Path planning is commonly done by discretizing the space and using the centre of each unit as a moving point. Each movement location either has a barrier to avoid or is devoid of impediments that may be accessed. Various discretization processes produce various motion paths [35]. Creating an environmental map is required for path planning. The construction of an exact positional description of diverse items in the area in which the robot is positioned, such as road signs, obstacles, and so on, is known as environmental map construction: in other words, the creation of a model structure or map. The goal of creating an environmental map is to allow the robot to plan the most efficient path from the beginning point to the destination point in the model of the specific environment with obstacles. Path planning methods may be classified into two strategies based on the known level of environmental knowledge: path planning based on global map data and path planning based on local map data [36].

3.3.1.1 Global path planning

To compute an initial path, a global path planner requires the beginning and ending points of a constructed map, which is also known as a static map. The search is performed on the constructed global map model using a global map description of the area where the robot is placed. The best algorithm will find the best path. As a result, global route planning consists of two parts: “creating an environmental model and the path planning strategy.” Heuristic A* searching algorithm is commonly used for global path planning [37].

A. Graph search based planner

Graph search based planning is a method that involves using graph exploration techniques to find solutions to problems represented as graphs. The first step in this approach is to create a graph representation of the problem, which can be done in various ways depending on the specifics of the problem. Once the graph is created, search-based planning algorithms are used to traverse the graph and find a solution to the problem. In the context of autonomous driving, the state space represents the environment in which the car operates, and the goal is to navigate from one point to another. The state space can be represented as a grid or lattice, where each cell represents a possible location of the car. Graph search algorithms can be used to search through this state space and find a path that leads from the starting point to the destination. One advantage of graph search-based planning is that it can handle complex environments with obstacles and other obstacles that must be avoided. By representing the environment as a graph, the planner can easily check for obstacles and avoid them by finding an alternative path. However, the solutions found by graph search algorithms may not always be optimal, and it can be challenging to scale the approach to larger state spaces. Overall,

graph search-based planning is a powerful technique for finding solutions to path planning problems, and it has many potential applications in autonomous driving and other fields. These algorithms have been applied to develop automated tools [8].

1- Dijkstra algorithm

Dijkstra developed this systematic search technique in 1959 to discover the shortest path between two places on a map based on navigation costs. Priority queueing saves money on non-negative contract costs. Dijkstra's algorithm visits all nodes in the graph from the starting point and completes the solution if available. Without prior knowledge of the chart, it will not calculate the distance between each node and the destination under optimal conditions. It is used equation (3.1).

$$f(n) = g(n) \quad (3.1)$$

Where $g(n)$ is the real cost of travelling from node n to the beginning node.

Dijkstra's algorithm performs a blind search that takes time and wastes resources in processing. All nodes in the weighting scheme presented in this method are searched in ascending order based on their distance from the origin. The priority queue, which runs in a monotonic way, determines the nearest node from the starting point. events in discrete event simulation are prioritised by the times at which they occur and extracted monotonically. Prior knowledge of the target node is not required in Dijkstra, which makes it a naive algorithm. It can be implemented in a multi-node environment without a priori at the nearest node. It chooses the least expensive at every step and sometimes doesn't need to search all the edges. Due to its more generic, it is open to others, not just non-periodic charts. It usually searches a large area on a map and thus can be applied to geographical maps such as Google Maps.

The edges of the positive weight are kept in a priority queue and are referenced according to the distance between the positions in this algorithm.

In Fig.3.2, an example of Dijkstra's algorithm is shown, whereby the shortest path from node v1 to all other nodes in the graph is generated by checking the distance from node v1 to its neighbouring nodes, which are v2 and v3. From the list of distances, it can be immediately determined that node v3 has a distance of 4. Then the search is completed for all existing nodes to find the lowest cost for the path v1-v3-v4-v5-v6. Fig. 3.3 shows the flowchart of Dijkstra algorithm in path planning.

Dijkstra's Algorithm

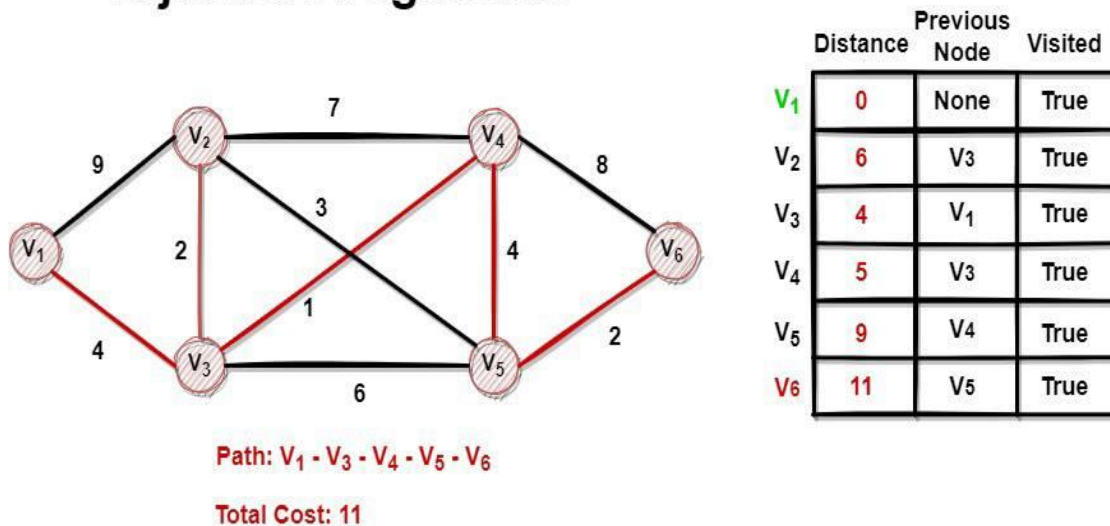


Fig. 3.2 Dijkstra example [38]

The Dijkstra technique is a well-known algorithm for determining the optimal path from the shortest route to find problems. However, with this method, the time required to find the optimal way is significantly longer when the search space is ample, so the Dijkstra method is unsuitable for real-time problems [39], [40].

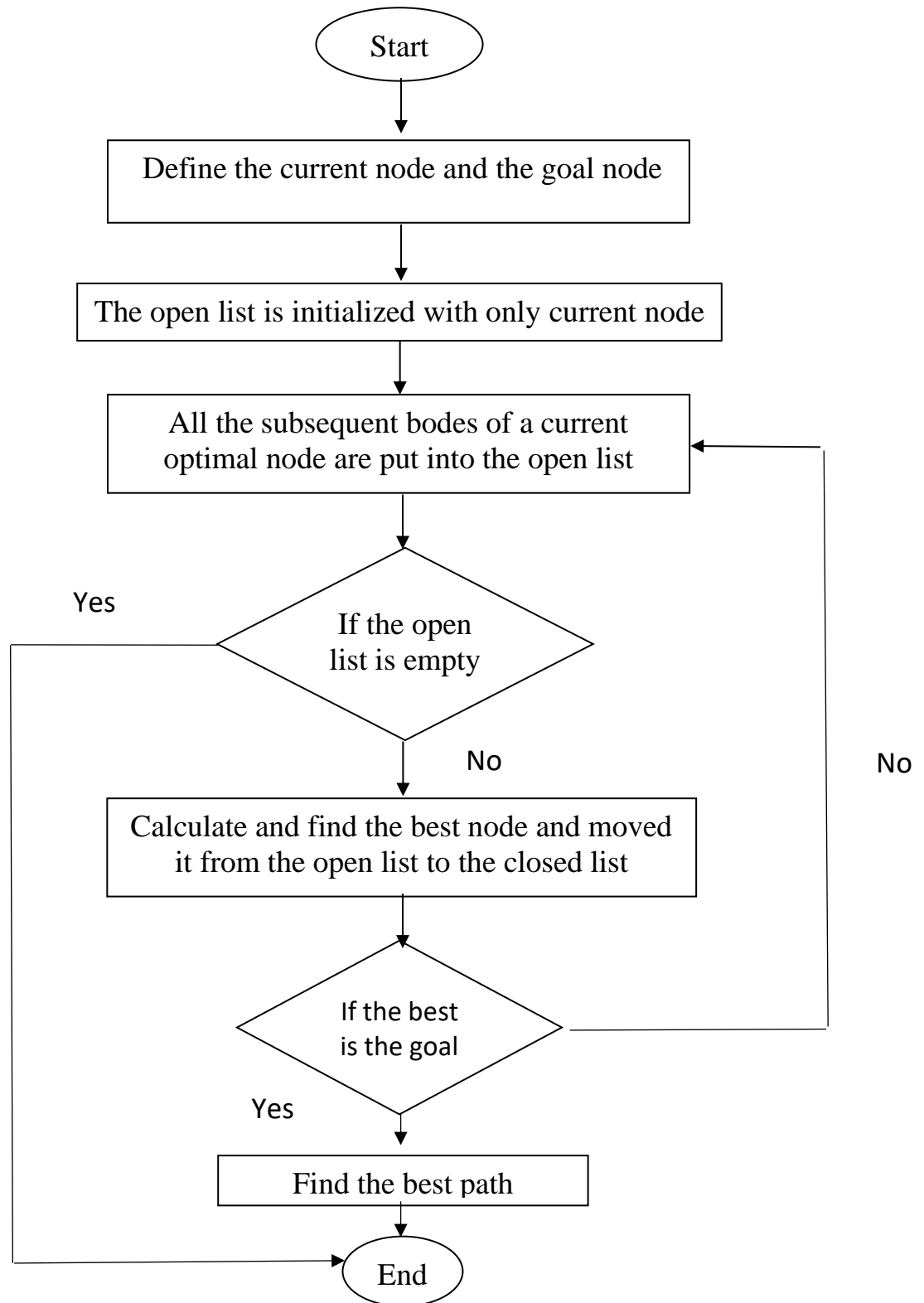


Fig. 3.3 Flowchart illustrates the Dijkstra algorithm, which is used in path planning

2- A* Algorithm

In 1968, Hart proposed the A* heuristic technique. It is a common graph path planning algorithm. A* works in the same way as Dijkstra's algorithm, except that it directs its search to the most promising situations, which can save a large amount of processing time. A* is mostly utilised to provide a nearly perfect solution with the current dataset/nodes. This approach is widely utilised in stationary environments and, in certain circumstances, in dynamic environments. The core functionality of a particular application or domain can be customised according to our needs. A*, like Dijkstra, follows a road tree from its beginning point to its goal. A* must decide which of its pathways to expand at each iteration of its main loop. It does so based on the path's cost and an estimate of the cost of extending the path all the way to the target. Specifically, A* uses the formula below to choose the path that minimises node search space (3.2).

$$f(n) = g(n) + h(n) \quad (3.2)$$

Where n is the next node on the path, $g(n)$ represents the actual expense cost from node n to the beginning node, and $h(n)$ represents the cost of the best path from n to the destination node. Fig. 3.4 shows an example of finding the short path in the A* algorithm.

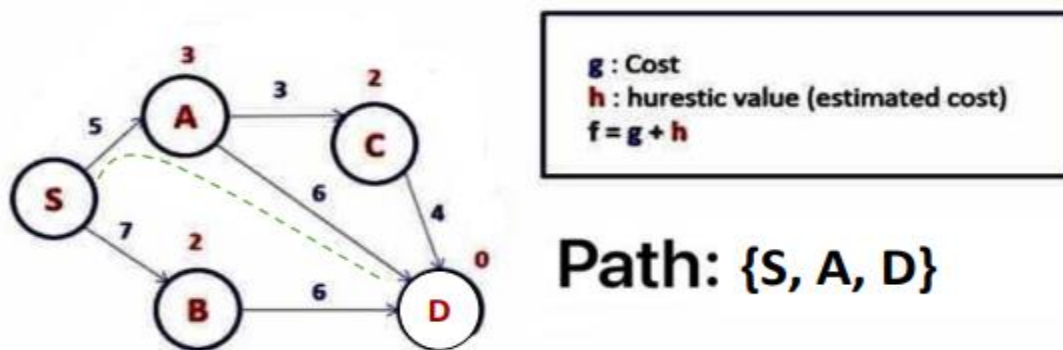


Fig. 3.4 A* example

In the game industry, the A* algorithm is commonly employed. The A* method has since been utilised for robot path planning, intelligent urban transportation, graph theory, and automated control as artificial intelligence has advanced.

The A* algorithm is a heuristic that use heuristics to choose the best path. The A* algorithm must locate nodes on the map and apply appropriate heuristics for guidance as shown in Fig. 3.5. Table 3-1 contains common heuristic functions such as Euclidean distance, Manhattan distance, and Octile distance.

Table 3-1 Most Common Types of Heuristic Functions Used in Path Planning Algorithms.

Function	Equation
"Euclidean distance"	$\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$
"Manhattan distance"	$ X_1 - X_2 + Y_1 - Y_2 $
"Octile distance"	$\text{Max}(X_1 - X_2 , Y_1 - Y_2)$

The A* algorithm is computationally simple compared with other algorithm ways of arranging calculations (ex; D*, state lattice). A* is suitable for car applications with car kinematics and steering angle [39]–[41].

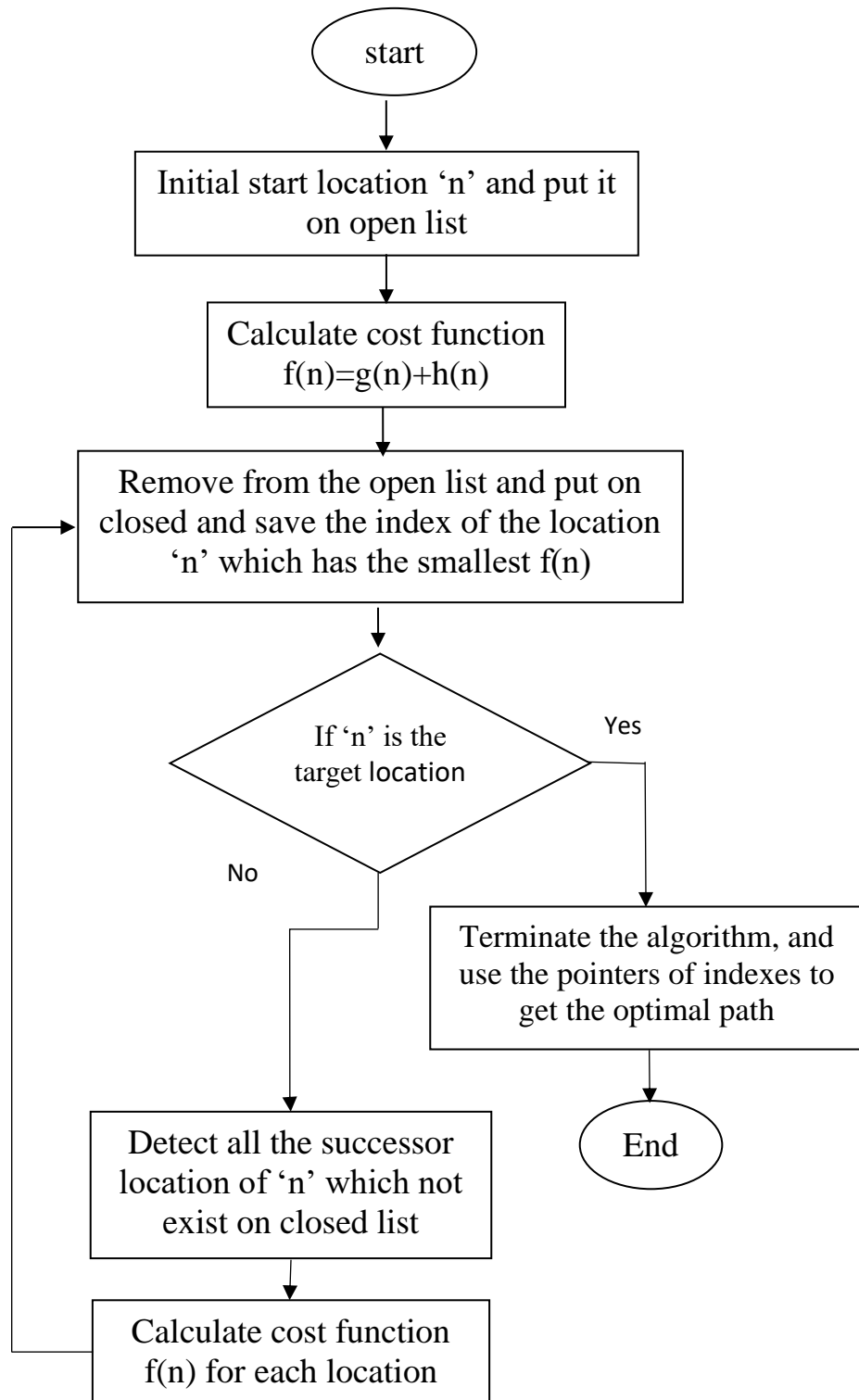


Fig. 3.5 Path planning flowchart using the A* algorithm

3- D* Algorithm

In (1994), Stentz Anthony proposed an informed incremental search algorithm D*, that is developed based on the A-Star and Dijkstra algorithms. It is designed to solve route planning problems in unknown environments by managing the condition of the robot and computing the case sequence using back indications to guide the robot to the target position or to update the cost due to obstacle detection. The algorithm places the appropriate states in the available list, and the states are processed until the cost on the path from the current state to the target is less than the minimum, at which point the cost changes are propagated to the next state and the robot continues to follow the indicators in the next sequence target [9], [42], [43].

The D* algorithm has two main functions: process-state and modify-cost. The process-state function is responsible for updating the state of the robot based on changes in the environment. This function takes the current state of the robot and the positions of any new obstacles that have been detected, and updates the paths in the inflation graph accordingly. The process-state function also places the updated states in the priority queue, so they can be expanded in the next iteration. The modify-cost function is responsible for updating the costs of the states in the inflation graph. This function is called when changes in the environment are detected, and it updates the costs of the states that are affected by the changes. The modify-cost function also propagates the cost changes to the neighboring states, to ensure that the paths in the inflation graph remain consistent.

Together, the process-state and modify-cost functions allow the D* algorithm to efficiently adapt to dynamic changes in the environment and find the shortest path to the goal. D* algorithm is typically used in robotics, where

the environment is constantly changing and the robot needs to adapt its path to the new conditions. It is also used in games and simulations where the environment is dynamic. However, it is important to note that D* has a higher computational cost than A* and it may not be suitable for real-time systems with limited computational resources [12].

4- State Lattices

Automated route planning frequently uses lattice-based graphs. Aircraft, cars, boats, and all-terrain cars, for example, all use the country's navigation networks. The state lattice method is an improved graph search algorithm such as A*, work with a large complex environment, is a discrete collection of all the system's reachable configurations. It is built by discretizing space into a hyperdimensional grid and attempting to connect the origin to every grid node through a feasible path, an edge. In general, the lattice is assumed to contain all feasible paths up to a given resolution, which means that if a car can travel from one node to another, the lattice contains a sequence of paths to perform this manoeuvre. As a result, it is concluded that this formulation is capable of resolving entire planning issues. Many reference trajectories are produced in this manner, and then the best one is selected based on the given cost function [45].

State lattices planner uses an A* lookup to get an agent from the start state to the target state. This example where the start point at (0,0, east, centre) and the goal point at (14,14, east, centre), aims to find a path between a car of two states given heading, wheel angle, and the presence of random obstacles. Initially, the agent has no knowledge of the state space except how it is structured, so the agent makes an initial plan to go directly to the target, using A*. This means that the agent sees a certain amount of the actual state space,

which initially, as far as the agent knows, is entirely free of any obstacles. As the agent moves along its initial path, A* updates its knowledge of the state space by "perceiving" the area around it. If the agent realises an obstacle is blocking its path, it will remap with A*. The agent has only "seen" a certain amount of the virtual state network at any point along the way, so it will plan according to what it knows. Moreover, while on the move, each position in grid mode is grouped by shape (X, Y, vertex, wheel angle). X and Y are two integers that form a coordinate location. The head chooses one of four options: north, south, east or west, and the wheel angle chooses one of three options: centre, left, or right. The probability distribution shows the probability of an obstacle in a given region of the network of states. Because of these additional settings, the agent is a more realistic representation of a real robot. In Fig. 3.6, the agent's visibility is 1 unit, and the probability of node blocking is 10%. The agent set up seven A* plans, incurred a route cost of 137, and expanded 6169 knots.

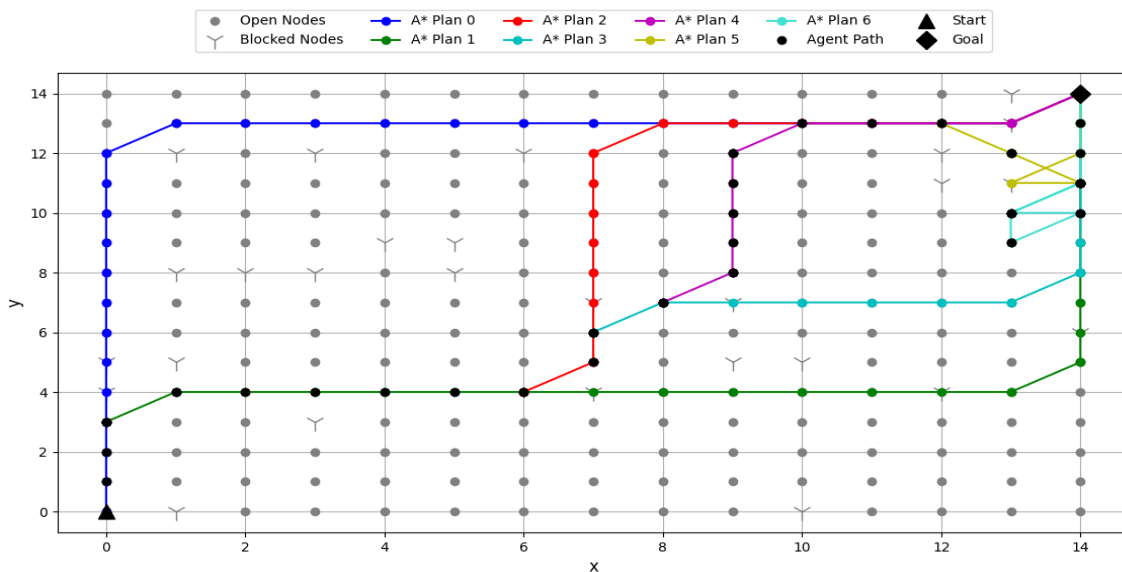


Fig. 3.6 State lattice planner when the agent's visibility is 1 unit

In Fig. 3.7, the agent's visibility is seven units, and the probability of node blocking is 10%. The agent set up three A* plans, incurred a route cost of 70, and expanded 3969 knots.

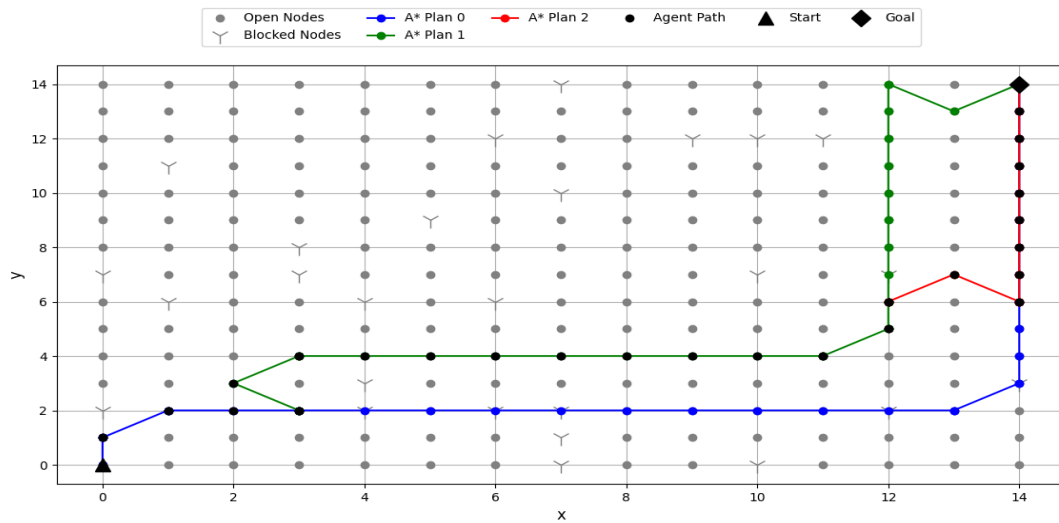


Fig. 3.7 State lattice planner when the agent's visibility is 7 unit

In Fig. 3.8, the agent's visibility is 15 unit, and the probability of node blocking is 10%. The agent set up two A* plans, incurred a route cost of 50, and expanded 2745knots.

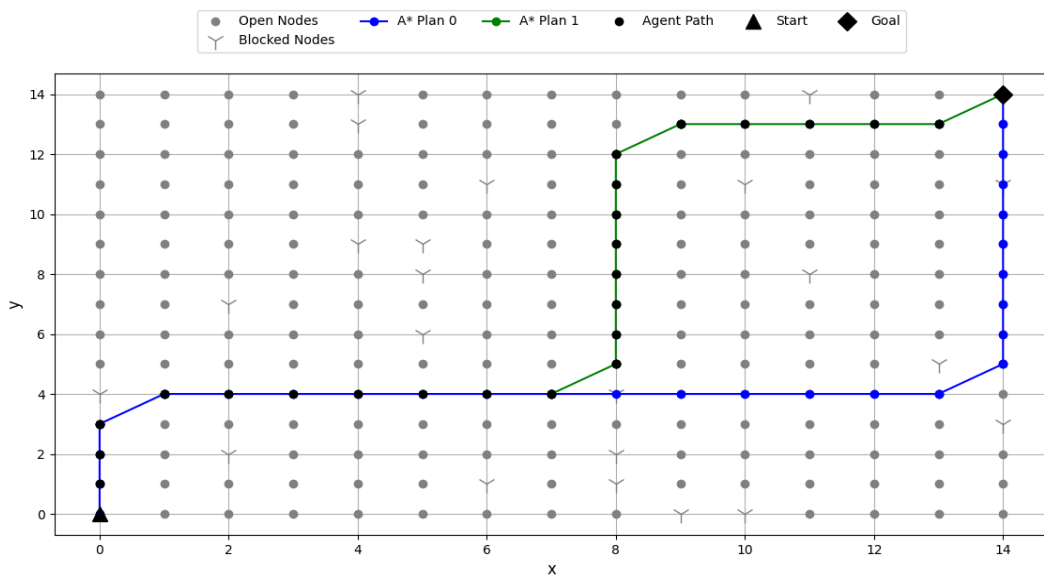


Fig. 3.8 State lattice planner when the agent's visibility is 15 unit

As the agent's visibility increases, the average number of A* plans have to minimise because the agent can get more information and apply more knowledge to each plan. Because of the randomisation of the case space, the comparisons are not straightforward, but it is natural to see that if the agent is less visibility, the cost will be higher, and the agent will likely have to make more plans A*.

3.3.1.2 Local path planning

Path planning that requires the robot to navigate in an uncertain or dynamic environment is known as local path planning. The algorithm will adapt to barriers and changes in the environment wherever it is used for path planning. Local route planning may be characterised as real-time obstacle avoidance employing sensory-based information on contingencies impacting the robot's safe navigation. Normally, a robot is driven with one path in local path planning. The shortest path from the starting position to the goal point is a straight line, which the robot follows until it encounters an obstruction. The robot then executes obstacle avoidance by deviating from the line while also updating certain key information, such as the updated distance from the present location to the goal point, the obstacle departure point, and so on. In order to reach the destination exactly, the robot must constantly know the position of the destination point from its present position in this type of path planning. The potential field approach is a well-known local path planning technique [46].

1- Potential Field Algorithm

The potential field function will be to predict a comprehensive path planning algorithm that takes the robot via vector quantities of the target's attractive force and repulsive forces from obstacles in the area. The aim is to

discover a direct path from the robot's starting point to the destination position while avoiding obstacles. The potential functions to be investigated are differentiable real value functions; hence, given that the potential function's value is energy, the gradient of this function will create the force. A field potential gradient is predicted to drive the robot to the goal position based on this simple but powerful assumption.

The job's success is dependent on the robot's possible attractive and repulsive gradients. The robot and the rest of the obstacles are believed to be positively charged, whereas the target is supposed to be negatively charged. This charge difference produces repulsive forces that push the robot and pull the target. The potential function is the sum of the potential attractive and repulsive on a robot.

$$\mathbf{U} = \mathbf{U}_{att} + \mathbf{U}_{rep} \quad (3.3)$$

where, U_{att} is the attractive potential and U_{rep} is the repulsive potential. Attraction tends to drag the robot towards the target position, and repulsion tends to push the robot away from obstacles. The gradient U yields a vector field for artificial forces $F(d)$.

$$\mathbf{F}(d) = -\nabla \mathbf{U}_{att} + \nabla \mathbf{U}_{rep} \quad (3.4)$$

$$\mathbf{F}(d) = -\mathbf{F}_{att} + \mathbf{F}_{rep} \quad (3.5)$$

Where, ∇U is the gradient vector of U at robot point $d(x, y)$.

The general form of suitable potential field functions suggested by Kathip follows.

(a) attractive potential field and force

$$\mathbf{U}_{att} = \frac{1}{2} \zeta * d^2 \quad (3.6)$$

$$\mathbf{F}_{att} = \nabla \mathbf{U}_{att} = \zeta * (d) \quad (3.7)$$

Where, U_{att} is the attractive potential field, F_{att} is an attractive force, ζ is the attractive potential coefficient, $d = |d_{vehicle} - d_{goal}|$, $d_{vehicle}$ is the car location in the Cartesian coordinate system (x, y) , d_{goal} is the goal location in the Cartesian coordinate system (x, y) . The attractive force is a linear function which decreases as the car nears the goal.

(b) repulsion potential field and force

$$U_{rep} = \begin{cases} \frac{1}{2}\eta * \left(\frac{1}{d} - \frac{1}{d_0}\right)^2 & \text{if } d \leq d_0 \\ 0, & \text{if } d > d_0 \end{cases} \quad (3.8)$$

$$F_{rep} = \nabla U_{rep} = \eta * e^{-|d-d_0|} \quad (3.9)$$

Where U_{rep} is the repulsive potential field, F_{rep} is an repulsive force, η is the repulsive potential coefficient, $d = |d_{vehicle} - d_{obstacle}|$, $d_{vehicle}$ is the car position at (x, y) , $d_{obstacle}$ is the obstacle position at (x, y) , and d_0 is the influence of distance. The repulsion capability ensures that the potential increases significantly as the car approaches the obstacle and has no effect when the car is further away [47].

3.3.2 Controller Design

3.3.2.1 model predictive control

"Model Predictive Control (MPC)" is a term that refers to a variety of control approaches used in single input single output (SISO) and multiple input multiple output (MIMO) systems. It was originally utilised in 1970 by Shell Oil and is currently employed in a variety of sectors. One of the most effective advanced control strategies necessitates the use of a process model to minimise the discrepancy between predicted and actual outputs. The intended result may be applied to both basic and complicated procedures. The basic principle behind MPC is to forecast the future behaviour of the managed

system over a fixed time horizon and compute an optimal control input that minimizes a present cost function while meeting the system limitations. More specifically, at each sampling instant, the control input is calculated by solving an optimal open-loop control problem with a finite horizon; the first part of the resulting optimal control trajectory is then applied to the system until the next sampling instant when the horizon is shifted and the procedure is repeated. MPC is particularly effective because it enables the explicit insertion of complex state and input limitations, as well as an acceptable performance criterion, into controller design [48].

1) MPC strategy

Fig. 3.9 explains the MPC method clearly. At the current time k , the future predicted outputs ($y(k+N)$ for $N=1$ to P) of the system are projected at each instant using the process model across a prediction horizon (P), knowing values up to instant k (past inputs and outputs) and future inputs ($u(k)$, $u(k+1)$, ..., $u(k+P)$). The plant's state is measured at each sampling instant, and only the first element of the future input is applied to the plant, as a new measurement of the state may be available at the next sampling instant. This method is repeated at the following sampling period with the addition of the new measurement, which is known as the receding horizon method.

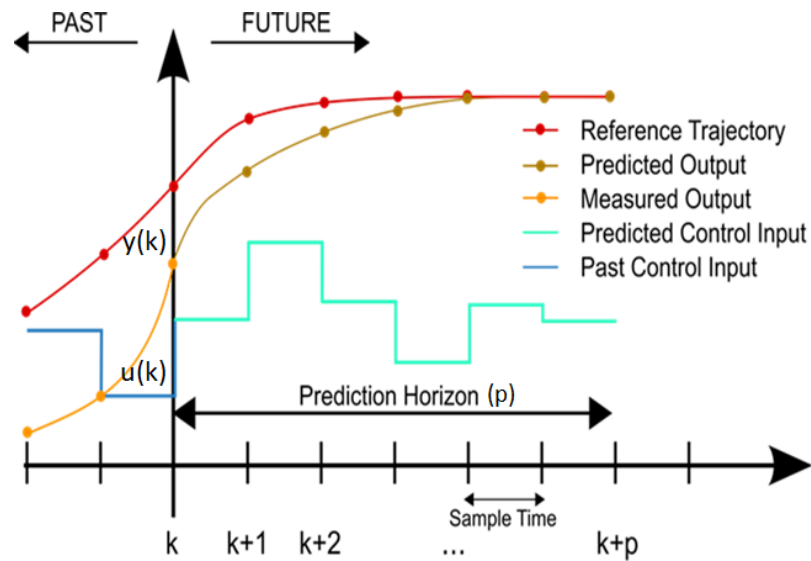


Fig. 3.9 MPC strategy[49]

The MPC approach predicts future car motion states by combining current sampling states and target states (reference trajectory) provided by the path planner. At each period, the MPC controller generates a control action sequence that satisfies the system constraints and optimizes the objective function. The MPC controller chooses the system output variable by minimizing a quadratic function of states and control inputs, which is the most common objective function.

MPC consists of the following components[50] :

- **Process Model:** Describes the dynamics of a process in which all inputs and outputs must be addressed. Models such as feedforward, feedback, and disturbance can be used.
- **Objective Function:** The sum of all terms having a control need, also known as the cost function. It can be both linear and nonlinear. The objective function tracks a reference trajectory for predicting the future output.

- Receding horizon method: Predicts how the process will behave within a given range that takes into account both the present and the future. The estimated output constraints at each time interval in the horizon depend on the data provided to the controller at time t .

The fundamental structure of MPC is shown in Fig. 3.10. Based on the system's previous inputs and outputs, a model predicts future outcomes. At each time step, the predicted output of the plant is compared to its reference path, and future errors of the plant are estimated. The optimizer determines the optimal future control sequence by taking into account the intended functionality and limitations. The plant receives only the first component of this optimal control sequence, and the same operation is repeated at the following sampling period [51].

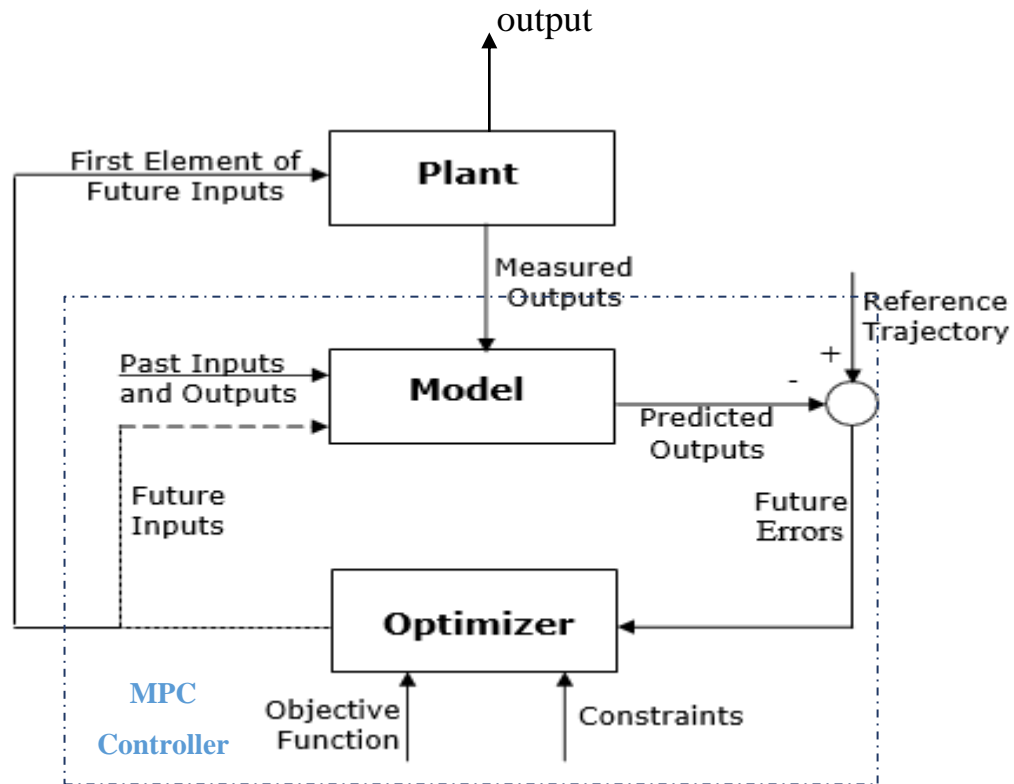


Fig. 3.10 Basic structure of MPC [51]

2) Basic parameter MPC [52]

The prediction horizon (P), control horizon (h), and sampling period (Δt) are the important parameters that affects the performance of the MPC system

the prediction horizon refers to the length of the future prediction made by the model. It is the time over which the MPC algorithm predicts future states of the system based on the current state and control inputs. A longer prediction horizon provides more accurate predictions of future states, but also requires more computational resources and longer computation times. A shorter prediction horizon provides faster response time but may not accurately capture the dynamics of the system.

The control horizon refers to the number of steps over which the control actions are applied. It is the time interval over which the control inputs calculated by the MPC algorithm are applied to the system. The control horizon is usually shorter than the prediction horizon to ensure a fast response time. The choice of control horizon is a trade-off between the accuracy of the control inputs and the computational resources required to calculate them. A longer control horizon provides more accurate control inputs but requires more computational resources and longer computation times. A shorter control horizon provides faster response time but may not accurately capture the dynamics of the system.

The sampling time refers to the time interval at which the control algorithm updates its predictions and control actions. It is the time interval between two consecutive measurements of the system's state. In general, a sampling time in the range of 10-100 milliseconds is commonly used in self-driving car simulations. This provides a desirable balance between computational efficiency and the accuracy of the simulation results.

3) MPC Formulation

Model Predictive Control (MPC) is a control strategy that uses a model of a system to predict its future behavior and optimizes control actions to achieve a desired objective. The optimization problem is solved at each time step, using the predicted state and control inputs to determine the optimal control action for the current time step and the predicted future. The MPC controller is designed in three stages: Firstly, an augmented state-space model is constructed. Second, the calculation of the vector of predicted outputs within the prediction horizon through the augmented model is performed. Finally, the control law is determined by solving an optimal control problem [53].

A) Augmented State Space Model

The control equation for the standard discrete-time state-space model is as follows:

$$\mathbf{x}(\mathbf{k} + \mathbf{1}) = \mathbf{A}(\mathbf{k})\mathbf{x}(\mathbf{k}) + \mathbf{B}(\mathbf{k})\mathbf{u}(\mathbf{k}) \quad (3.10)$$

$$\mathbf{y}(\mathbf{k} + \mathbf{1}) = \mathbf{C}(\mathbf{k})\mathbf{x}(\mathbf{k}) + \mathbf{D}(\mathbf{k})\mathbf{u}(\mathbf{k}) \quad (3.11)$$

Where $\mathbf{x}(\mathbf{k})$ is the vector of state variable, $\mathbf{y}(\mathbf{k})$ is the vector of controlled variables, $\mathbf{u}(\mathbf{k})$ is the vector of manipulated variables, $\mathbf{A}(\mathbf{k})$ is the system matrix linearized at a time \mathbf{k} , $\mathbf{B}(\mathbf{k})$ is the input matrix, $\mathbf{C}(\mathbf{k})$ is the output matrix, and $\mathbf{D}(\mathbf{k})$ is the feedthrough matrix. $\mathbf{x}(\mathbf{k})$ and $\mathbf{u}(\mathbf{k})$ are members of a convex set subject to a set of linear constraints[53].

B) Prediction of Output and State

Using the augmented model, the predicted output and state are calculated at time instances $\mathbf{k}+1, \mathbf{k}+2, \dots, \mathbf{k}+P$ based on the current state $\mathbf{x}(\mathbf{k})$ and the future incremental inputs $\Delta\mathbf{u}(\mathbf{k}), \Delta\mathbf{u}(\mathbf{k}+1), \Delta\mathbf{u}(\mathbf{k}+2), \dots, \Delta\mathbf{u}(\mathbf{k}+h-1)$. Here, P and h represent the prediction horizon and control horizon, respectively.

The predicted output and state at time instance $\mathbf{k}+j|\mathbf{k}$ (where $j=1,2,3,\dots,P$) are denoted by:

$\mathbf{y}(\mathbf{k}+j|\mathbf{k})$: predicted value of the output variable \mathbf{y} at time step $\mathbf{k}+j$, given the information available at time step \mathbf{k} .

$\mathbf{x}(\mathbf{k}+j|\mathbf{k})$: predicted value of the state variable \mathbf{x} at time step $\mathbf{k}+j$, given the information available at time step \mathbf{k} .

C) Optimization

Future points are predicted based on the model, making the system considered in open loop with the plant to drive towards the goal. A cost

function for the optimization problem is defined based on the state input, providing optimal inputs while satisfying the constraints.

The cost function is defined as:

$$\text{Cost function} = \mathbf{V}(x_p) + \sum_{k=0}^{P-1} \mathbf{L}(x_k, u_k) \quad (3.12)$$

Where $\mathbf{V}(x_p)$ is the positive definite terminal cost and $\mathbf{L}(x_k, u_k)$ is the positive definite cost function for the state and input variables. The states must satisfy the system dynamics (3.10).

The $\mathbf{V}(x_p)$ and $\mathbf{L}(x_k, u_k)$ can be formulated using positive definite matrices.

$$\mathbf{L}(x_k, u_k) = \mathbf{x}(k)^T \mathbf{Q} \mathbf{x}(k) + \mathbf{u}(k)^T \mathbf{R} \mathbf{u}(k) \quad (3.13)$$

$$\mathbf{V}(x_p) = \mathbf{x}(p)^T \mathbf{D} \mathbf{x}(p) \quad (3.14)$$

Where \mathbf{Q} and \mathbf{R} are positive definite matrices on state and control variables. \mathbf{D} is so chosen matrices to make the system drive towards the final goal. At each time step, this optimization problem should be solved, and the sequence of the first element $u^*(k)$ is applied to the system. The output $x(k+1)$ obtained by optimizing the cost function provides the state vector, and the input vector for the P points on the horizon [54].

3.3.2.2 Neural Network Approach

A neural network (NN) is a machine learning algorithm that is designed to mimic the structure and function of the human brain. These algorithms use machine learning to interpret sensory input, label or aggregate raw data, and detect numerical patterns in vectors that include various types of real-world data, such as images, audio, time series, and text. The primary objective of neural networks is to categorize raw data. They can be trained on labelled or unlabelled data to identify patterns and subsequently categorize new data, a process known as learning. Neural networks have the ability to adapt to

changing inputs autonomously, meaning that the output parameters do not need to be redefined every time the input changes in order to achieve optimal results [55].

1) Components and Architectures of NN [56] [57] :

A neural network is consisting of the following parts:

- **Neurons:** Neurons simulate the behaviour of organic neurons using mathematical operations. The neuron receives input data, calculates a weighted average, and then applies a nonlinear function, such as an activation function, to produce an output.
- **Connection and weight:** Connections connect neurons in one layer to neurons in another layer, with each connection having a weight value that represents the strength of the relationship between the two components. The goal of training a neural network is to minimize a loss function, which measures the difference between the network's predicted outputs and the actual outputs for a given set of inputs. Lowering the weight values is one way to achieve this goal.
- **Propagation function:** There are two types of propagation functions in a neural network: forward propagation and backpropagation. Forward propagation calculates the expected value, while backpropagation computes the gradients of the loss function with respect to the weights of the network, which is used to update the weights during training.
- **Learning rate:** To optimize the weights, neural networks are trained via gradient descent, which is an optimization algorithm used to minimize the loss function of a neural network by iteratively adjusting the weights in the direction of steepest descent.

The main types of neural network architecture include:

- Forward neural networks: Are the most common type of architecture, with the first layer serving as the input layer and the last serving as the output layer, and all of the layers in between are hidden.
- Recurrent neural networks: This network's design is a collection of neural networks in which the connections between nodes build a directed graph over time, specifying a transient dynamic behavior.
- Symmetrically Connected Neural Networks are similar to recurrent neural networks, but they differ in the connections between their modules. Unlike non-symmetrically connected neural networks, where the weights of connections between modules can differ in both directions, the connections in symmetrically connected neural networks have the same weight values in both directions.

2)Neural Networks in Control Systems:

Neural networks in control systems have been suggested by Werbos in 1989 and Narendra in 1990 [58]. The control of neural networks had two primary purposes: and

- Approximate dynamic programming using neural networks
- Neural networks in optimum control problem solving and closed-loop feedback control.

The challenge of using neural networks for feedback control purposes is to define an appropriate control system architecture and then show how to adjust neural network weights using mathematically proper techniques to ensure

stability and performance in a closed loop. A model predictive controller is a well-known model controller for a neural network.

There are two steps when using neural networks for control:

- Defining the system: Developing a neural network model for the facility on which we are based.
- Control design: A neural network factory model for designing (or training) a control unit.

3)Neural network-based Model Predictive Control (N-MPC):

The first step in predictive control of the model is to define the NN facility model (system description). The controller then uses the plant model to predict future performance.

System Description:

1. The first step in predictive control of the model is to train the NN to represent the forward dynamics of the plant.
2. The estimation error between the plant output y_p and the NN output y_m is used as the NN training signal.
3. The neural network plant model (NNPM) uses past inputs and past plant outputs to predict future values of plant output [58]. NNPM is a critical part of the N-MPC methodology. Fig. 3.11 depicts the structure of the NNPM, where the input signal is $u(t)$ represents the system input and $y_p(t)$ represents the plant output, in layer1 (hidden layer) the blocks labeled TDL represent tapped delay lines that store previous values of the input signal and $IW^{l,j}$ represents the weight matrix from the input j to layer i . The sum of the weighted inputs and the bias forms the input to the transfer function S . The job of the transfer function is to combine multiple inputs into one

output value so that the activation function can be applied. Layer 2 (output layer) takes input from preceding hidden layers and comes to a final prediction based on the model's learnings, $LW^{l,j}$ denotes the weight matrix from layer j to layer i . The sum of the weighted input and the bias of the output layer pass to the activation transfer function l to get the output $y_m(t)$. This layer is considered the most important, as it provides the final output of the neural network model plant [59].

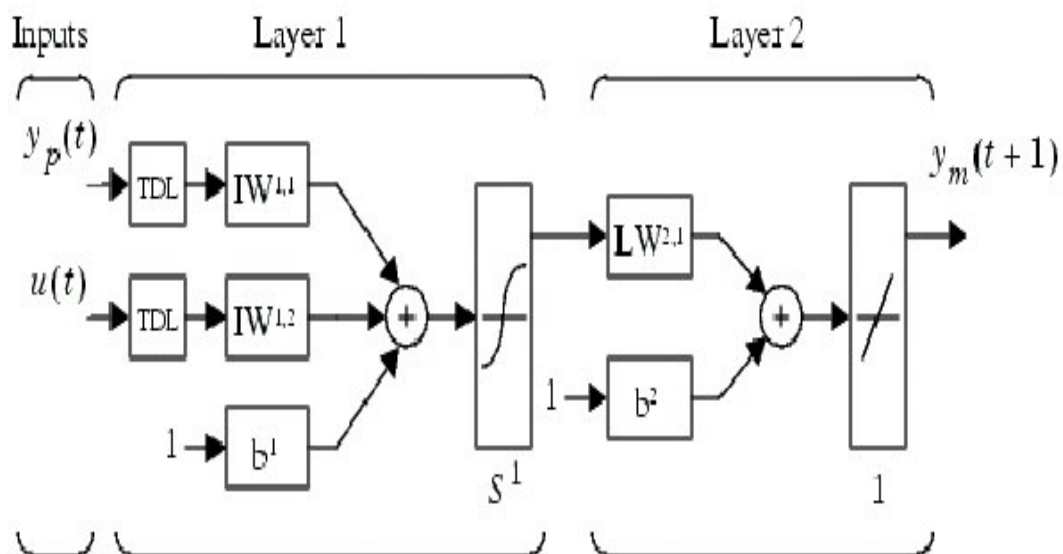
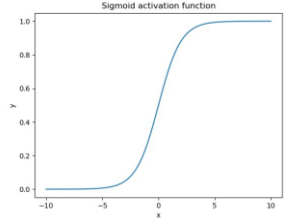
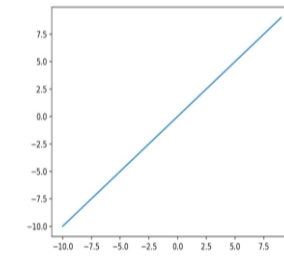
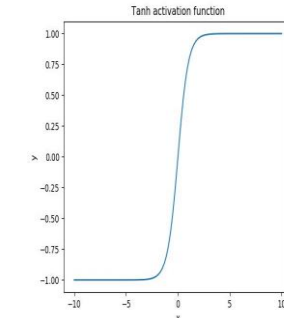
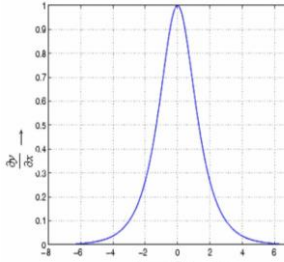


Fig. 3.11 The structure of the neural network plant model [59]

Typically, all hidden layers in a neural network use the same activation function. However, the activation function used in the output layer may differ from the hidden layers, depending on the type of prediction or goal of the model. Table 3-2 shows the different types of activation functions commonly used in neural networks, according to sources [60], [61].

Table 3-2 Mathematical equations for activation function

Activation Function	Description	Equation	Implementation
Sigmoid	Transforms any input to a value between 0 and 1.	$\sigma(x) = \frac{1}{1 + e^{-x}}$	
Linear	Output is equal to its input	$y_{\text{linear}} = x$	
Hyperbolic tangent activation (Tanh)	Takes any real value as input and outputs values between -1 and 1	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
Derivative of tanh	Used to find the maxima and minima of functions when the slope is zero.	$\tanh' = 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2$	

3.3.2.3 Mathematical Model

The kinematic model of a non-holonomic car can be used to produce successful autonomous driving on urban roadways under the following assumptions:

- 1) The car is considered to go in a straight line, and vertical, pitch, and spin motions are ignored.
- 2) Both wheels have zero slip angles.

In the context of a self-driving car, the MPC formulation can control the car's behavior based on its current state and desired objective. The state of the car, including position (x and y), orientation (ψ), and velocity (v), is approximated using a kinematic model in this formulation. The control inputs, namely the steering angle (δ), assuming only the front wheel is used for steering. and acceleration (a), are considered as well. The center of the car is supposed to be in the middle of the rear axle and in the case of the bike model, at the rear wheel.

The state (S) and input (u) of the system are defined as [x, y, ψ, v] and [a, δ], respectively.

The kinematic bicycle model is represented by the following set of equations in an inertial frame based on the axis system with SAE standards [62]

$$\dot{x} = v * \cos(\psi) \quad (3.15)$$

$$\dot{y} = v * \sin(\psi) \quad (3.16)$$

$$\dot{\psi} = \frac{v * \tan(\delta)}{L} \quad (3.17)$$

$$\dot{v} = a \quad (3.18)$$

where L is the wheelbase.

The kinematic model approximates the state by considering the car's motion as a function of its position and velocity, taking into account the steering angle and acceleration. The model is represented as:

$$\dot{m} = A' * S + B' * u \quad (3.19)$$

Or as a function of state and input:

$$\dot{m} = f(S, u) \quad (3.20)$$

Where A' is the Jacobian of the state and B' is the Jacobian of the control input.

$$A' = \begin{bmatrix} 0 & 0 & -v * \sin(\psi) & \cos(\psi) \\ 0 & 0 & v * \cos(\psi) & \sin(\psi) \\ 0 & 0 & 0 & \frac{\tan(\delta)}{L} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B' = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{v}{L * \cos^2(\delta)} \\ 1 & 0 \end{bmatrix}$$

The MPC algorithm predicts the future state of the car over a specified time horizon using the kinematic model and the current state of the car. The state at the next time step after converting this model into a discrete-time analysis by setting the sampling time dt is calculated as:

$$S(k + 1) = S(k) + \dot{m} * dt \quad (3.21)$$

Using expansion Taylor series up to the first degree around the reference point (\hat{m}) we get,

$$S(k + 1) = S(k) + (f(S, \hat{u}) + A'(S(k) - S) + B'(u(k) - \hat{u}))dt \quad (3.22)$$

$$S(k + 1) = (1 + dt A')S(k) + (dt B')u(k) + (f(S, \hat{u}) - A'S - B'\hat{u})dt \quad (3.23)$$

This can be simplified as [63]:

$$S(k + 1) = A * S(k) + B * u(k) + C \quad (3.24)$$

Where A, B matrix are known and C is the matrix represent any constant factors that affect the system's current state.

$$A = \begin{bmatrix} 1 & 0 & -v * \sin(\psi) dt & \cos(\psi) dt \\ 0 & 1 & v * \cos(\psi) dt & \sin(\psi) dt \\ 0 & 0 & 1 & \frac{\tan(\delta)}{L} dt \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{v}{L * \cos^2(\delta)} dt \\ dt & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} v * \sin(\psi) * \psi * dt \\ -v * \cos(\psi) * \psi * dt \\ \frac{v * \delta}{L * \cos^2(\delta)} dt \\ 0 \end{bmatrix}$$

The MPC algorithm optimizes the control inputs, namely the steering angle and acceleration, over the same time horizon to minimize the objective function while satisfying the constraints. The optimal control inputs are then applied to the car, updating its current state, and the process is repeated.

3.3.2.4 Convex Optimization

Convex Optimization is one of the most significant approaches in the world of mathematical programming, with several applications. It also has considerably larger applications outside of mathematics, including machine learning, data science, economics, medicine, and engineering.

Convexity is significant in convex optimizations. Convexity is defined as the continuity of the first derivative of a convex function. It assures that convex optimization problems are smooth and have well-defined derivatives, allowing gradient descent to be used. Convex functions include linear, quadratic, absolute value, logistic, and exponential functions, among others. Convex sets are the most significant in terms of convexity. A convex set comprises all points on or within its border, as well as all convex combinations of points in its interior. A convex set is a collection of all convex functions. Simply said, the convex function takes the shape of a hill. Finding the global maximum or minimum of a convex function is thus a convex optimization problem. Convex sets are frequently employed in convex optimization approaches because they may be modified using certain operations to maximize or minimize a convex function. An example of a convex set is a convex hull, is the smallest convex set that may include a given convex set. On every convex interval, a convex function takes the value between its lowest and maximum values. This indicates that this convex function has no local extremes (on the convex region). It also expresses that just one point in this collection, which is on the convex hull, is closest to the minimum as shown in Fig. 3.12 [64].

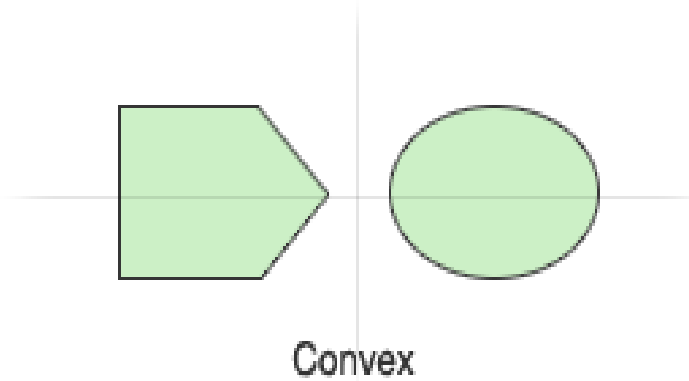


Fig. 3.12 Convex optimization (show that one point on the convex hull, is closest to the minimum)

Convex optimization issues are classified into two types:

- Constrained convex optimization: The convex function to optimize is constrained in some way.
- Unconstrained convex optimization: The convex function to optimize is not constrained in any way.

Chapter Four: The Proposed Work

4.1 Python Implementation

To control the autonomous car to move from the starting point to the end point, three different modules are used:

1-The map on which the car will operate, the starting and target location is selected and presented to the program.

2- In order to select the best algorithm for both global and local path planning, various path planning algorithms were tested. The results of these tests revealed the following:

- Dijkstra algorithm: One of its primary benefits is its low complexity, which is practically linear. It may be used to compute the shortest path between a single node and all other nodes, as well as the shortest path between a single source node and a single destination node, by ending the process once the shortest distance is reached for the destination node.
as well as their drawbacks it does an occluded investigation that takes a long time to process, it is unable to handle negative edges, it heads to the acyclic graph, so it cannot accomplish the exact shortest path, and there is also a requirement to keep track of vertices that have been visited.
- A* Algorithm: Is a heuristic search algorithm that uses an estimated cost to the goal to guide the search. It is known for its simplicity and its ability to guarantee the discovery of the optimal solution, i.e., the shortest path, between the starting point and the destination. A* is an excellent choice for global planning when the environment is relatively static, and the cost of each edge is known beforehand. In such a scenario, A* is typically faster than D* because it does not require incremental updates to the path.

Furthermore, A* is often more memory-efficient than D* because it does not need to store the entire graph in memory.

- D* algorithm: Is an incremental algorithm that updates the path as new information becomes available. It begins with an initial path and then iteratively improves it by considering the cost of edges and any modifications in the environment. D* is a great choice when there is a significant degree of uncertainty about the environment or when the environment is constantly changing. This is because D* can adapt quickly to changes in the environment by updating the path incrementally. Additionally, D* has the ability to handle dynamic environments where obstacles may move or appear suddenly.
- Potential field algorithm: A local path planning strategy used in real-time obstacle avoidance. It is an attractive approach because of its elegance and simplicity. However, the robot may quickly fall to a local minimum when using this strategy. As a result, there is a need to use it with another algorithm for global planning.

Based on the results of the above algorithms, the A* algorithm was selected for global path planning, and the Potential field algorithm was chosen for local path planning.

The A* algorithm was executed on the known environment to create a global path from start to target, in order to prevent the car from getting stuck in local minimums that may exist on the map if obstacles are not considered. The path provided by the A* algorithm was then defined at multiple equally spaced path points, which served as potential intermediate field targets. These waypoints guided the car across the map, with the lane point closest to the car's starting position generating an attractive starting potential field.

The potential field algorithm was executed at each time step to generate a path up to a few time steps into the future, enabling the car to navigate through obstacles in dynamic environment.

The path planning algorithms provided a short and reference trajectory for the car to follow, while constantly updating the path in response to moving obstacles.

3- Predictive controller typically works in a continuous loop to ensure that a car follows a specific path. The control unit achieves this by taking inputs such as the current state of the car, the reference trajectory of the car, and a predicted set of control inputs for future time steps up to the horizon.

The desired states of the car are calculated by the controller based on its current speed and the coordinates of specific points on the reference path in the future. The index of the closest point on the path to the current car position is determined and used to set the initial desired state. The desired state is then calculated up to the horizon by iterating through a loop. If the calculated point is within the total points of the path, the desired state is set accordingly. Otherwise, the final point of the path is used, and the desired states are returned along with the target point.

After the desired state is set, a set of future states for the same number of future time steps up to the horizon is computed by the controller. This prediction is based on a given set of control input values, namely the steering angle and acceleration, which are calculated by the controller to ensure that the car follows the desired path.

Using these predicted states, the controller approximates the kinematic model of the car by utilizing a mathematical model that describes the relationship between the car's motion and its control inputs. This model takes into account

various physical parameters of the car and uses them to predict how the car will behave in response to different control inputs.

To make this prediction, the controller first calculates the current state of the car, including its position, velocity, and heading angle. It then utilizes this information, along with the predicted control inputs for future time steps, to estimate the car's trajectory over that time horizon.

By comparing this predicted trajectory to the desired trajectory, the controller can determine whether the car is on track or needs to be adjusted. The kinematic model is crucial because it provides a way for the controller to understand how the car will respond to different control inputs and to adjust its control strategy accordingly.

Finally, the controller finds the cost of its actions at each point, considers a range of objectives, including minimizing the discrepancy between the desired and actual position and orientation of the car, reducing the control input necessary to manage the car, and minimizing the rate of change of the input. Each objective is assigned a weight based on its significance, and the total cost is obtained by adding up the weighted objectives over a defined time horizon with the terminal cost, as shown in equation (4.1):

$$C = \sum_{k=0}^{P-1} \left(Q \begin{bmatrix} (x_K - x_{Kdesired})^2 \\ (y_K - y_{Kdesired})^2 \\ (v_K - v_{Kdesired})^2 \\ (\psi_K - \psi_{Kdesired})^2 \end{bmatrix} + R1 \begin{bmatrix} (\alpha_K)^2 \\ (\delta_K)^2 \end{bmatrix} + R2 \begin{bmatrix} (\alpha_{K+1} - \alpha_K)^2 \\ (\delta_{K+1} - \delta_K)^2 \end{bmatrix} \right) + D \begin{bmatrix} (x_P - x_{Pdesired})^2 \\ (y_P - y_{Pdesired})^2 \\ (v_P - v_{Pdesired})^2 \\ (\psi_P - \psi_{Pdesired})^2 \end{bmatrix} \quad (4.1)$$

Where Q is state error weightage matrix, $R1$ is input weighted matrix, $R2$ is rate of input change weightage matrix and D is final state weighted matrix. The cost function at each time step is typically computed by summing the individual costs over all points in the prediction horizon P .

After calculating the total cost generated by a set of control inputs, the neural network and optimizer (convex optimization was used) collaborate to identify the appropriate inputs for the plant, specifically in adjusting the plant input of self-driving cars as shown in Fig 4.1.

At the neural network, the training input is the difference between the predicted steering angle and the last steering angle applied to the plant model at each step. The training output is the required rate of change between the steering angle value in each step.

At each step of the car's motion, the neural network receives the input, which is the difference between the predicted steering angle for the current time and the previous steering angle applied to the self-driving car model.

The neural network generates a prediction, which represents the weight value necessary to achieve the appropriate rate of change in the steering angle. The network weights are continuously updated until reaching a maximum number of epochs or a minimal error, using the squared error of the difference between predictions and train output at each time step until reaching the goal. The hidden layer of the neural network utilizes the tanh activation, while the output layer uses the derivative tanh activation.

The neural network output represents the required weight, which when multiplied by the predicted steering angle, can control the change within an acceptable range for each step see Fig. 4.2.

The convex optimizer attempts to optimize the current conditions and control the inputs to minimize the cost function, while the following considerations are taken into consideration,

- $S(k+1) = A * S(k) + B * u(k) + C$ (car model)
- Maximum speed = 15 m/s
- Maximum reverse speed = 5 m/s
- Maximum steering angle = 45°
- Maximum steering rate = 30°
- Maximum deceleration = 6 m/s^2
- Maximum acceleration = 2.5 m/s^2

and then the optimal control input is input to the plant. The control inputs found using this method are given to the car for a one-time step. Then the whole process is repeated for the next step with its new state values. The system flowchart is represented in Fig. 4.3.

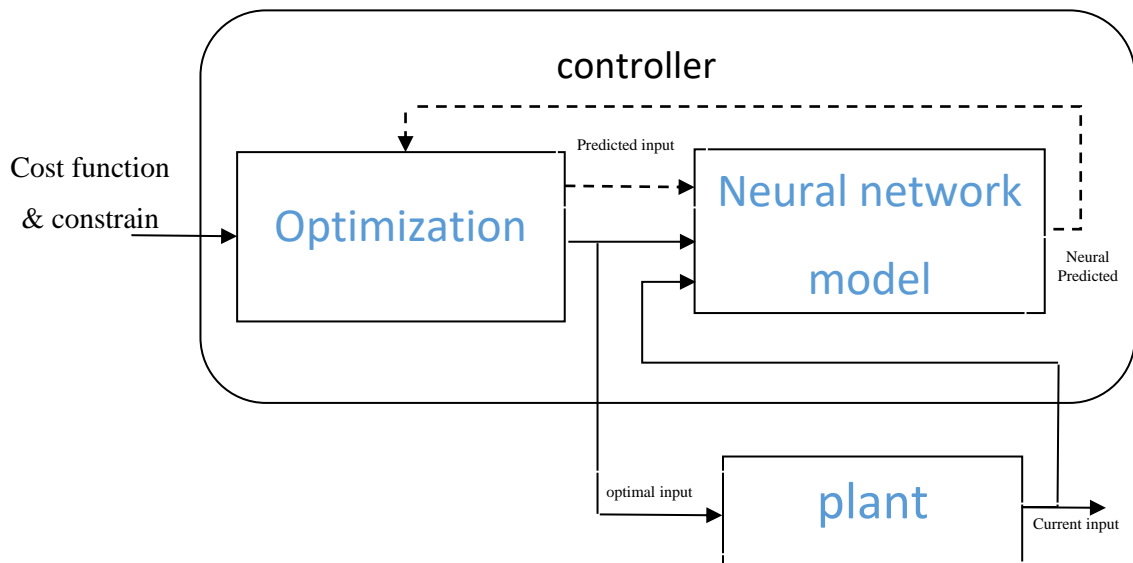


Fig. 4.1 Diagram of N-MPC

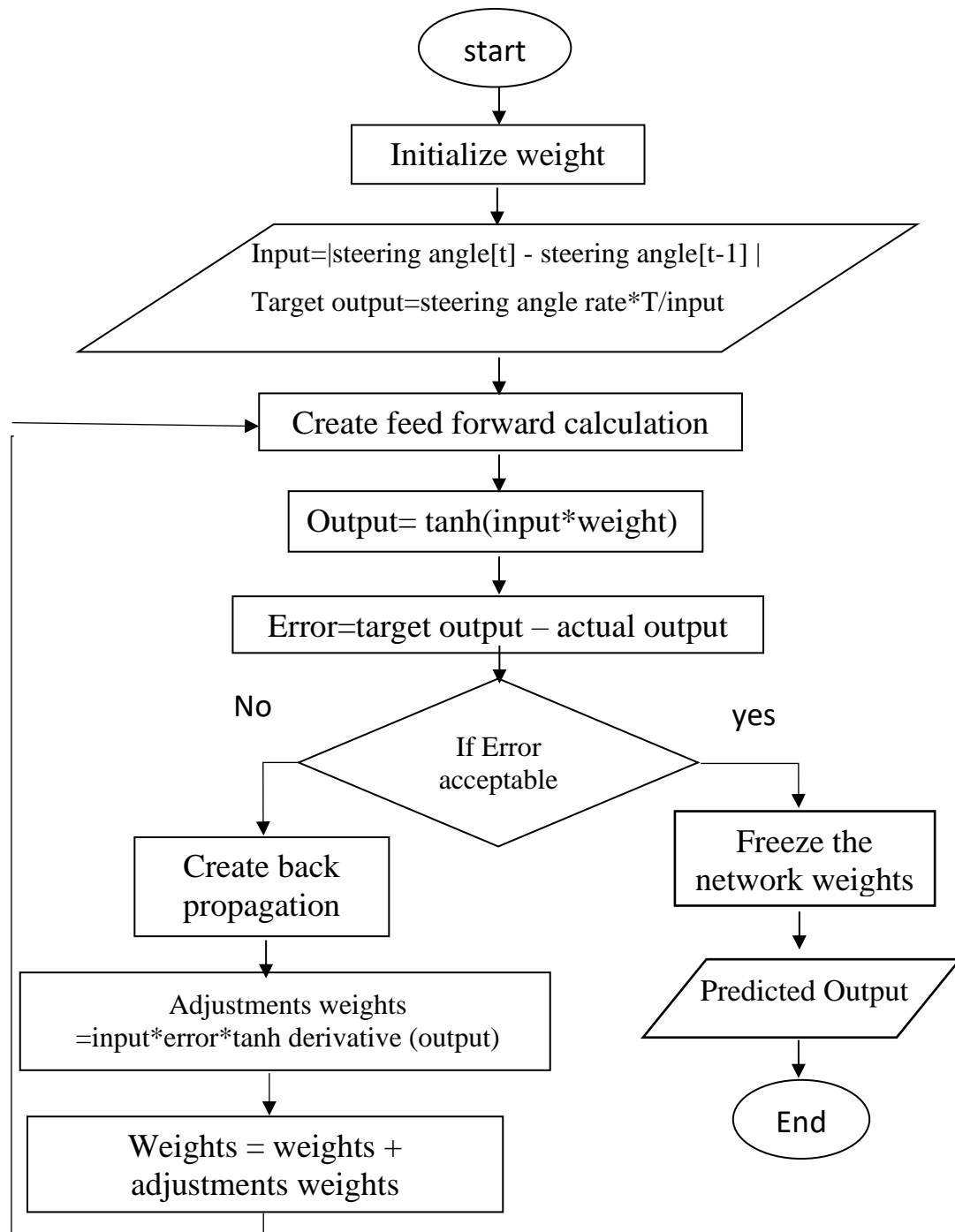


Fig. 4.2 Neural network flowchart

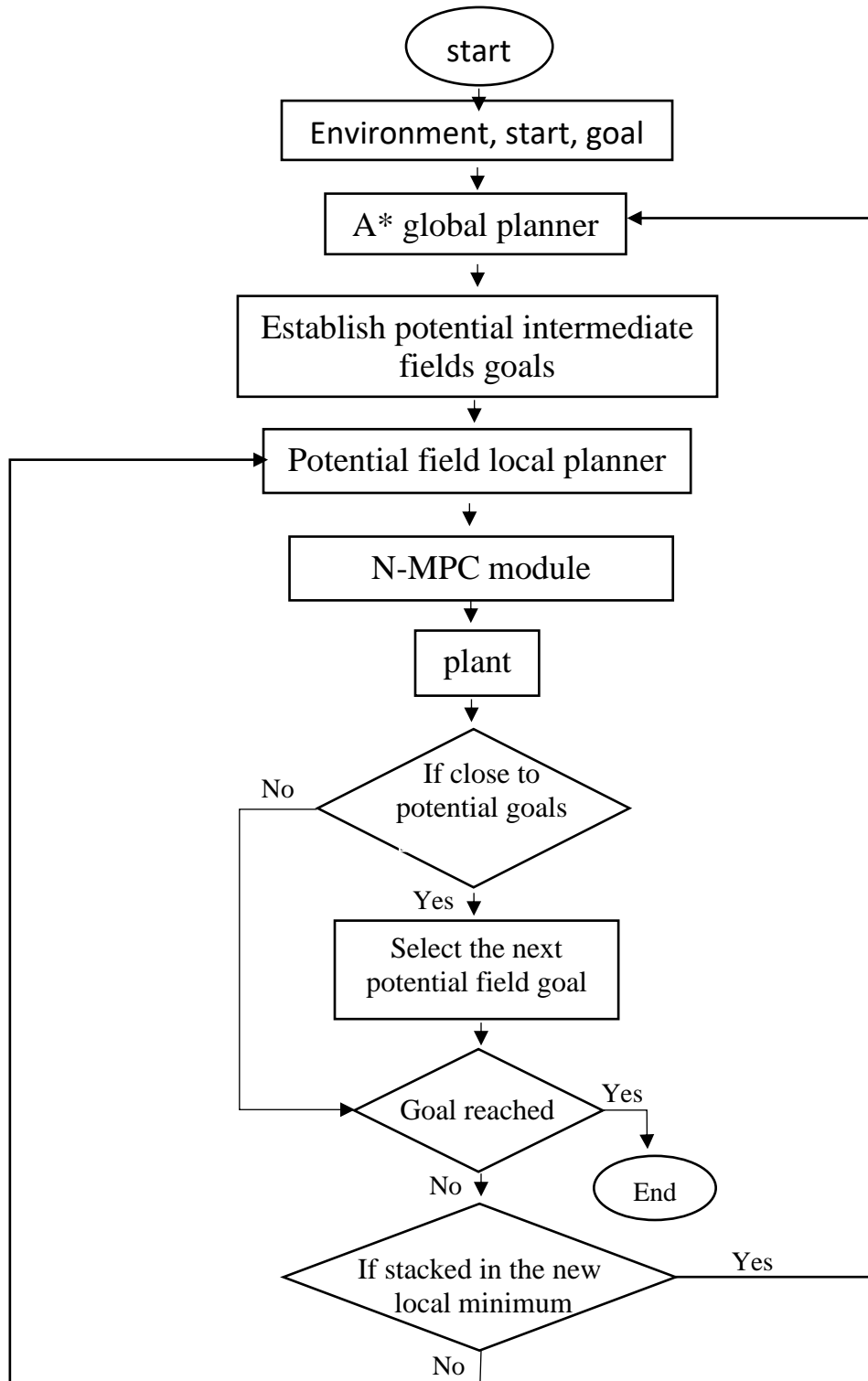


Fig. 4.3 System implementation flowchart

Chapter Five: Results and Discussion

5.1 Overview

This chapter presents the results obtained. An intelligent controller is used to control the self-driving car and supports motion planning in Python. Dijkstra, A*, D* and potential field algorithms results are discussed. Appropriate performance is determined by clarifying parameters that affect system performance, such as weights in the cost matrix, horizon distance, cost equations and constraints. The controller's performance has been tested for multiple paths to adjust the parameters to ensure static and dynamic obstacle avoidance in a constrained environment.

5.2 Motion Planning Algorithm Results

5.2.1 Dijkstra Algorithm

The algorithm was tested with three different maps, as shown in the Fig. 5.1, Fig. 5.2, and Fig. 5.3.

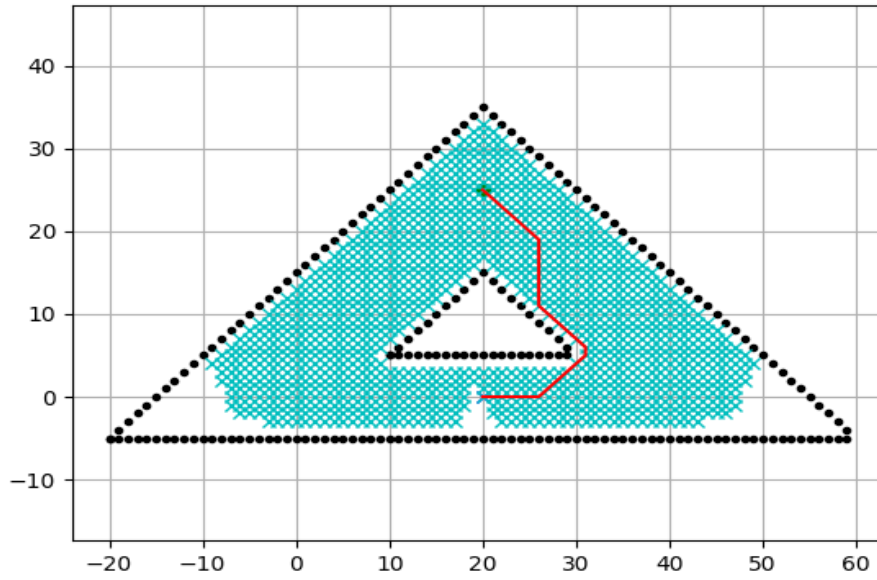


Fig. 5.1 Dijkstra algorithm path planning from (20,25) to (20,0)

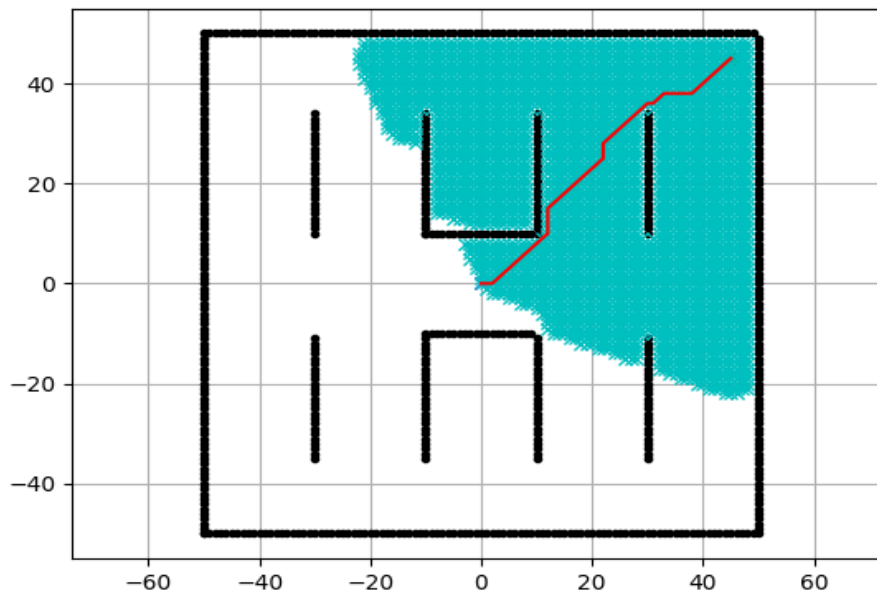


Fig. 5.2 Dijkstra algorithm path planning from (45,45) to (0,0)

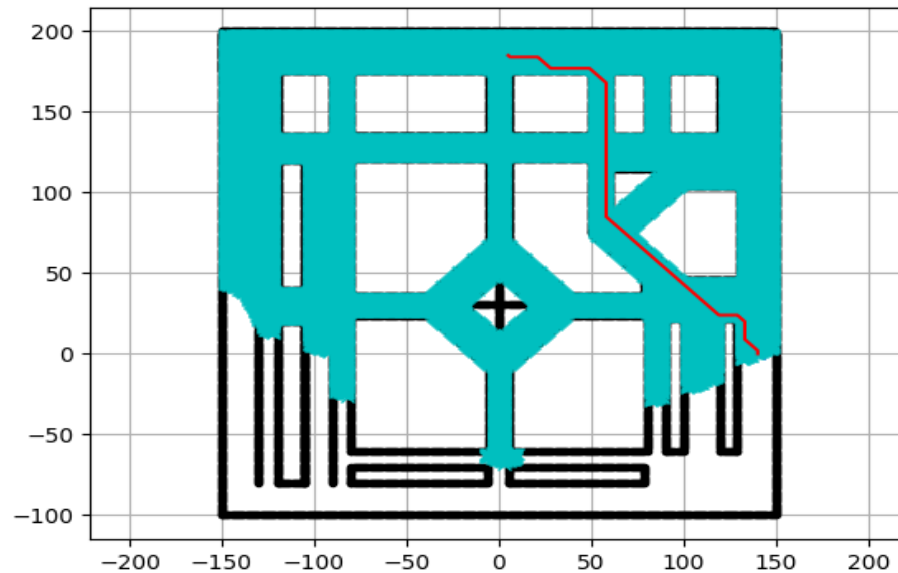


Fig. 5.3 Dijkstra algorithm path planning from (5,185) to (140,0)

The algorithm's main disadvantage is that it does a blind search, wasting a large amount of time, the blue color in the above figure represents the search area. Another problem is that it is incapable of dealing with negative edges. This results in acyclic graphs and, in most cases, failure to find the shortest path.

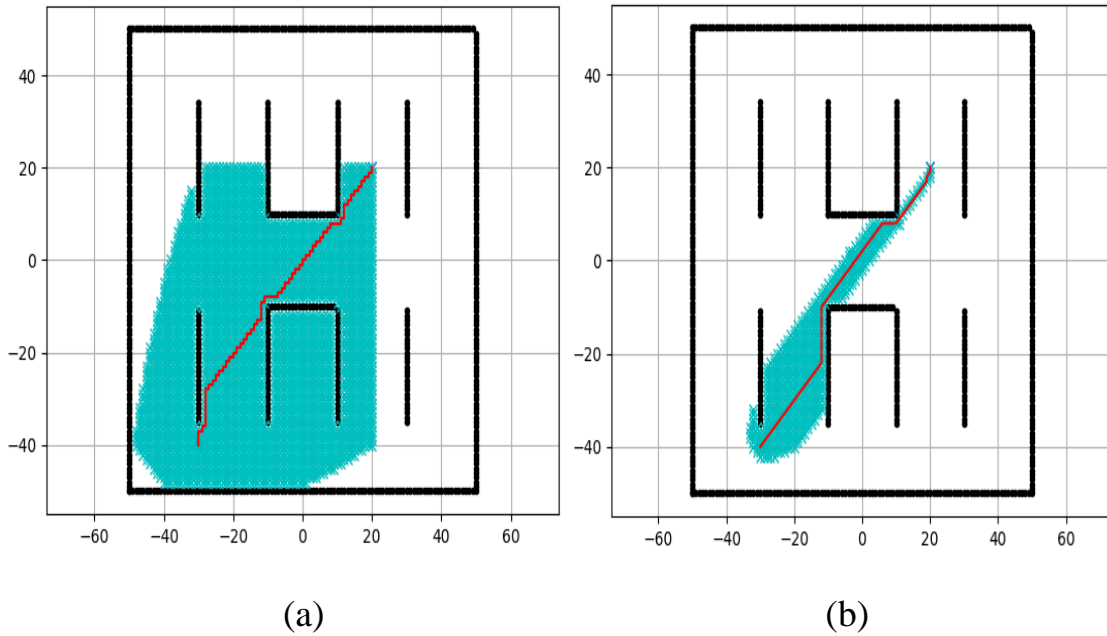
5.2.2 A* Algorithm

An efficient method that guarantees the shortest path solution for a relatively minor number of nodes is provided by A*. The cost that constitutes the cost of getting to a node from the starting position, $g(n)$, and the estimated cost of the node to the target, $h(n)$, are defined. These two are combined to get $f(n)$, which contains the information of both costs. A low $f(n)$ cost is adequate since it indicates that the nodes selected are closer to the goal, whereas a bigger value indicates that the nodes are moving away from the target. The nodes with the least $f(n)$ in the algorithm are chosen. Choosing the number of a successor depends on generating the next node. In this work, it was

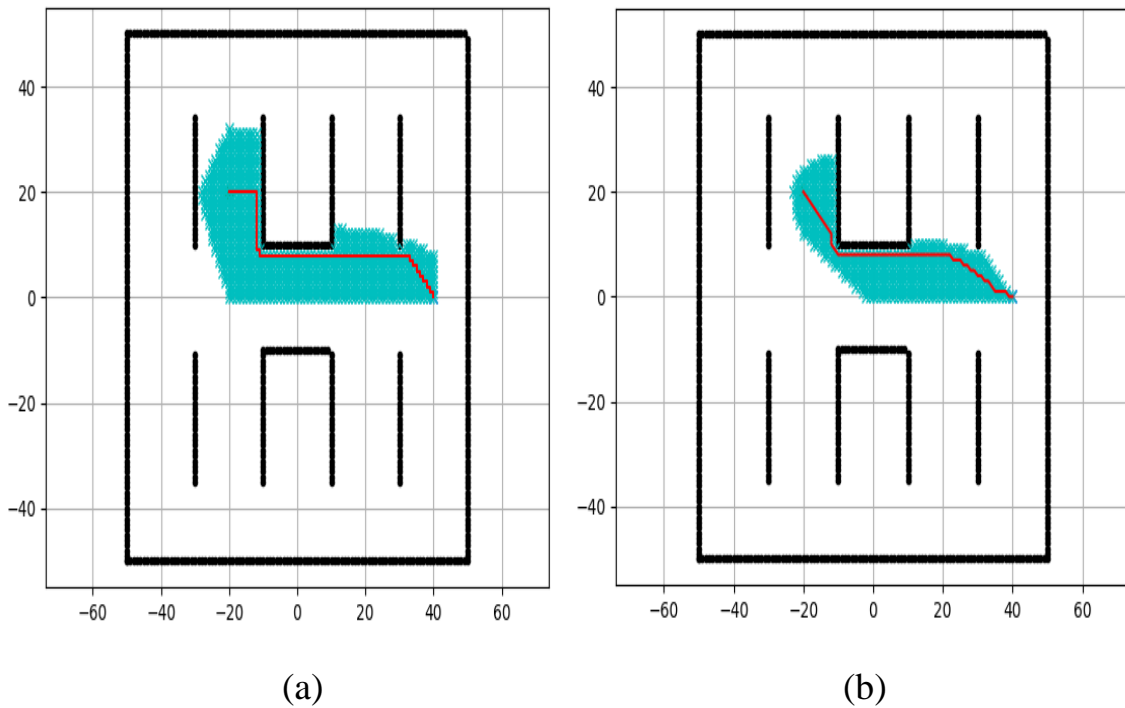
compared if the number was 4 or 8 to evaluate the trade-offs between the efficiency of the search and the completeness of the search.

- When the 4 successors of the node (North, West, East, South) are generated, it is observed that the path is zigzag and the search area is large, as shown in Fig. 5.4(a), Fig. 5.5(a), and Fig. 5.6(a). This leads to wasting time and energy consumption. The reason for this is that the algorithm is only exploring four potential paths at each node, which can result in a suboptimal solution if there are other better paths that are not being considered. Additionally, the algorithm may need to backtrack frequently to explore other paths, increasing the search area and wasting time.
- When all 8 successors of the node (North, North West, North East, South West, South East, West, South, East) are generated, the path becomes smoother and the search area becomes smaller than when generating only 4 successors. This is shown in Fig. 5.4(b), Fig. 5.5(b), and Fig. 5.6(b), which indicates that the access time has become less and energy savings are achieved.

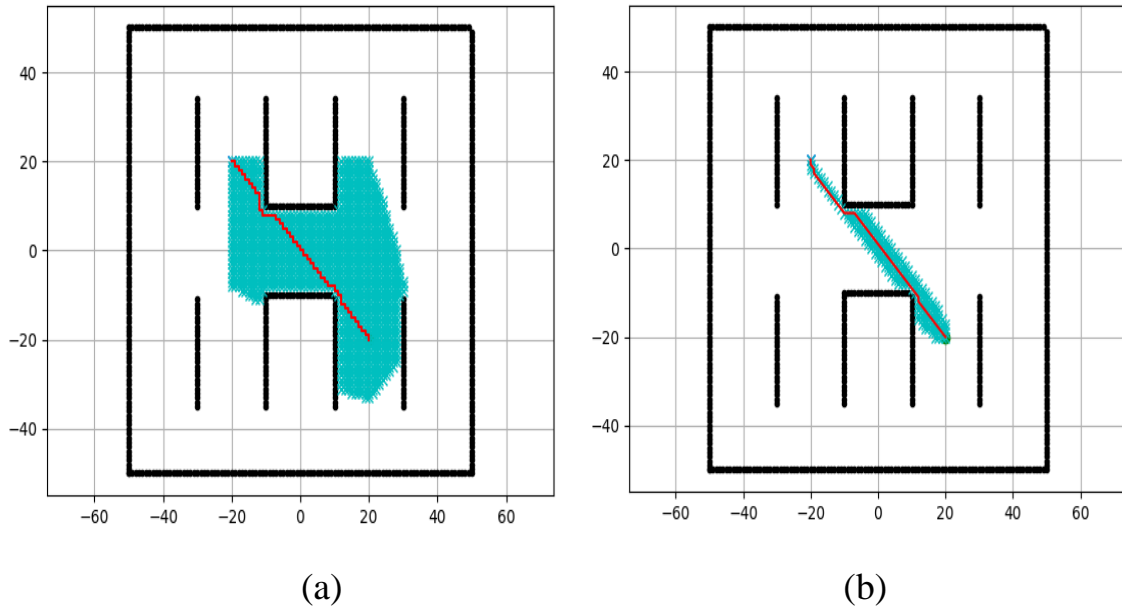
The additional successors (North West, North East, South West, South East) allow the algorithm to explore diagonal paths, which can be more efficient than moving only in the cardinal directions (North, West, East, South). This is because diagonals paths can cover more distance in fewer steps, which reduces the amount of searching and backtracking the algorithm has to do. Thus, generating eight successors rather than four can enhance the algorithm's performance. After that, the algorithm was tested in three different environments to ensure that the algorithm works correctly, shown in Fig. 5.7, Fig. 5.8, and Fig. 5.9.



**Fig. 5.4 A* algorithm path planning from (-30,-40) to (20,20),
(a) 4 successors, (b) 8 successors**



**Fig. 5.5 A* algorithm path planning from (40,0) to (-20,20),
(a) 4 successors, (b) 8 successors**



**Fig. 5.6 A* algorithm path planning from (20,-20) to (-20,20),
(a) 4 successors, (b) 8 successors**

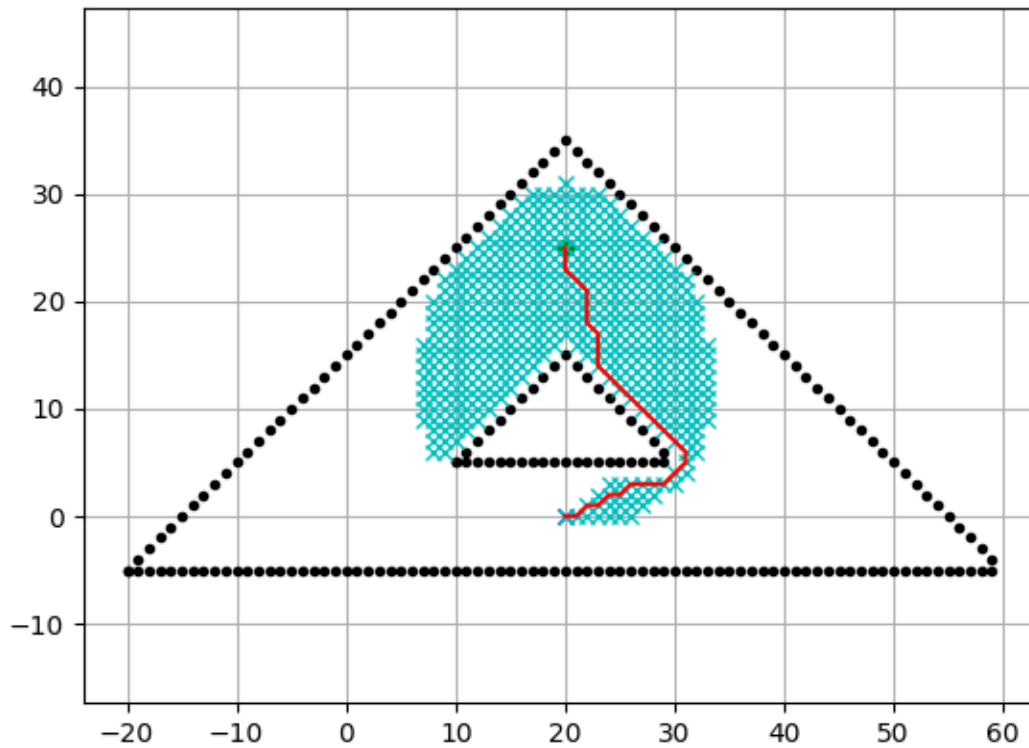


Fig. 5.7 A* algorithm path planning from (20,25) to (20,0)

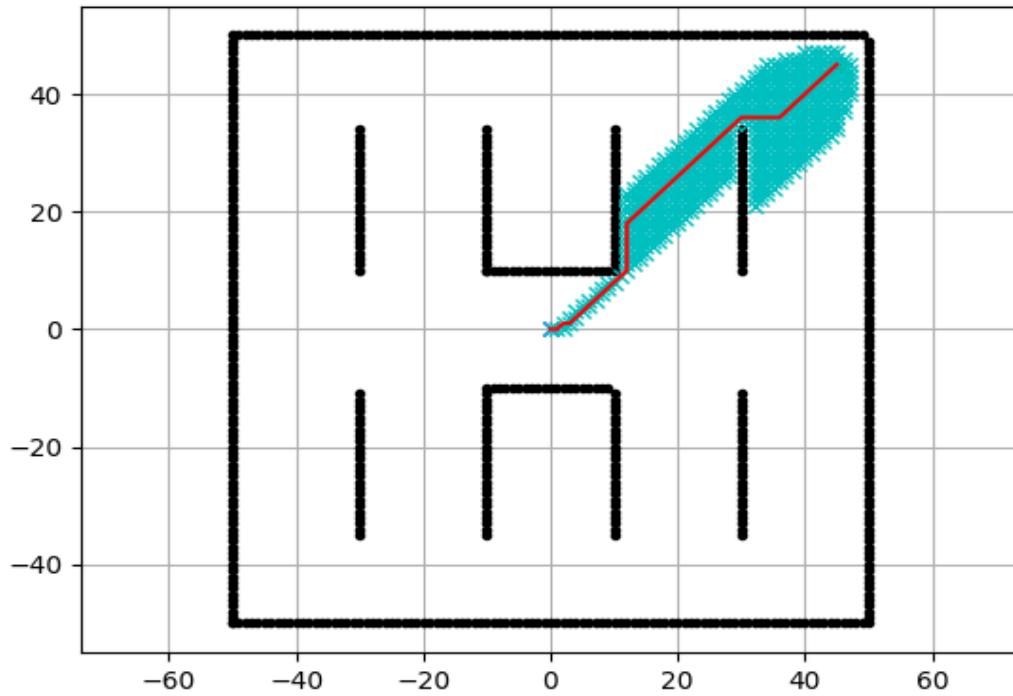


Fig. 5.8 A* algorithm path planning from (45,45) to (0,0)

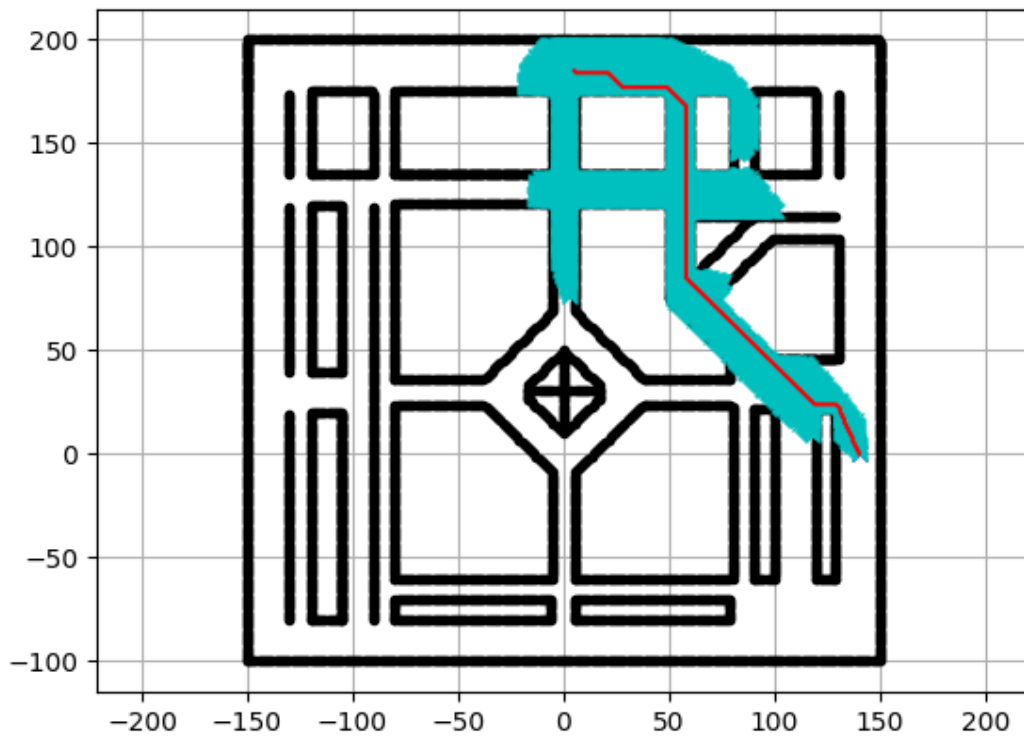


Fig. 5.9 A* algorithm path planning from (5,185) to (140,0)

5.2.3 D* Algorithm

The algorithm was tested with three different maps, as shown in Fig. 5.10, Fig. 5.11, and Fig. 5.12.

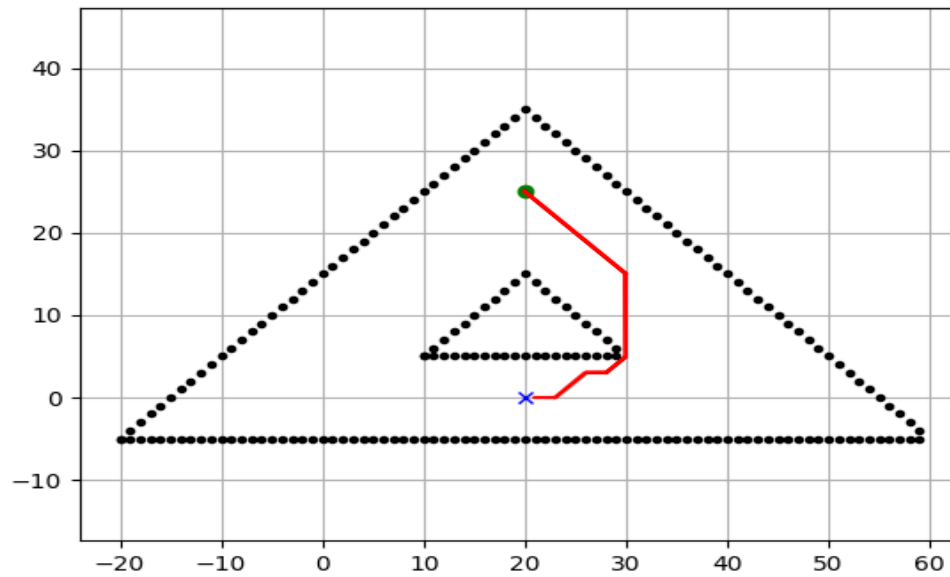


Fig. 5.10 D* algorithm path planning from (20,25) to (20,0)

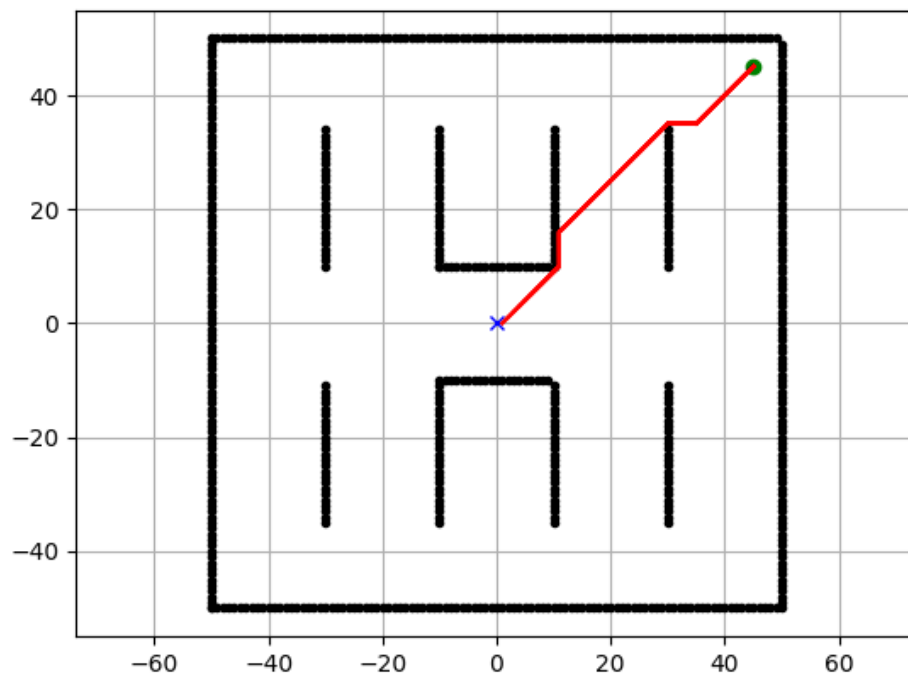


Fig. 5.11 D* algorithm path planning from (45,45) to (0,0)

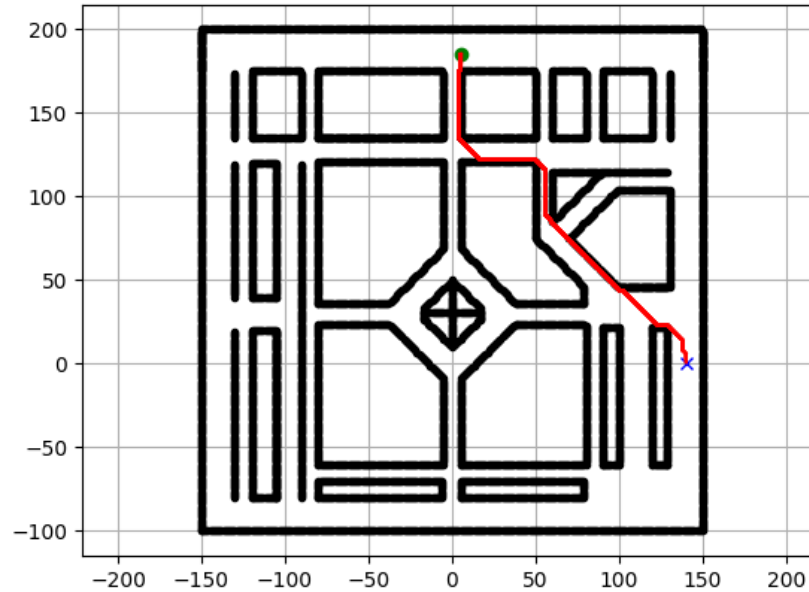


Fig. 5.12 D* algorithm path planning from (5,185) to (140,0)

The D* algorithm is better suited for dynamic environments because it can update the path as changes occur, which can be more efficient than re-planning the path from scratch. This is due to its use of incremental search and ability to reuse information from previous searches, enabling it to quickly update the path. While the D* algorithm requires more complexity in its implementation and uses more computational power than the A* algorithm, it can be highly beneficial in scenarios where changes to the environment occur frequently or in real-time.

5.2.4 Potential Field Algorithm

The potential field theory combines attractive and repulsive forces generated by the environment to guide the robot towards a target while avoiding collisions with obstacles. Each obstacle creates a repulsive force that is proportional to the distance between the robot and the obstacle. The target point generates an attractive force that pulls the robot towards it. However, the potential field theory may fail to find a solution in an environment with local minimums. The problem of local minima shown in Fig 5.13 can be defined as the reactive problem for an agent, attracted to a goal at position G. In general, a local minimum may form due to a superposition of the potential target and obstacles, causing the factor to fall into a state other than target G. To address this problem, the potential function in locales is used along with the A* algorithm. The values of ζ and η in equation (3.6), (3.8) affect the choice of the algorithm path, so the values must be adjusted according to the required performance.

To observe how the system is affected by different ζ and η values, four instances were taken:

- $\zeta = 1, \eta = 1$: The force of attraction with the target is low, and the force of repulsion with obstacles is low, creating an unwanted path, as shown in Fig. 5.14.
- $\zeta = 1, \eta = 100$: The force of attraction with the target is low, and the force of repulsion with obstacles is high, which generates a safe path far from obstacles and free of collision, as shown in Fig. 5.15.

- $\zeta = 100, \eta = 1$: The force of attraction with the target is high, and the force of repulsion with obstacles is low. Generates an undesirable direct path to the goal and does not avoid obstacles, as shown in Fig. 5.16.
- $\zeta = 100, \eta = 100$: The force of attraction with the target is high, and the force of repulsion with obstacles is high. Follow the desired path while avoiding obstacles, as shown in Fig. 5.17.

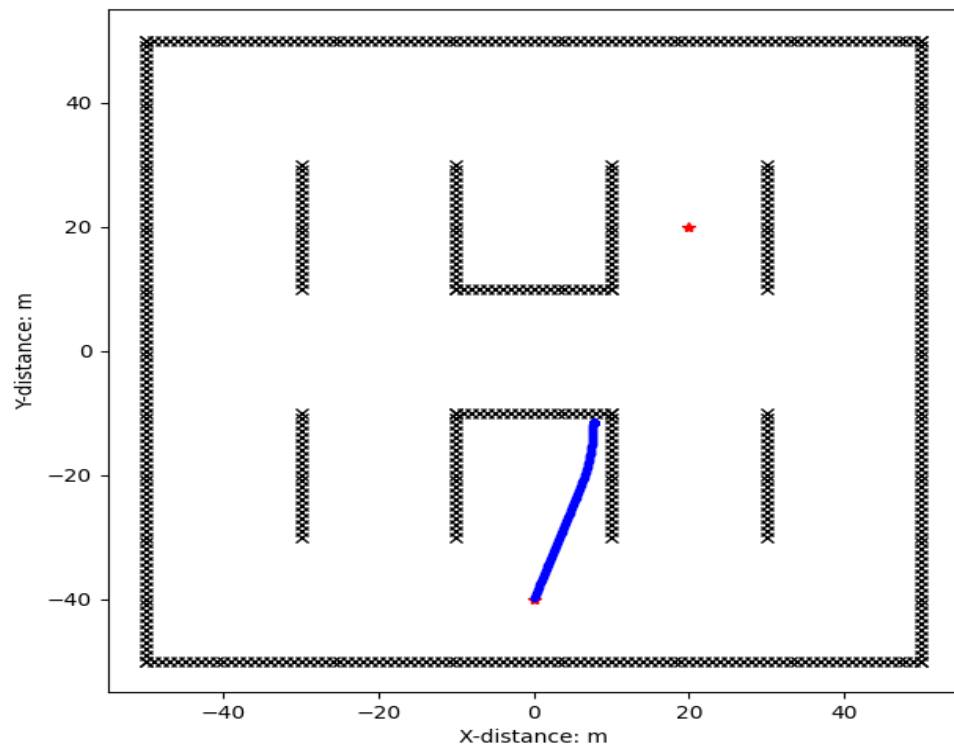
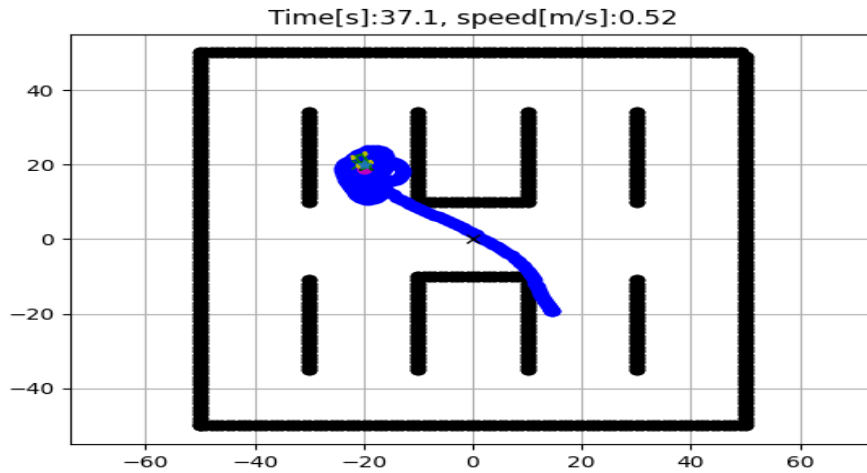
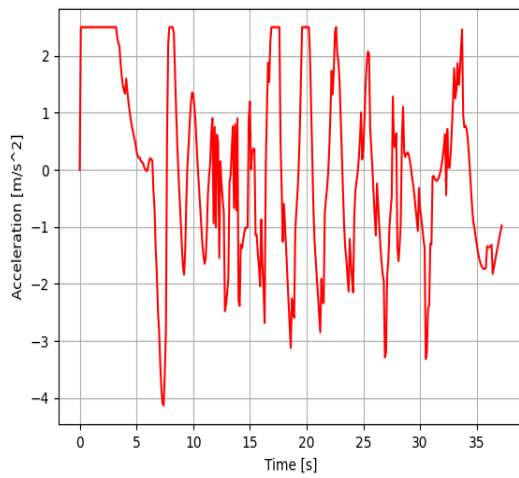


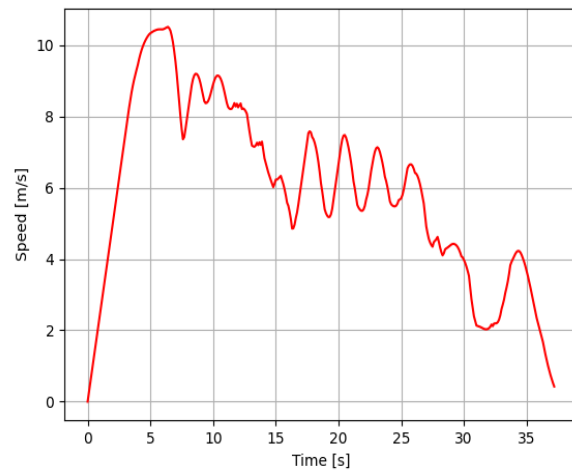
Fig. 5.13 The problem of local minima potential field



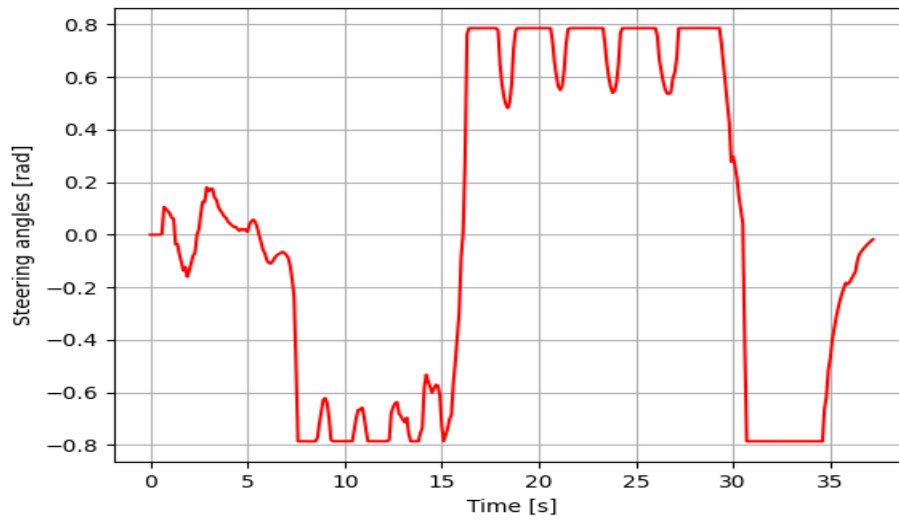
(a) Simulation of path following



(b) Acceleration vs time

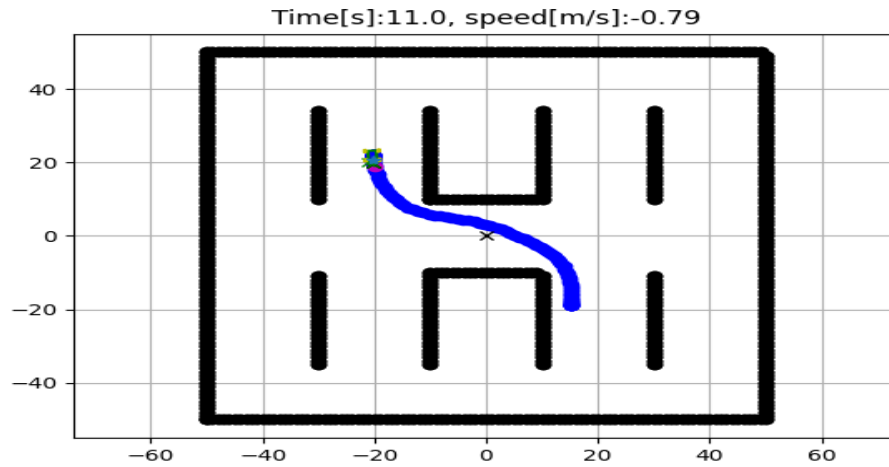


(c) Speed vs time

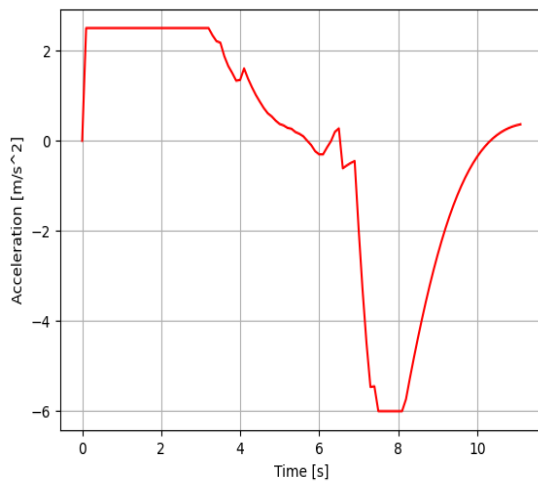


(d) Steering angle vs time

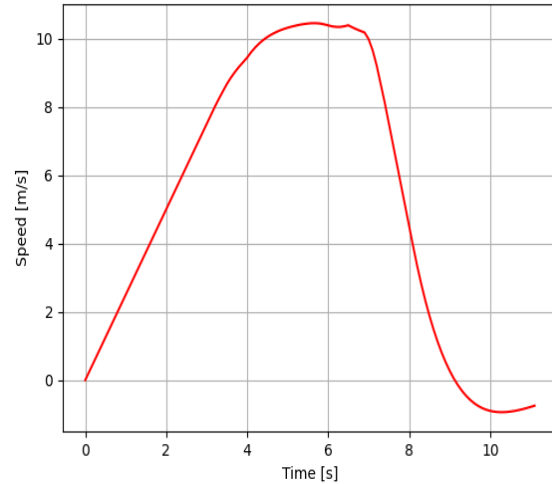
Fig. 5.14 System response where $\zeta = 1, \eta = 1$



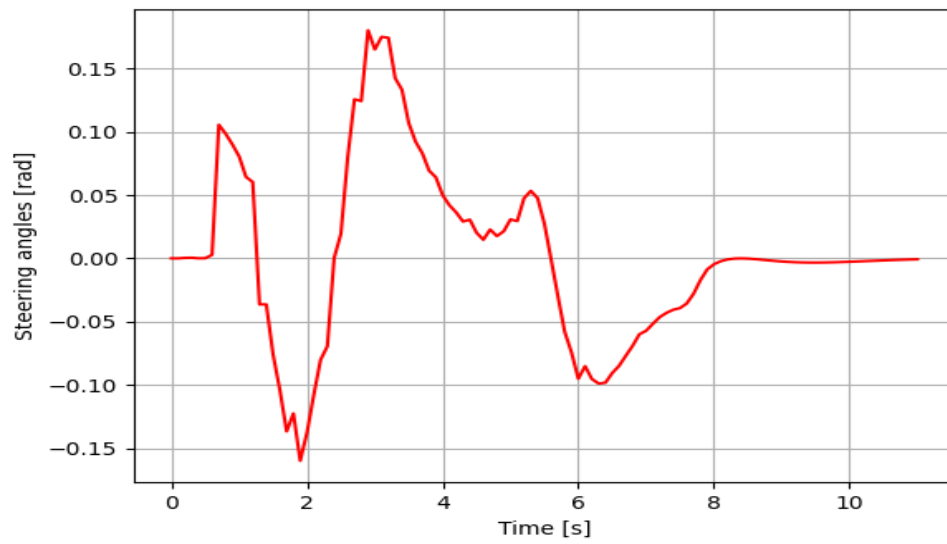
(a) Simulation of path following



(b) Acceleration vs time

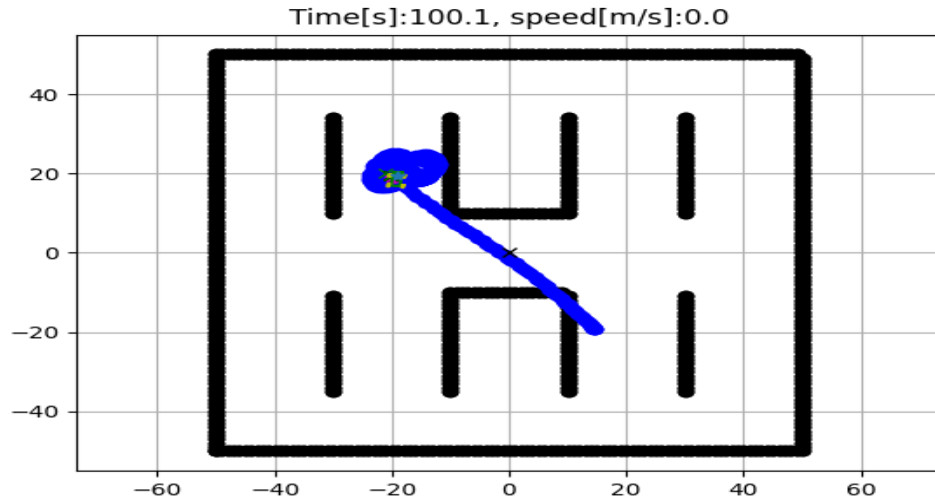


(c) Speed vs time

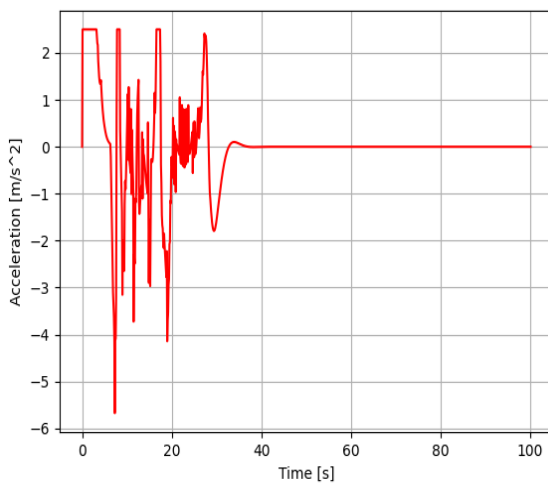


(d) Steering angle vs time

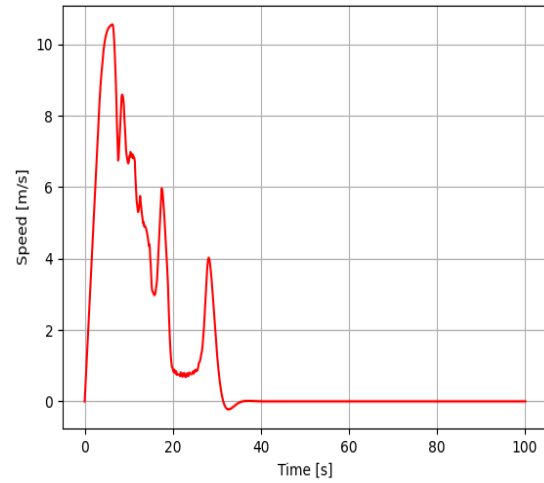
Fig. 5.15 System response where $\zeta = 1, \eta = 100$



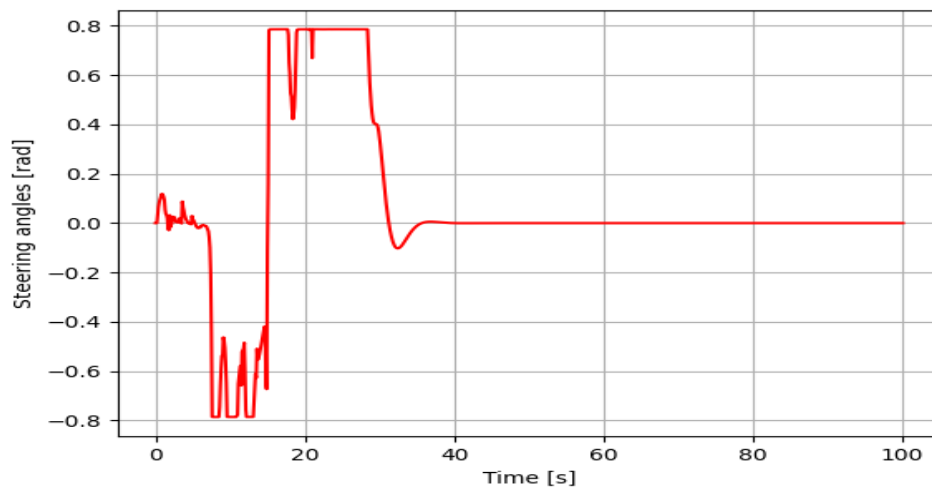
(a) Simulation of path following



(b) Acceleration vs time

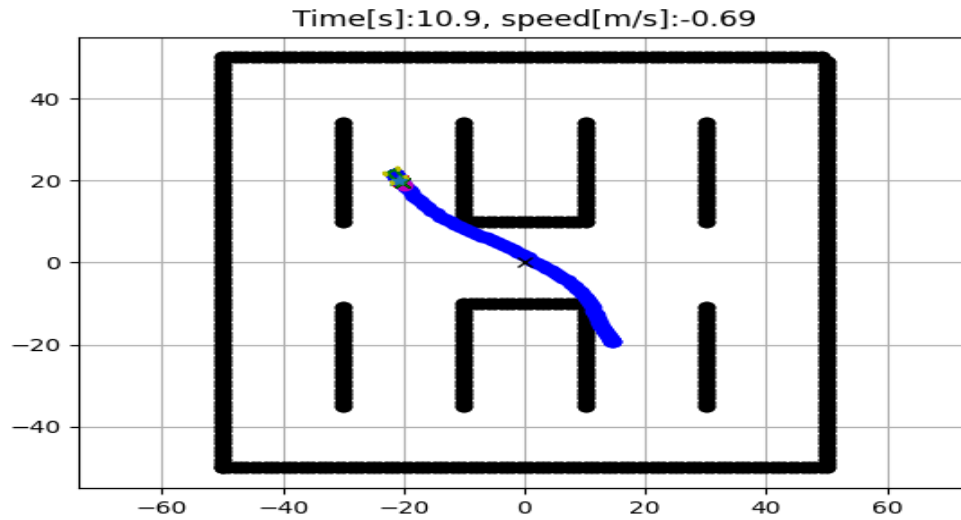


(c) Speed vs time

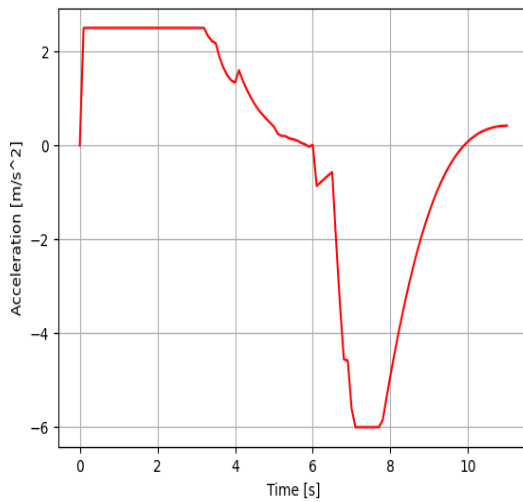


(d) Steering angle vs time

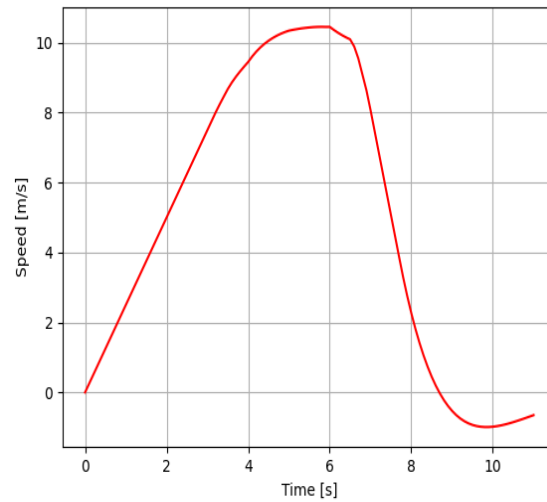
Fig. 5.16 System response where $\zeta = 100, \eta = 1$



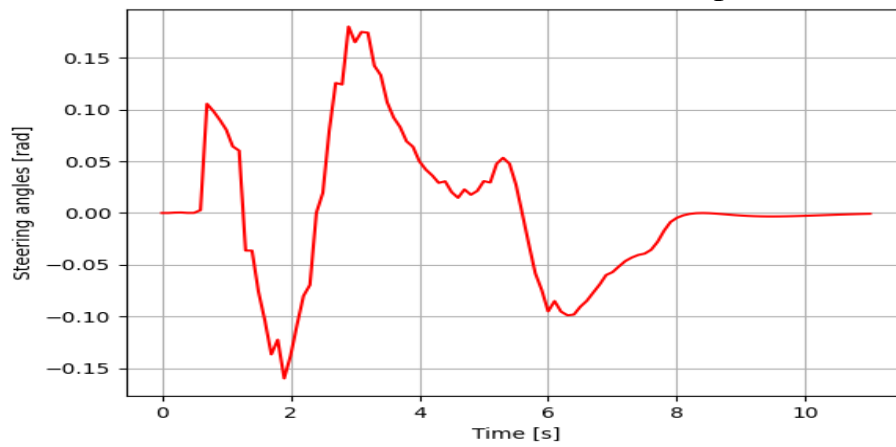
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

Fig. 5.17 System response where $\zeta = 100, \eta = 100$

5.3 Using a Neural Network with Predictive Control

To evaluate the effect of adjusting the rate of change of the steering angle on the response of the system, the simulation was tested both with and without the use of the neural network with model predictive control unit for the same path.

to clarify the results of the simulation, Fig. 5.18 displays the response of the steering angle over time for both the scenario in which the neural network and predictive control unit were used, as well as the scenario in which they were not utilized with the predictive control.

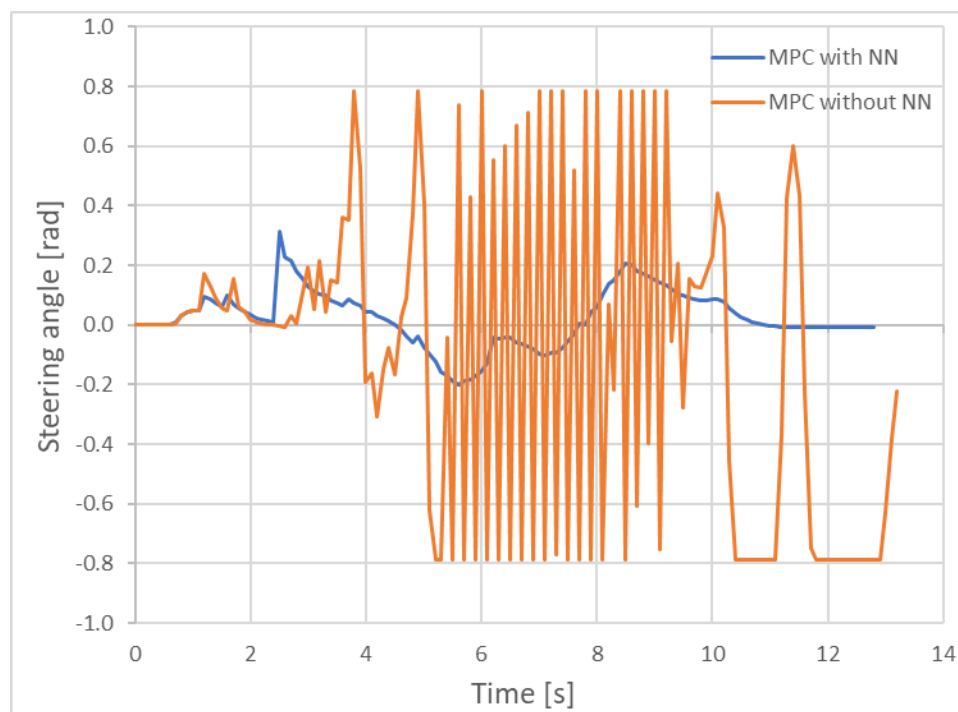


Fig. 5.18 System steering angle response with and without use neural network

Fig. 5.19 displays the response of the acceleration over time for both the scenario in which the neural network and predictive control unit were used, as well as the scenario in which they were not utilized with the predictive control.

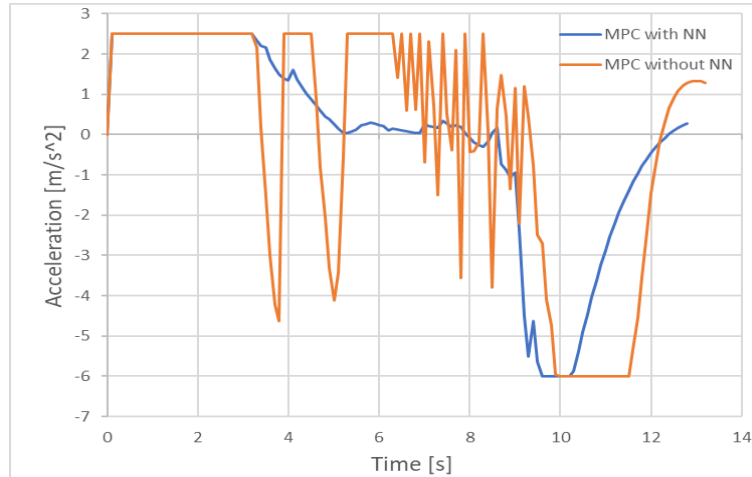


Fig. 5.19 System acceleration response with and without use neural network

Fig. 5.20 displays the response of the speed over time for both the scenario in which the neural network and predictive control unit were used, as well as the scenario in which they were not utilized with the predictive control.

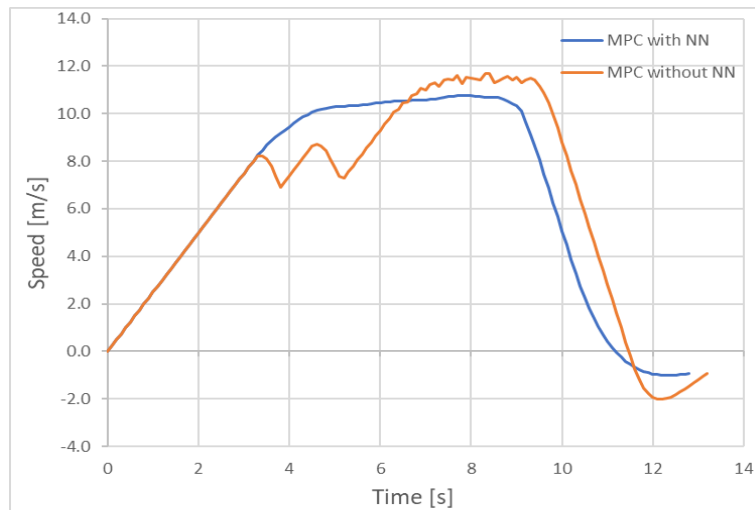


Fig. 5.20 System velocity response with and without use neural network

The simulation results showed that adjusting the change between steering angle rates had a significant effect on the speed and acceleration of

the self-driving car. When using the neural network, the self-driving car was able to achieve a smoother and more consistent speed and acceleration throughout the path. The results showed that the self-driving car was able to maintain a more stable speed and acceleration, with less variation in these parameters, when the change between steering angle rates was optimized using the neural network.

In contrast, when the neural network was not used, the self-driving car's speed and acceleration were more erratic and less stable, with more variation in these parameters throughout the path. This indicates that the neural network was able to improve the self-driving car's performance and stability by optimizing the change between steering angle rates.

Overall, these results suggest that the use of a neural network can be an effective way to optimize a self-driving car's performance and ensure its safety and reliability on the road. By adjusting the change between steering angle rates, the self-driving car can achieve a more stable and consistent speed and acceleration, this can lead to a more comfortable ride for passengers, reduce the risk of accidents, and enable more accurate and precise control of the car movements, which is essential for safety and reliability on the road.

5.4 Obstacle Avoidance in a Constrained Environment Results

The proposed system was tested in a closed environment without obstacles and in a closed environment with moving obstacles that may be human or other cars. The movement towards the target is carried out by following the initial path provided by the A* algorithm as shown in Fig. 5.21. As for the moving obstacles in the path, they are detected using the potential field generated by the obstacle during the step-by-step planning of the local path as shown in Fig. 5.22.

The path that the car follows is simulated, where the NMPC control unit generates inputs within the specified constraints for the following path and according to the type of environment, where Fig 5.23(a) represents the path if the environment is fixed, and the Fig. 5.24(a) represents the path if there are obstacles that were avoided during the simulation. It tries to maintain the desired speed with the least positional error. Every jerk in steering angle and acceleration is like a jerk on a car. Fig. 5.23(b), and Fig. 5.24(b) represent the acceleration of the cars, and the variable speed rate is maintained in controlling the car when the maximum acceleration must be less than the acceptable acceleration of 2.5 m/s^2 . Fig. 5.23 (c), and Fig. 5.24 (c) represent the speed of the car vs time, and the variance should be good in speed. The change in steering angle to time is plotted and displayed in Fig. 5.23 (d), and Fig. 5.24 (d). The rate of change of the steering angle in car control must be maintained at a maximum steering angle that is less than the accepted steering angle of 45° .

The system was designed to provide comfort and safety with minimal jerks, and the cost function was tuned accordingly. The simulation results showed that the system was able to achieve a smooth change in acceleration

and steering angle, resulting in a comfortable and safe ride in both static and dynamic environments.

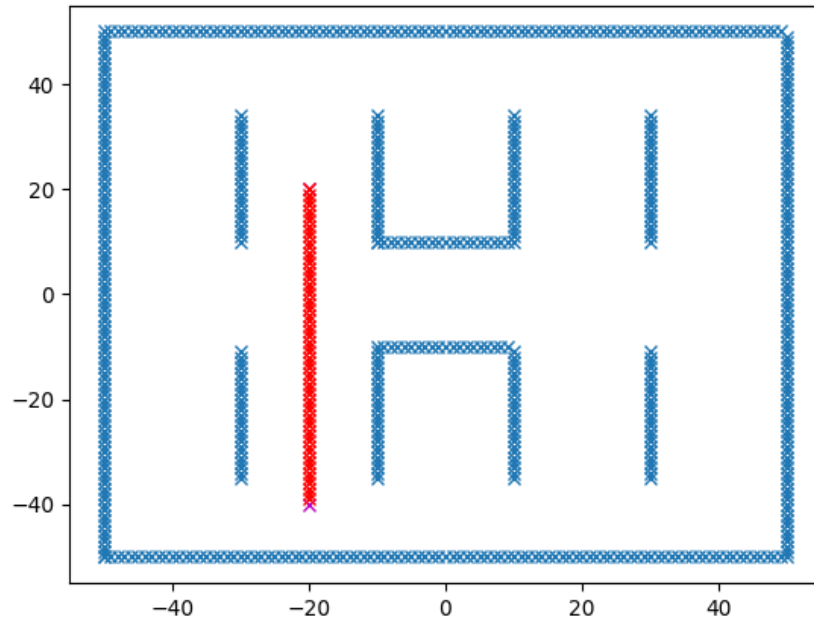


Fig. 5.21 A* path planning from starting point (-20,20) to the target point (-20, -40)

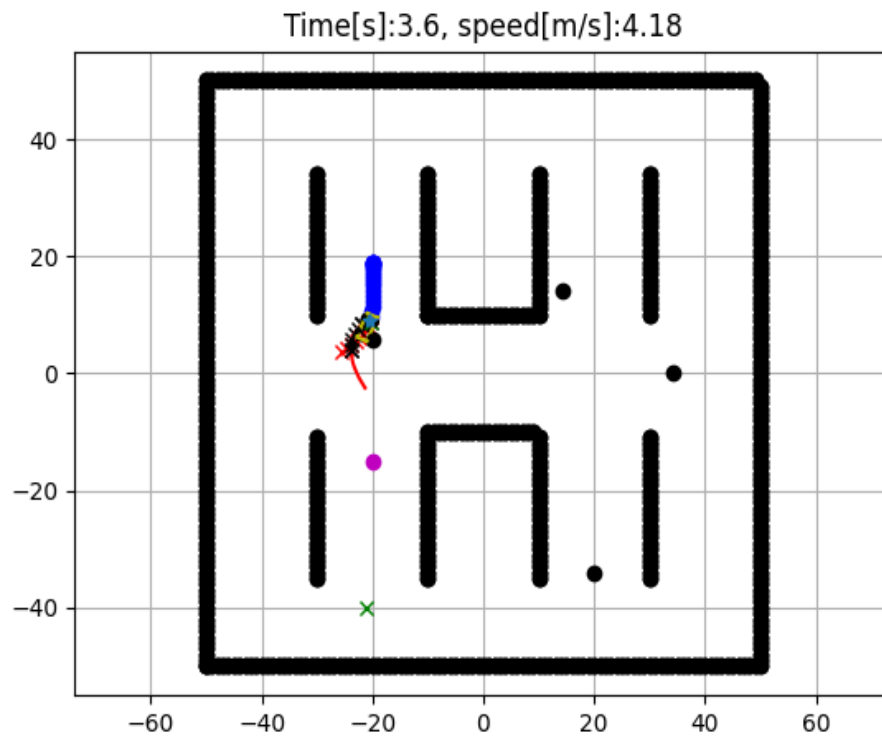
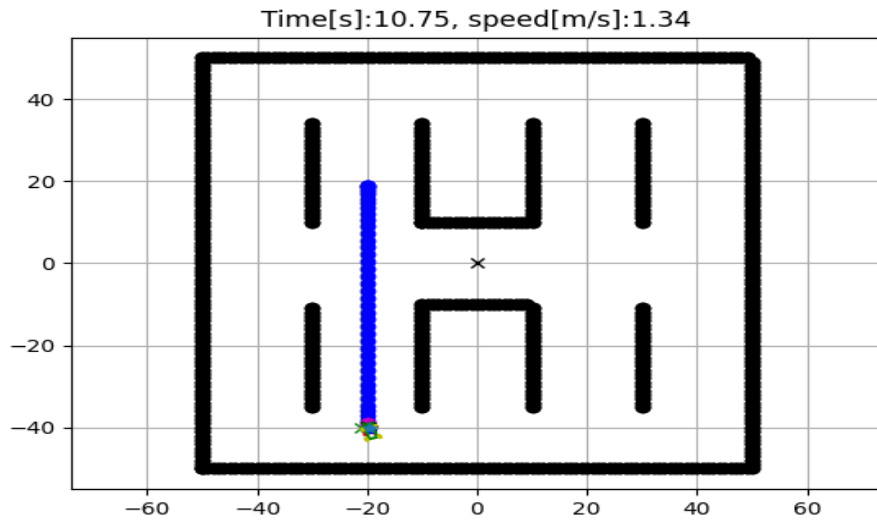
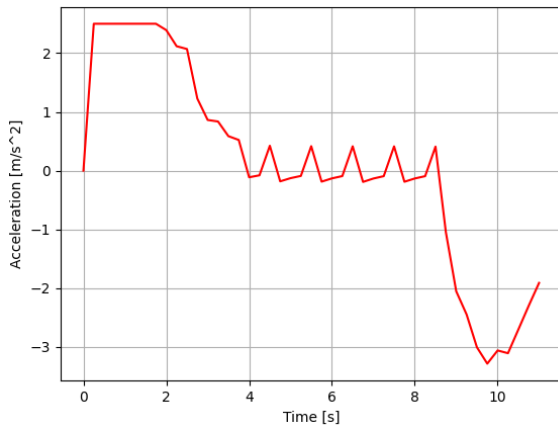


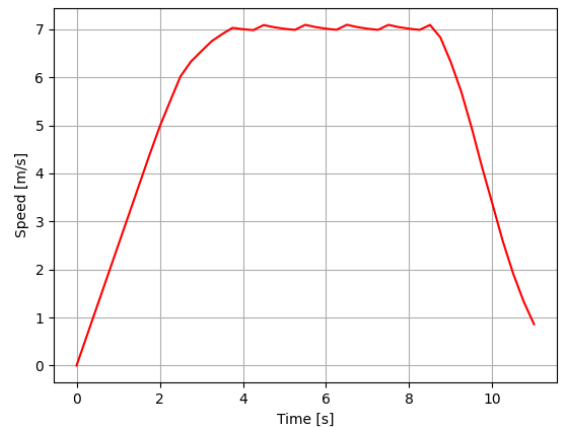
Fig. 5.22 Avoid path obstacle by using the potential field in local planne



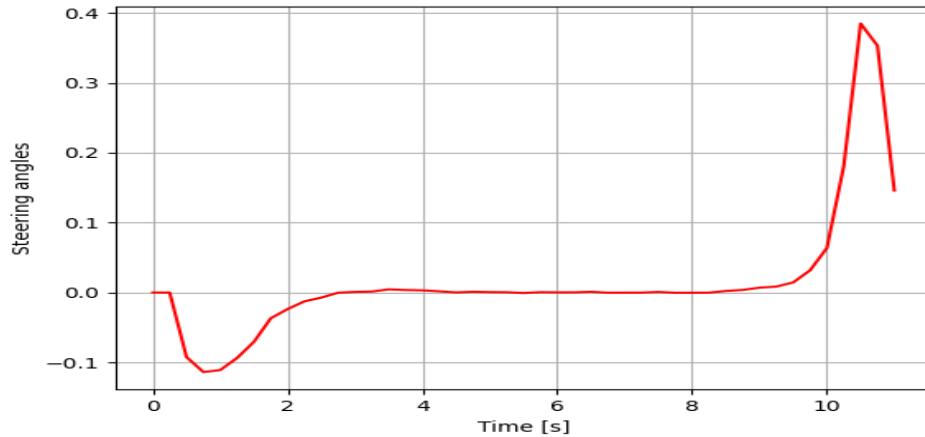
(a) Simulation of path following



(b) Acceleration vs time

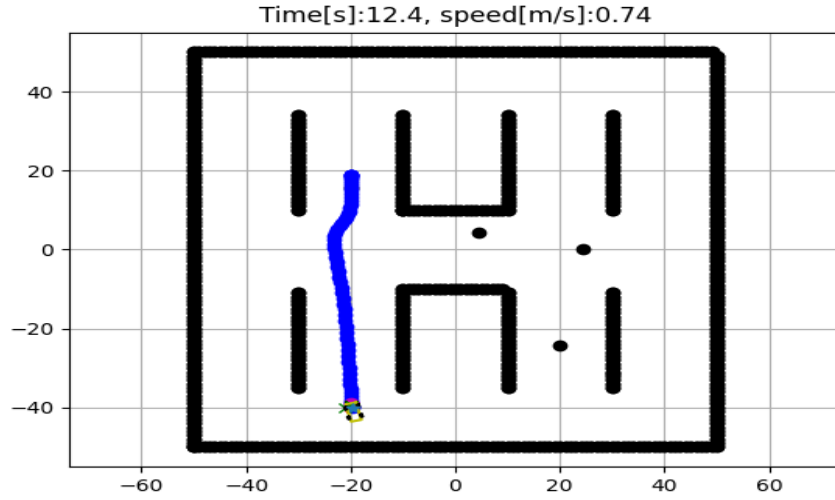


(c) Speed vs time

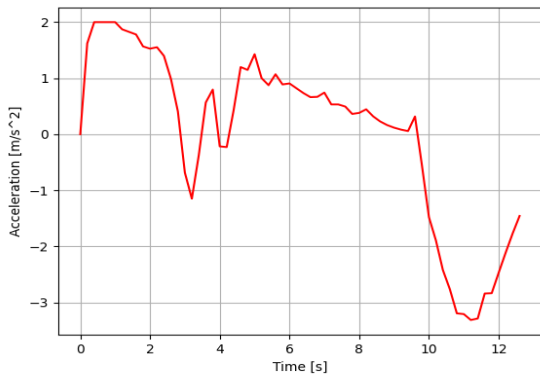


(c) Steering angle vs time

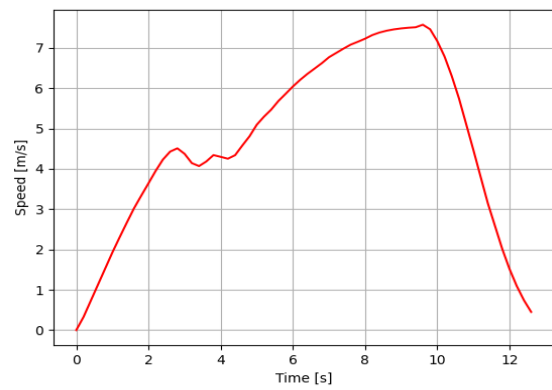
Fig. 5.23 Result static obstacle simulation path following from (-20,20) to (-20, -40)



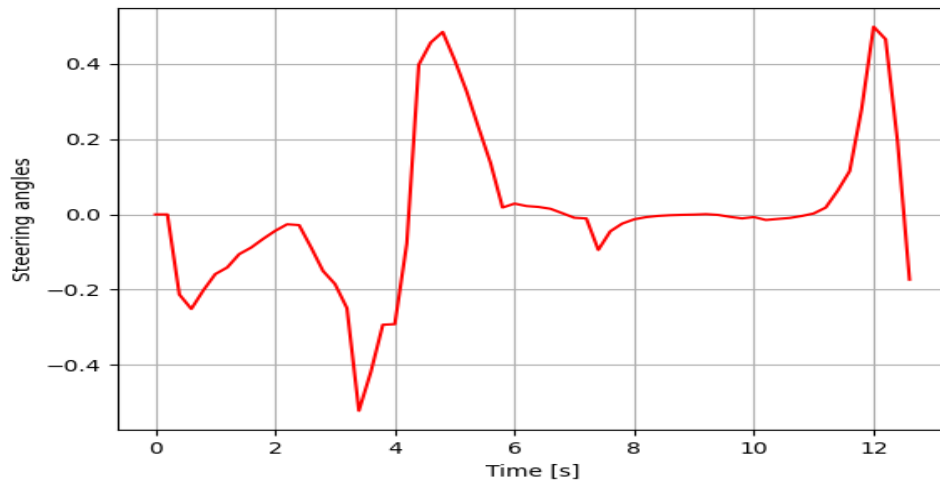
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

Fig. 5.24 Result dynamic obstacle simulation path following from (-20,20) to (-20, -40)

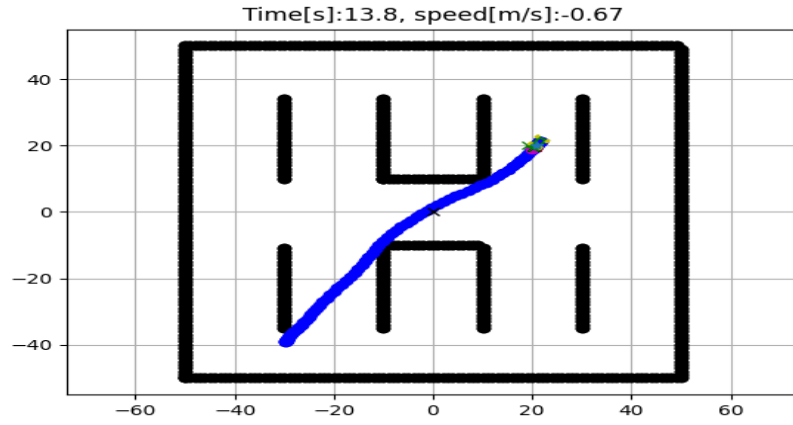
The performance of the controller is affected by:**1- Cost function**

The cost function is a crucial component of the controller as it determines the trade-off between different performance metrics, such as comfort, safety, and energy efficiency. The choice of cost function can significantly impact the behavior of the controller and the resulting car trajectory. In this proposed system, the controller calculates the cost of its actions at each point in the trajectory, taking into account a range of objectives such as minimizing the discrepancy between the desired and actual position and orientation of the car, reducing the control input necessary to manage the car, and minimizing the rate of change of the input.

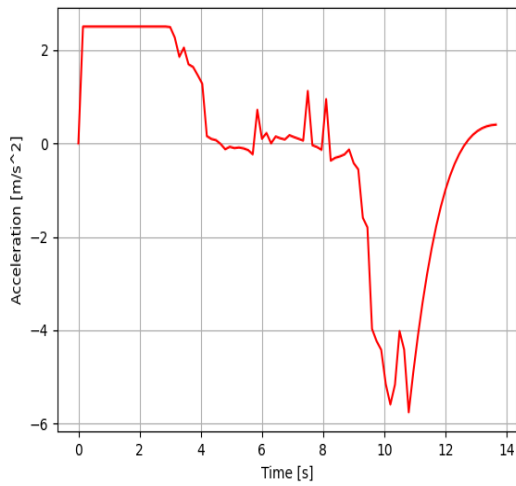
2- weights in the cost matrix

The weight of the cost matrix in equation (4.1) is used to assign weights to each state variable in the cost function based on their relative importance. These weights determine the trade-off between the different objectives in the cost function. If a state variable has a higher weight, it will be prioritized more in the cost function, and the controller will try to minimize its deviation from the desired value more aggressively. On the other hand, if a state variable has a lower weight, the controller will be less concerned with minimizing its deviation from the desired value. The weights of the weight cost matrix should be carefully chosen based on the specific application to achieve the desired balance between different objectives. The weights of the cost matrix were increased and decreased for two different paths to see their effect.

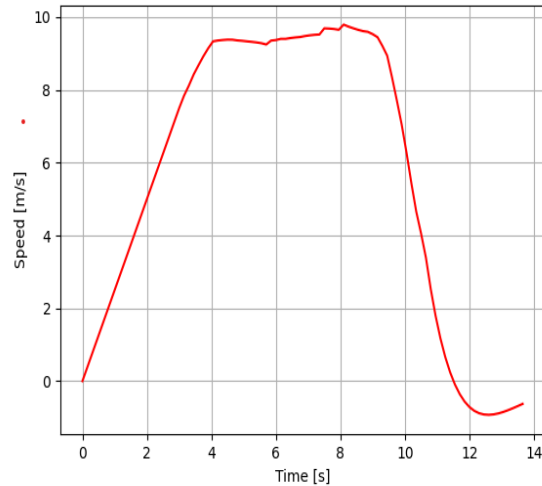
- **Input cost weights:** The simulation results for the self-driving car with varying input cost weights are presented in Fig. 5.25 and Fig. 5.26 for low input weight, while Fig. 5.27 and Fig. 5.28 illustrate the simulation results for high input weight.



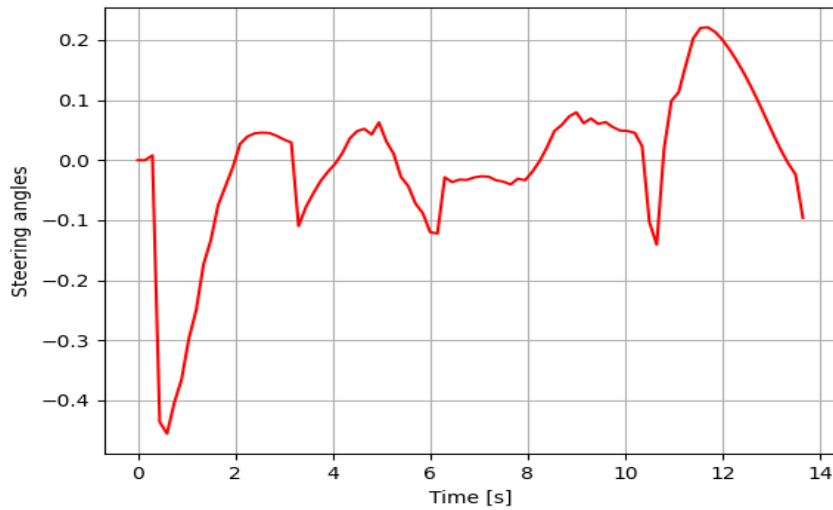
(a) Simulation of path following



(b) Acceleration vs time

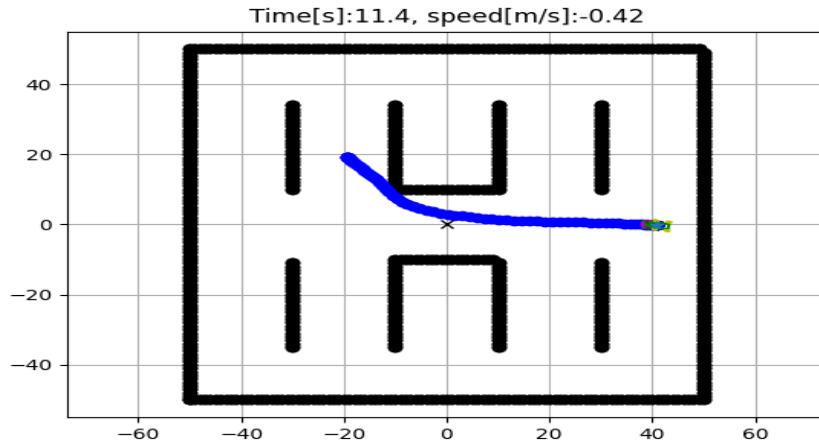


(c) Speed vs time

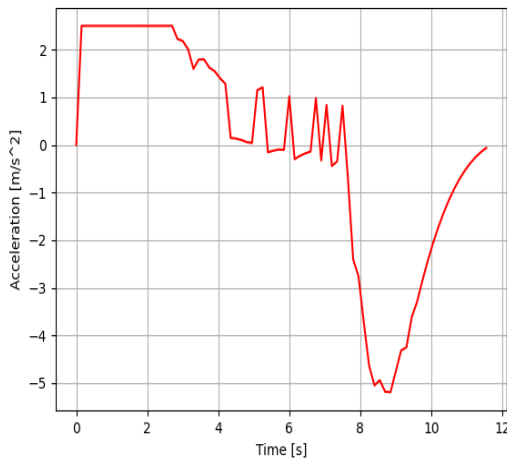


(d) Steering angle vs time

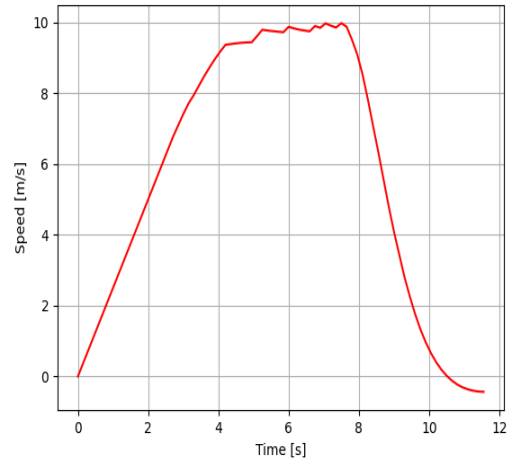
Fig. 5.25 Input cost weights low, path from (-30,40) to (20,20)



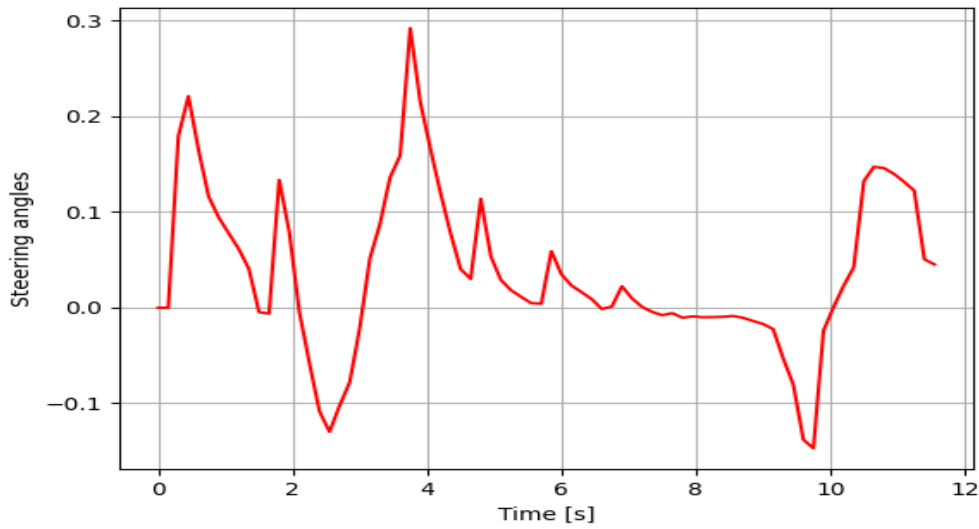
(a) Simulation of path following



(b) Acceleration vs time

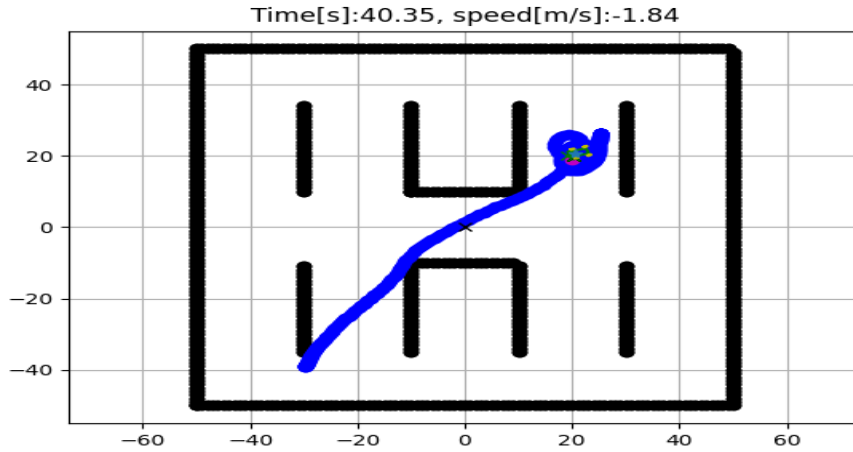


(c) Speed vs time

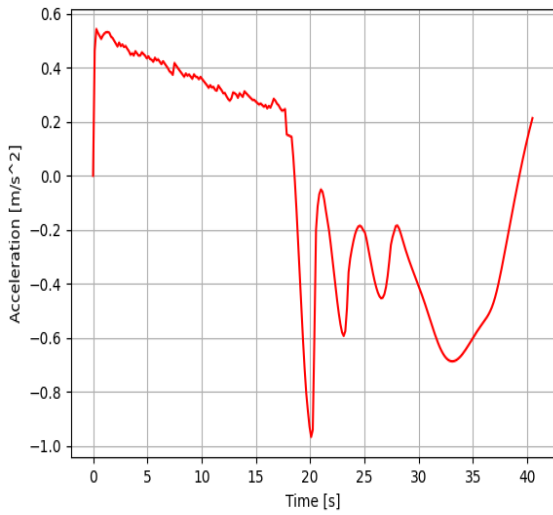


(d) Steering angle vs time

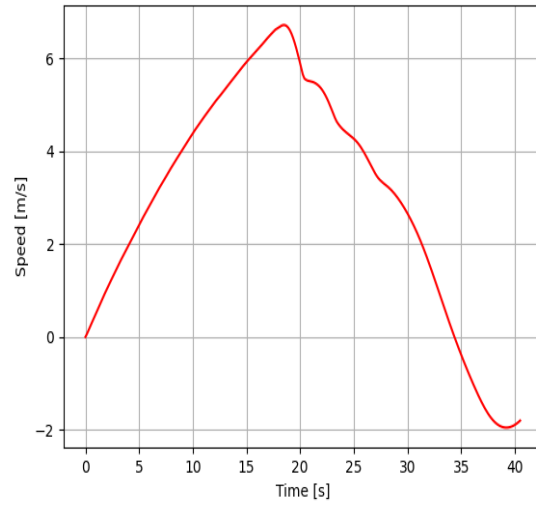
Fig. 5.26 Input cost weights low, path from (-20,20) to (40,0)



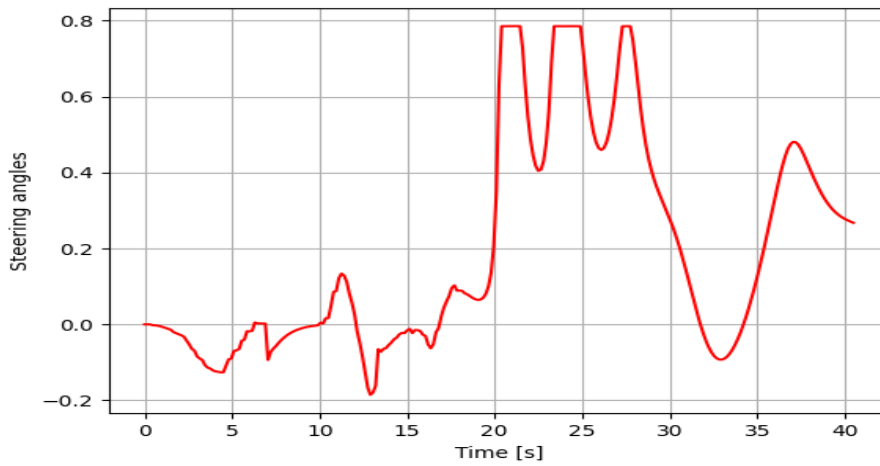
(a) Simulation of path following



(b) Acceleration vs time

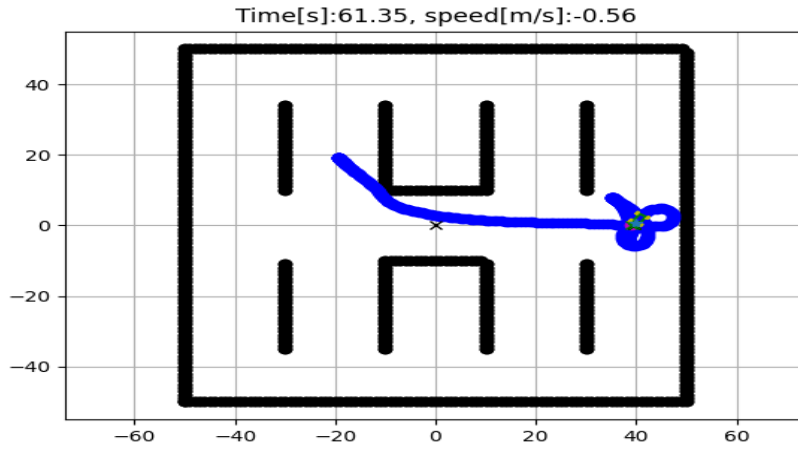


(c) Speed vs time

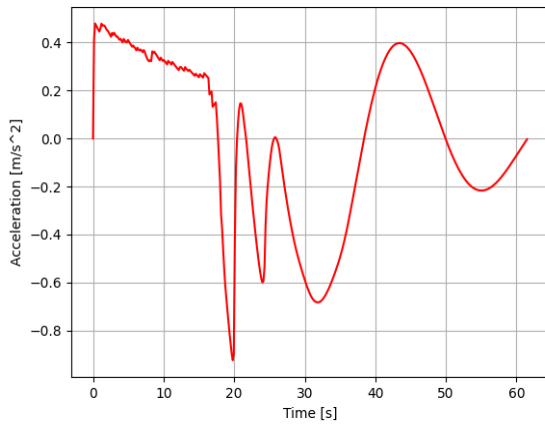


(c) Steering angle vs time

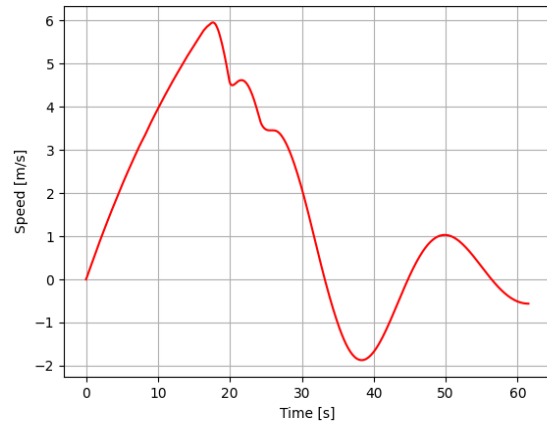
Fig. 5.27 Input cost weights high, path from (-30,40) to (20,20)



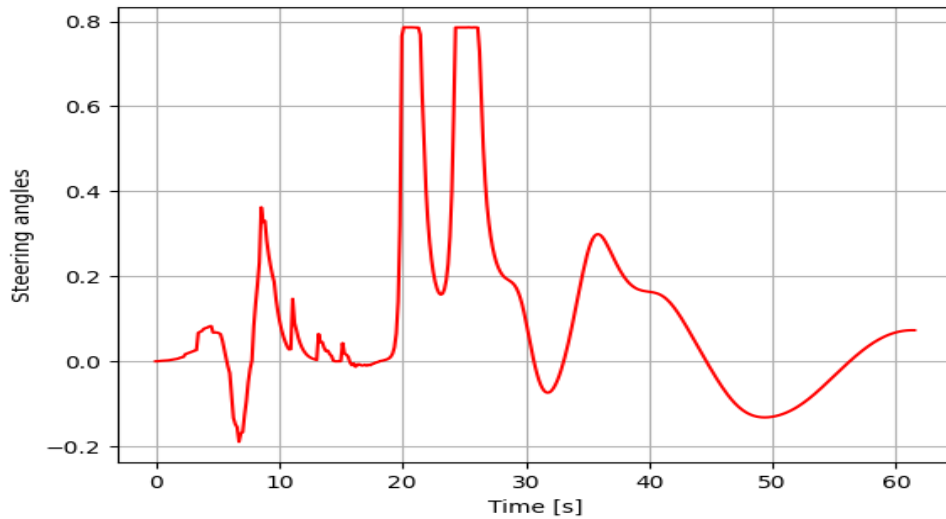
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

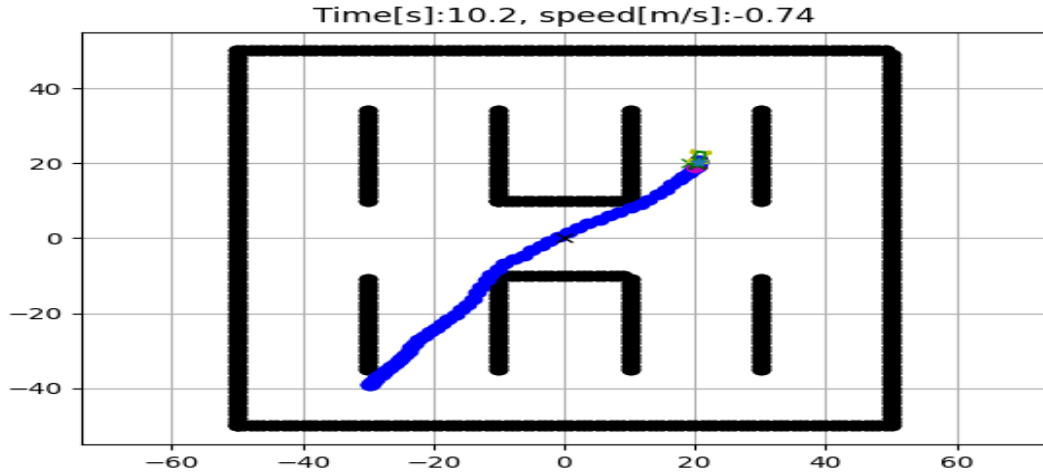
Fig. 5.28 Input cost weights high, path from (-20,20) to (40,0)

From the simulation result, the behavior of self-driving cars during simulation is significantly influenced by the input cost weights (R1).

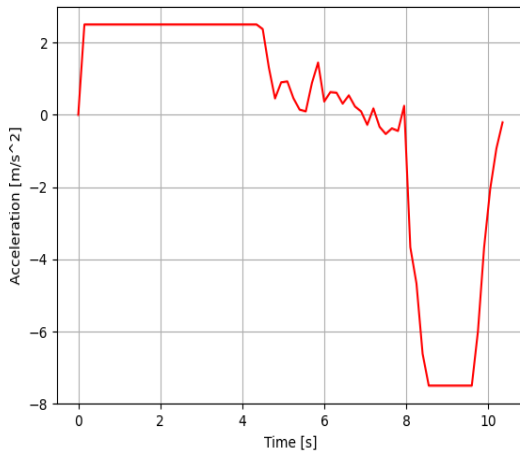
In simulations where R1 is set to low values, the controller prioritizes keeping the state of the car close to its desired values instead of controlling inputs. This leads to larger and more aggressive control inputs, resulting in a dynamic and responsive driving style. However, this behavior may cause the car to make more aggressive maneuvers that could potentially affect passenger comfort.

On the other hand, simulations with high input cost weights result in smaller and less aggressive control inputs, potentially causing the car to repeatedly overshoot its target and deviate from the intended path. The controller prioritizes keeping the control inputs close to their desired values, leading to small and repeated adjustments that may cause an uncomfortable ride for passengers.

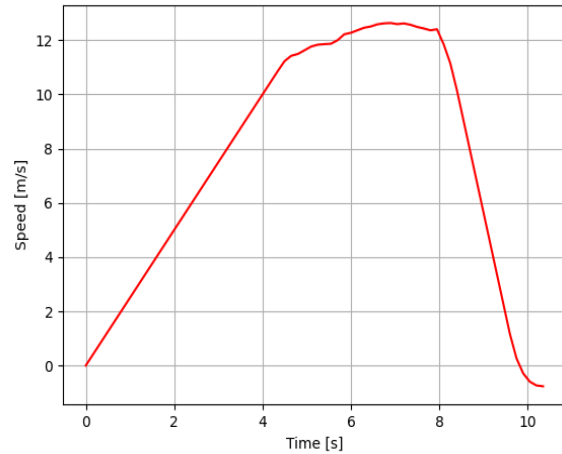
- **State error cost weights:** The impact of different state error cost weights on the behavior of the self-driving car controller is depicted in Fig. 5.29 and Fig. 5.30 for low weight values, and Fig. 5.31 and Fig. 5.32 for high weight values. These simulation results demonstrate how the choice of state error cost weight influences the controller's priority between keeping the state of the car close to its desired values versus the control inputs, and how this affects the driving style and passenger experience.



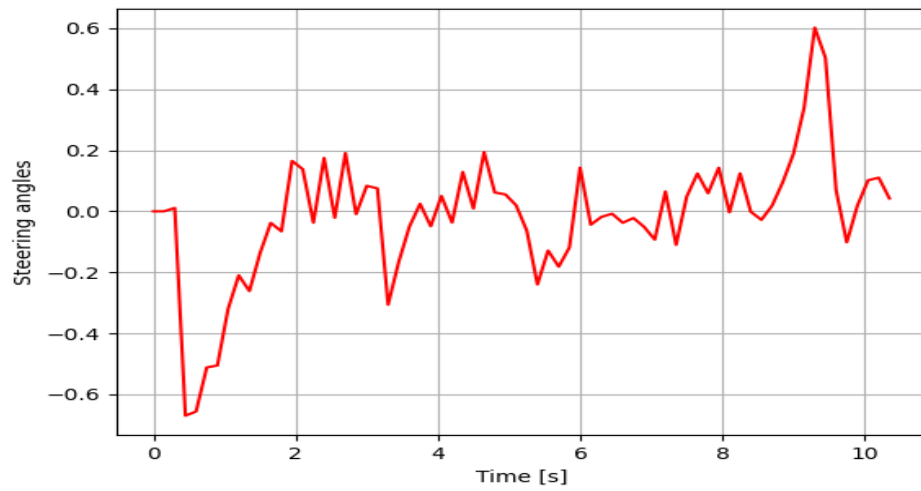
(a) Simulation of path following



(b) Acceleration vs time

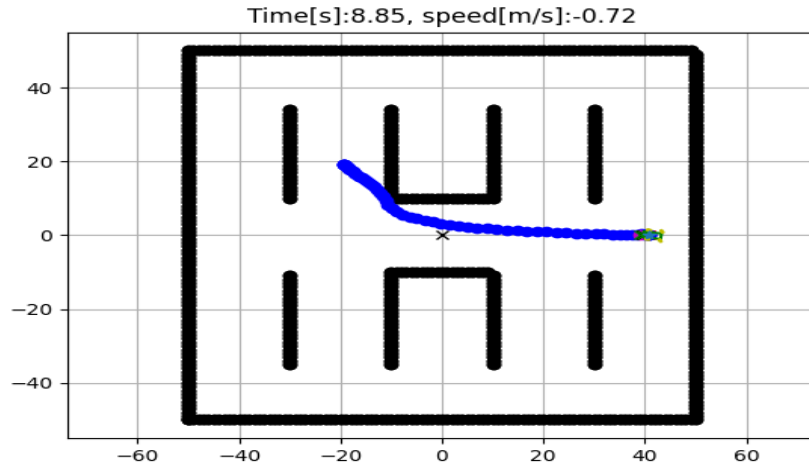


(c) Speed vs time

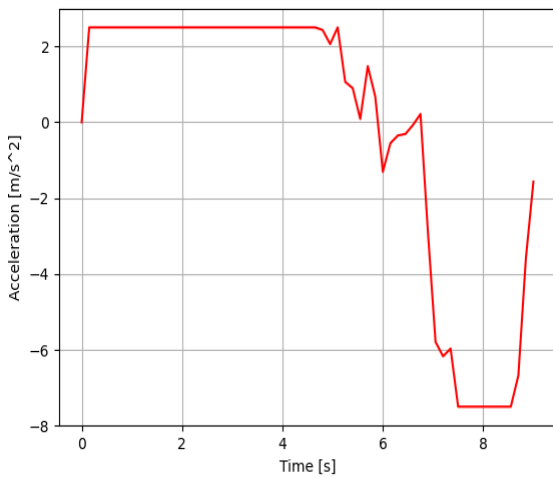


(d) Steering angle vs time

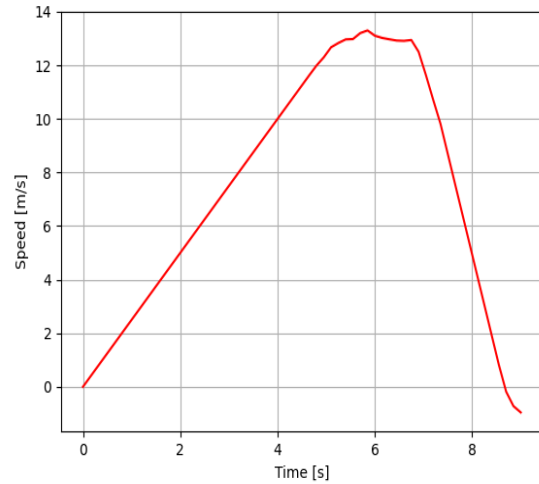
Fig. 5.29 State error cost weights low, path from (-30,40) to (20,20)



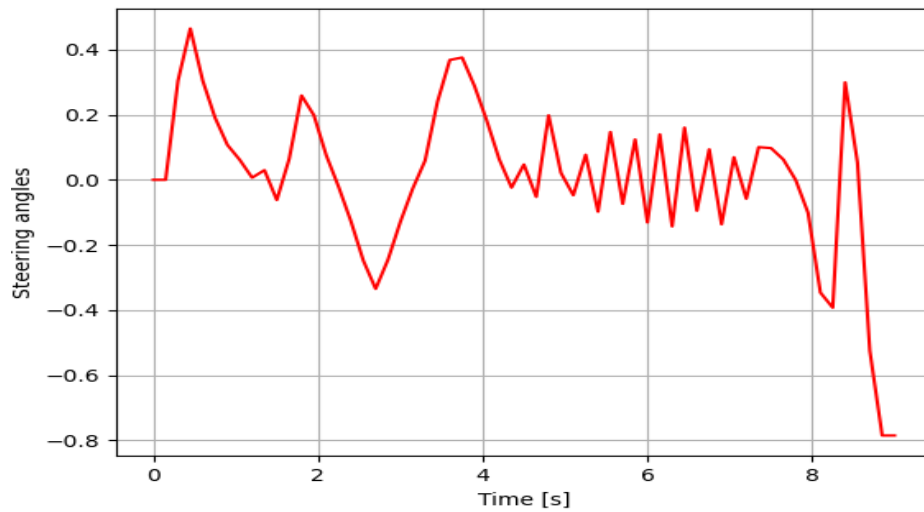
(a) Simulation of path following



(b) Acceleration vs time

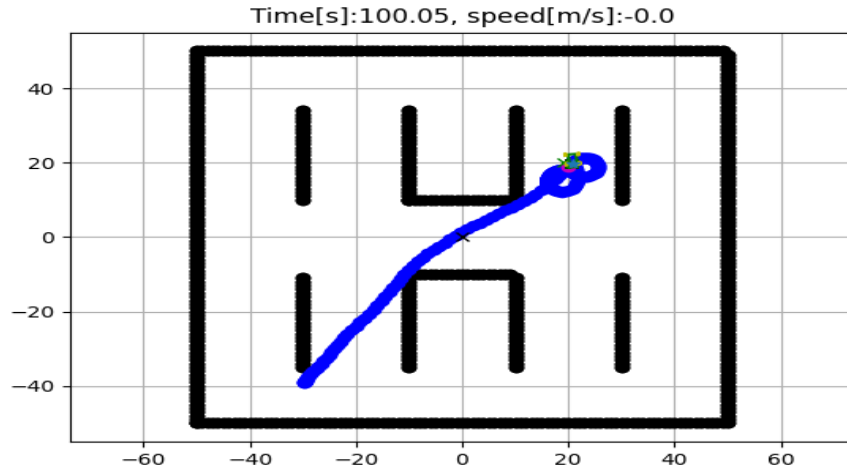


(c) Speed vs time

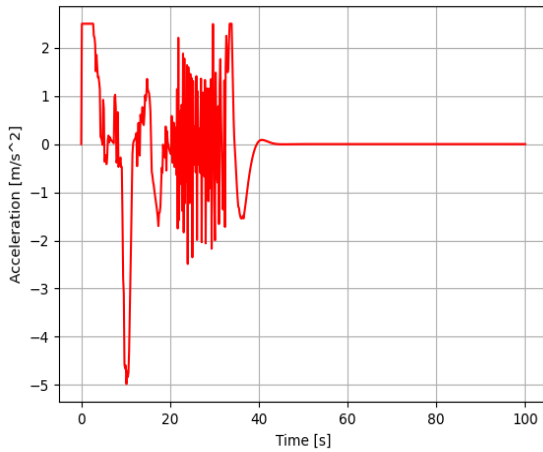


(d) Steering angle vs time

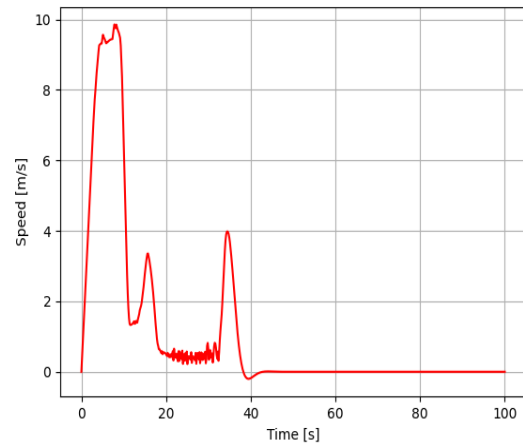
Fig. 5.30 State error cost weights low, path from (20,20) to (40,0)



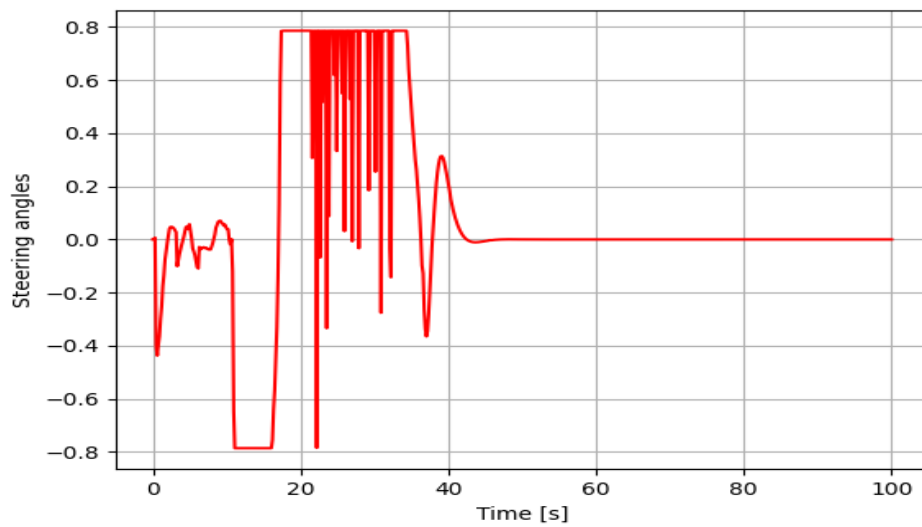
(a) Simulation of path following



(b) Acceleration vs time

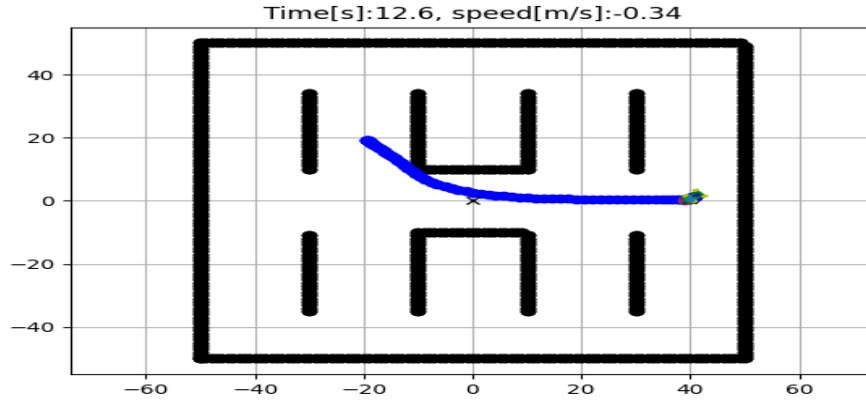


(c) Speed vs time

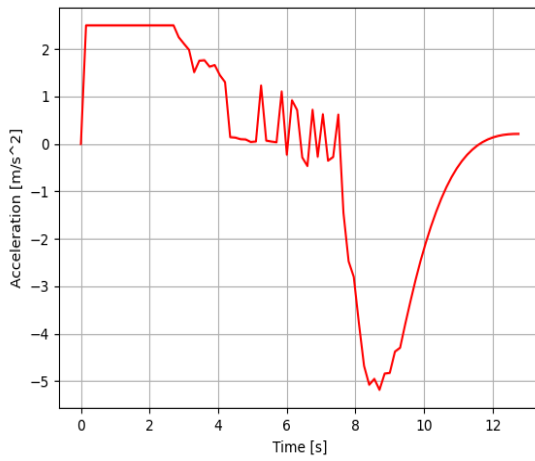


(d) Steering angle vs time

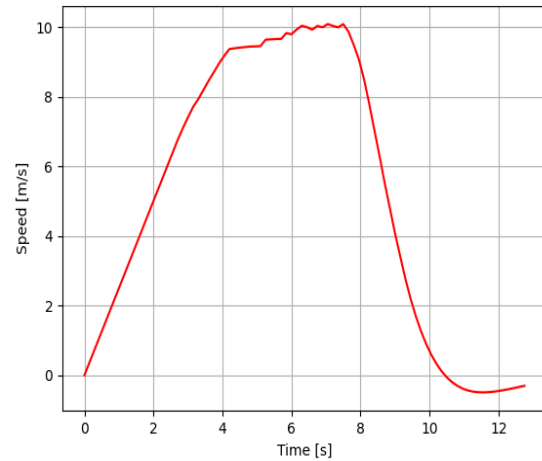
Fig. 5.31 State error cost weights high, path from (-30,-40) to (20,20)



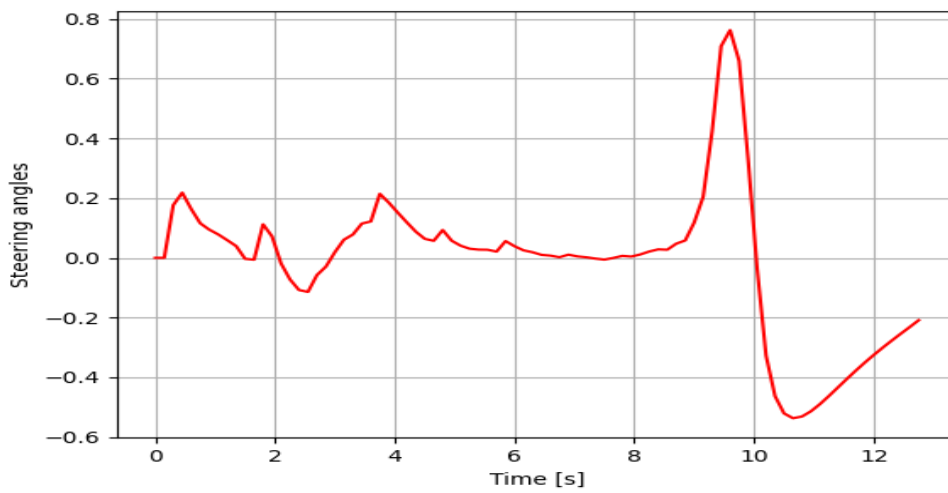
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

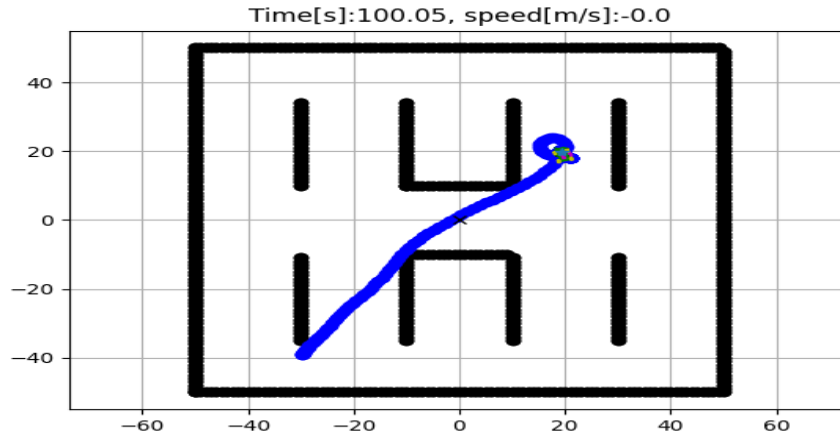
Fig. 5.32 State error cost weights high, path from (-20,20) to (40,0)

The simulation results reveal that the state error cost weights (Q) play a critical role in determining the behavior of the controller in the self-driving car.

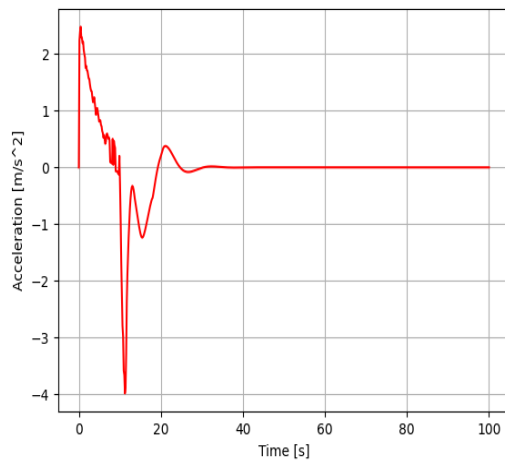
When the state error cost weights are low, the controller focuses more on the control inputs (u) being close to their desired values and less on keeping the state of the car (x) close to its desired values. As a result, the self-driving car simulation with low state error cost weights prioritizes the control inputs, which may lead to imprecise and inaccurate path tracking. The car's behavior could become less predictable, and passengers may experience an uncomfortable ride due to the larger and more aggressive control inputs.

On the other hand, when the state error cost weights are high, the controller focuses more on keeping the state of the car close to its desired values, and less on the control inputs being close to their desired values. This results in a more precise and accurate path tracking, but with larger and more aggressive control inputs that could make the car's behavior less predictable and cause an uncomfortable ride for passengers. The controller may also make smaller and less aggressive control inputs to keep the state of the car close to its desired values, which could cause the car to respond slowly to changes in the environment.

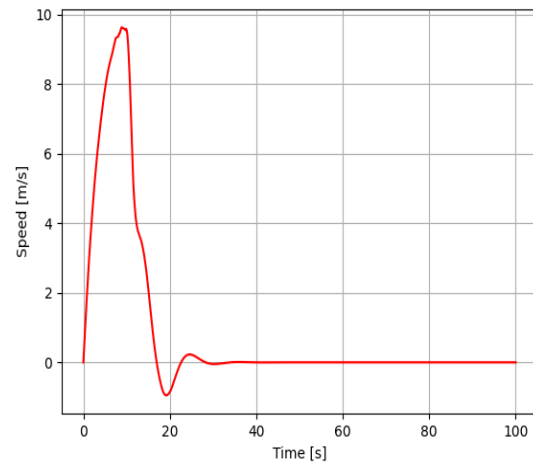
- **Rate input change cost weights:** The simulation results for the self-driving car with varying rate input change cost weights are presented in Fig. 5.33 and Fig. 5.34 for low rate input change weight, while Fig. 5.35 and Fig. 5.36 illustrate the simulation results for high rate input change weight.



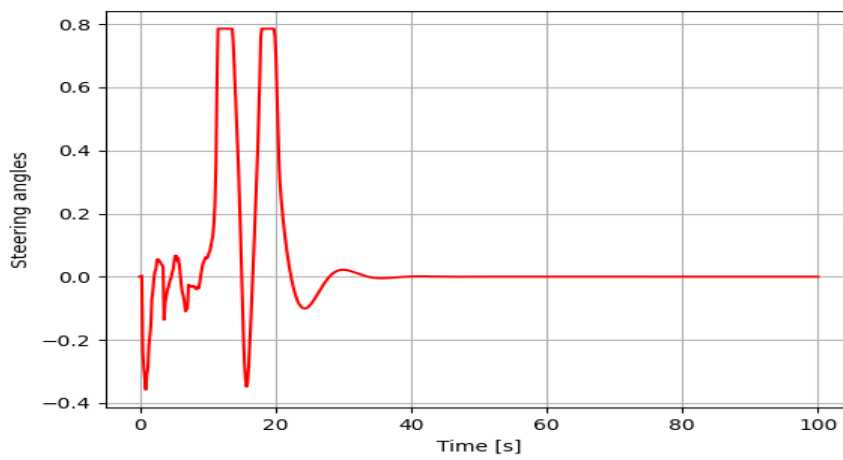
(a) Simulation of path following



(b) Acceleration vs time

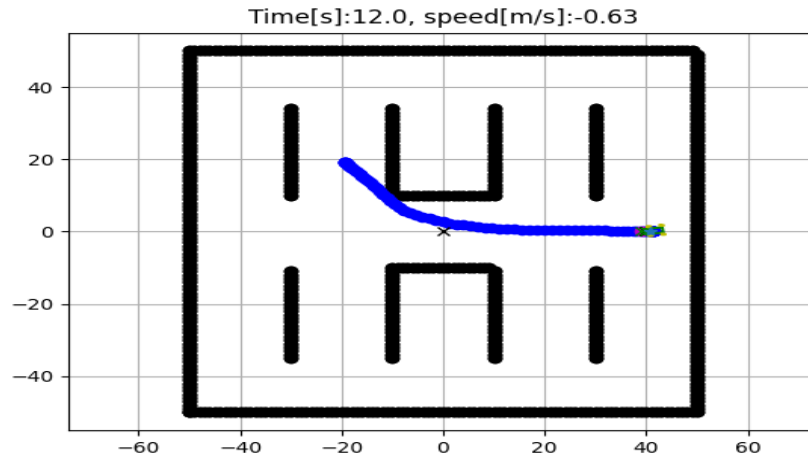


(c) Speed vs time

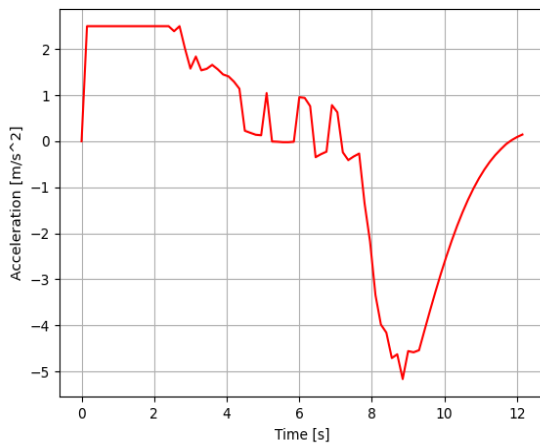


(d) Steering angle vs time

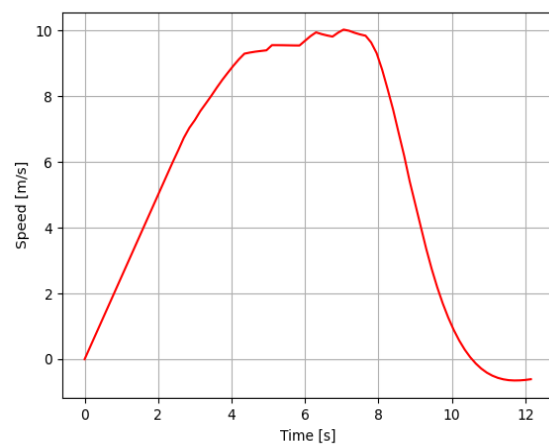
Fig. 5.33 Rate input cost weights low, path from (-30,-40) to (20,20)



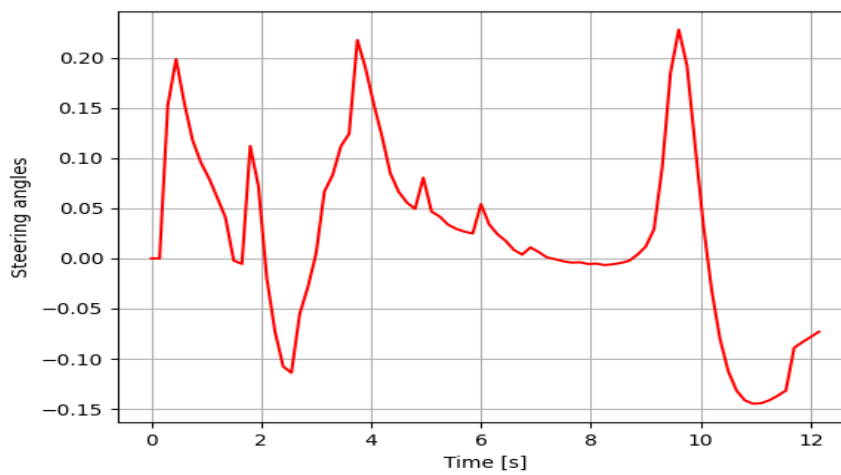
(a) Simulation of path following



(b) Acceleration vs time

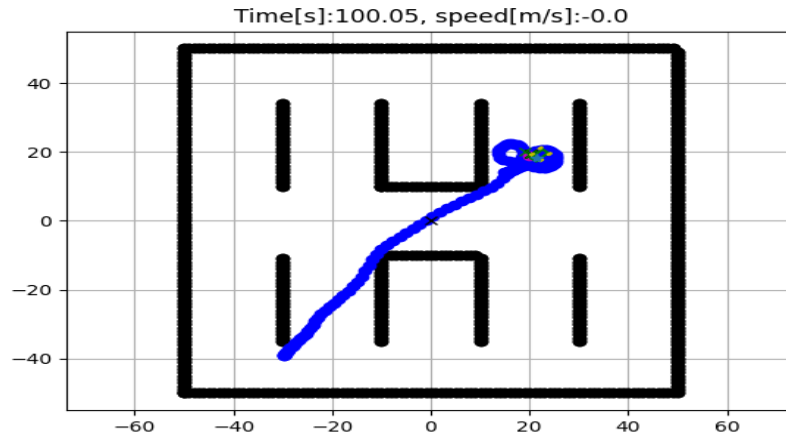


(c) Speed vs time

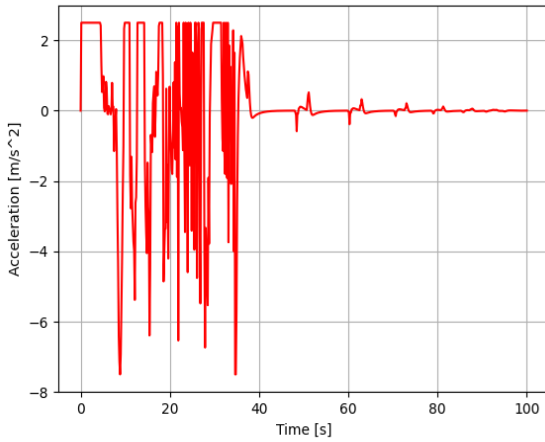


(d) Steering angle vs time

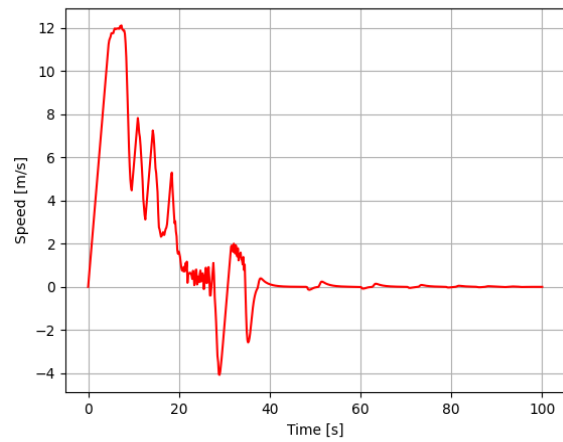
Fig. 5.34 Rate input cost weights low, path from (-20,20) to (40,0)



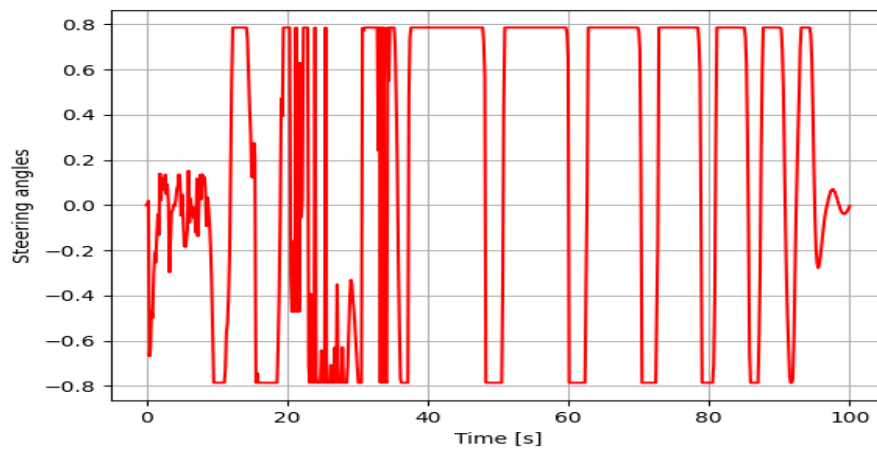
(a) Simulation of path following



(b) Acceleration vs time

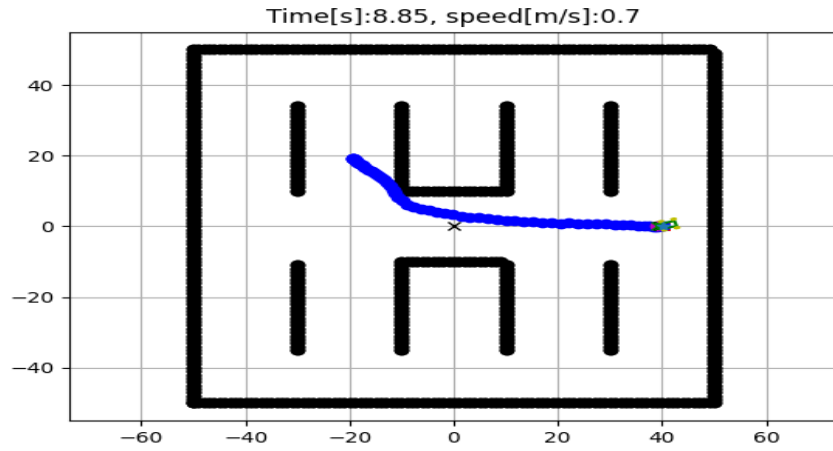


(c) Speed vs time

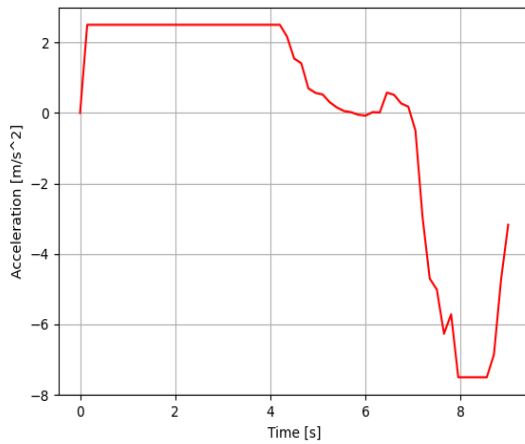


(d) Steering angle vs time

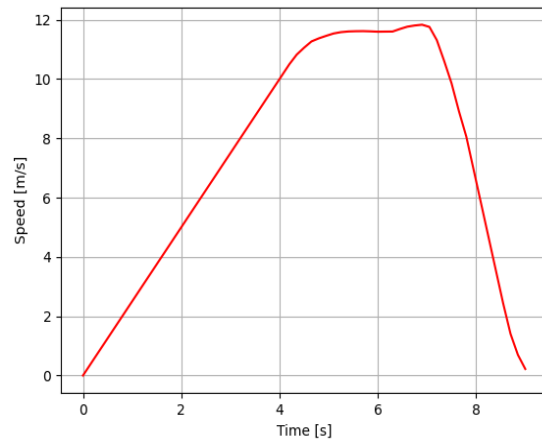
Fig. 5.35 Rate input cost weights high, path from (-30,-40) to (20,20)



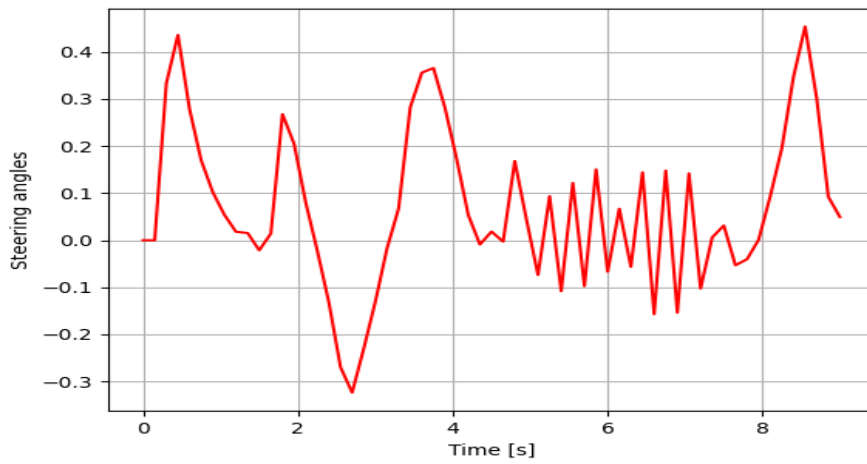
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

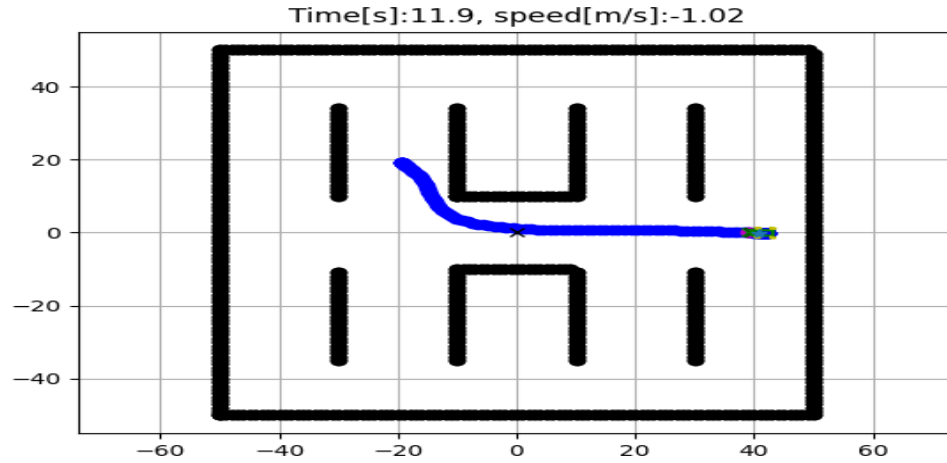
Fig. 5.36 Rate input cost weights high, path from (-20,20) to (40,0)

The simulation results for varying rate input cost weights (R_2) indicate that the choice of R_2 plays an important role in determining the behavior of the self-driving car.

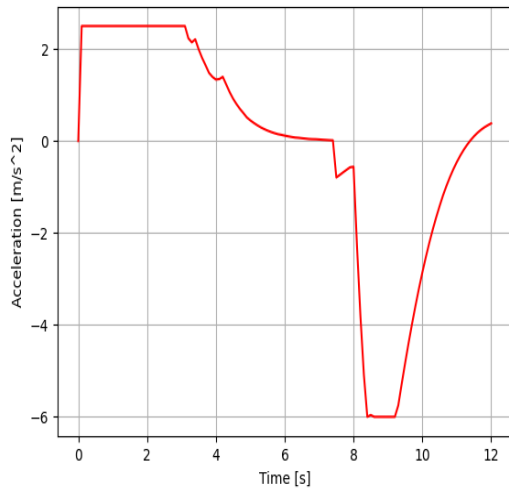
When the cost rate input weights are low, the controller is less concerned with keeping the rate of change of the control inputs small, resulting in larger adjustments to the control inputs. This behavior may lead to a bumpy and unpredictable ride for passengers.

On the other hand, when the cost rate input weights are high, the controller is more concerned with keeping the rate of change of the control inputs small, resulting in smaller adjustments to the control inputs. This behavior may lead to longer convergence time between path steps and could result in overshooting or not following the intended path closely. However, small adjustments can help to reduce energy consumption and prevent large and unnecessary changes in the control inputs.

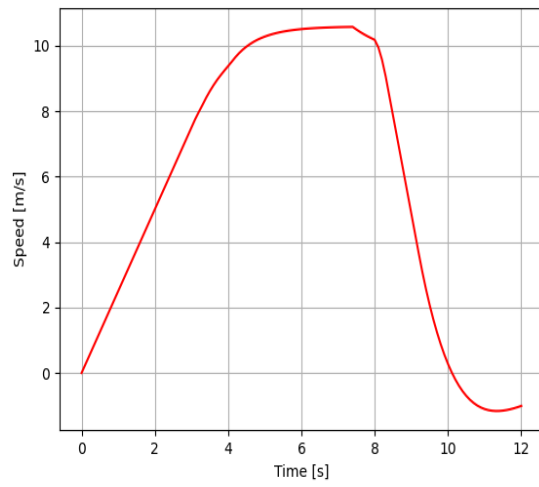
- **Terminal cost weights:** The simulation results of the self-driving car with different terminal cost weights are presented in Fig. 5.37 and Fig. 5.38 for low terminal weight, while Fig. 5.39 and Fig. 5.40 illustrate the simulation results for high terminal weight.



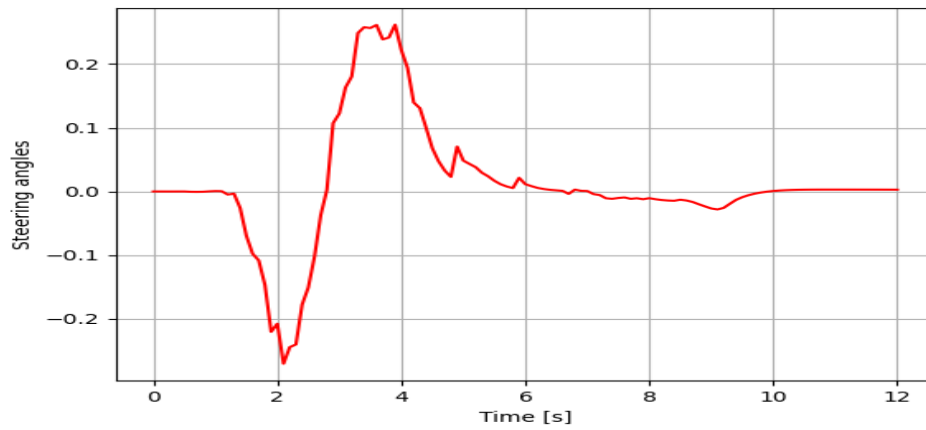
(a) Simulation of path following



(b) Acceleration vs time

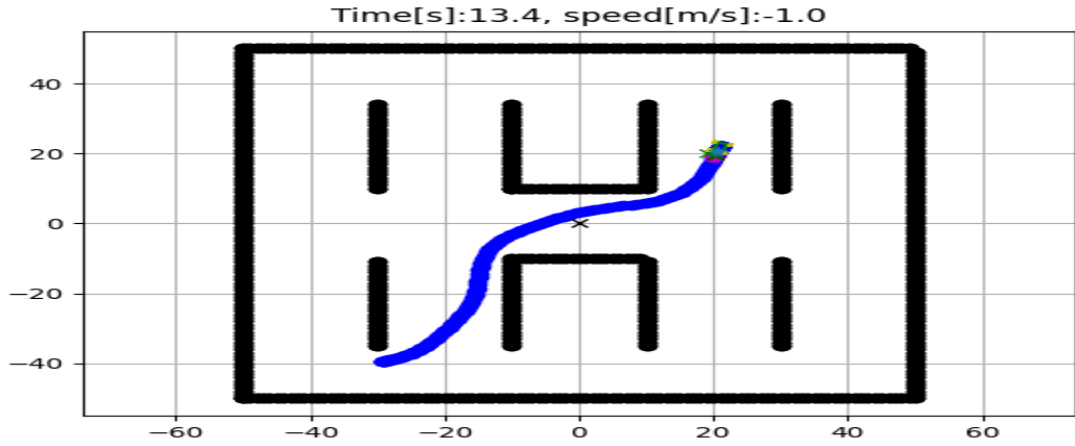


(c) Speed vs time

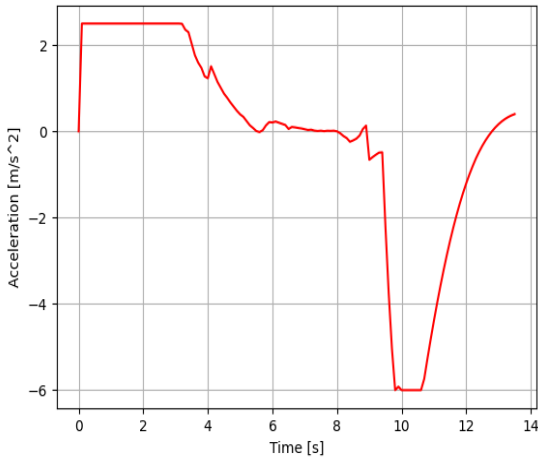


(d) Steering angle vs time

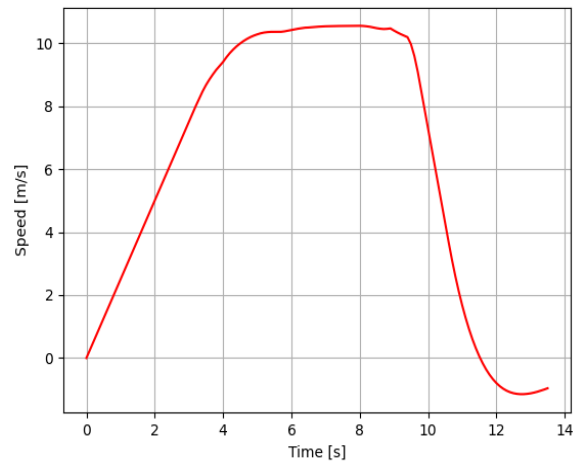
Fig. 5.37 Terminal cost weights low, path from (-20,20) to (40,0)



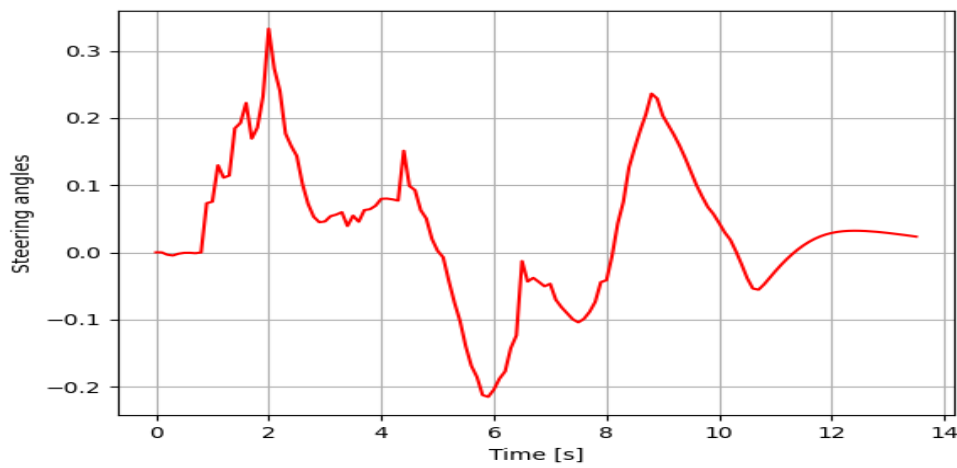
(a) Simulation of path following



(b) Acceleration vs time

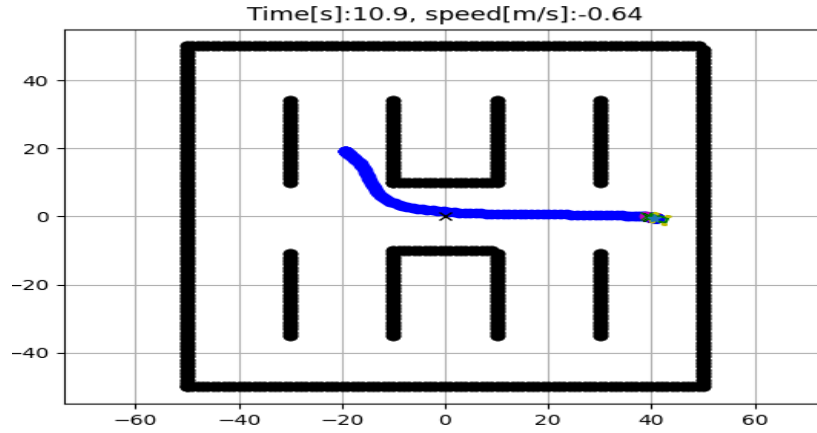


(c) Speed vs time

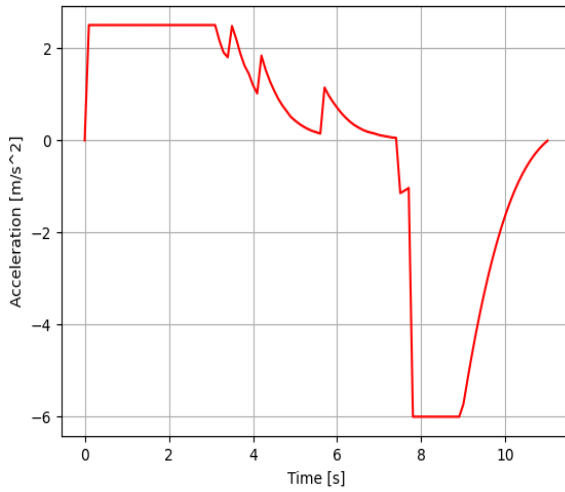


(d) Steering angle vs time

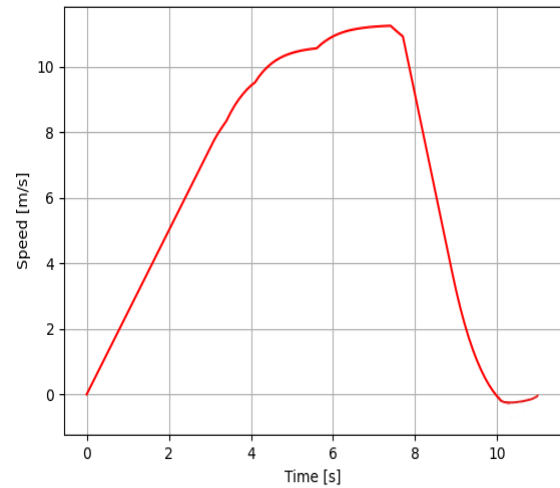
Fig. 5.38 Terminal cost weights low, path from (-30,-40) to (20,20)



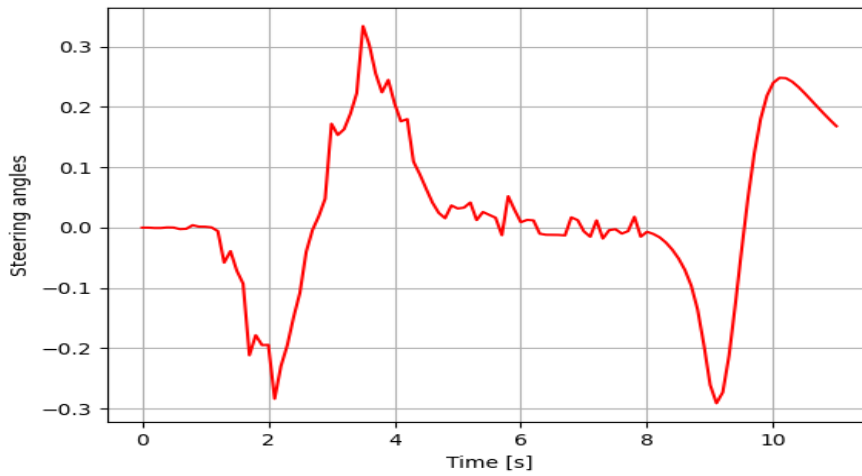
(a) Simulation of path following



(b) Acceleration vs time

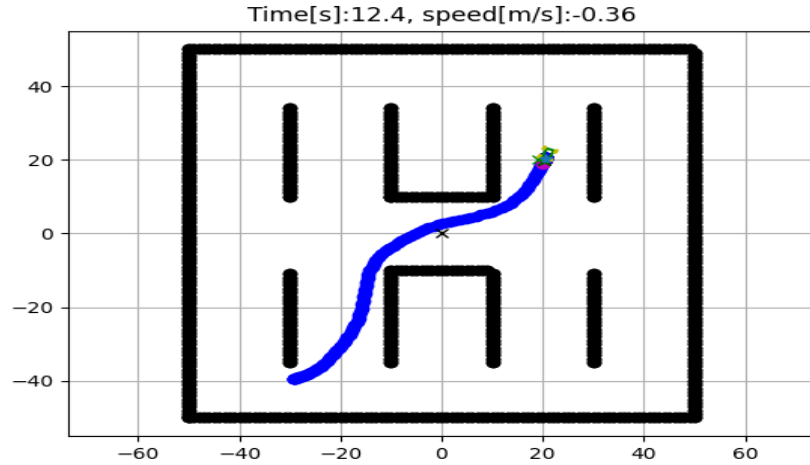


(c) Speed vs time

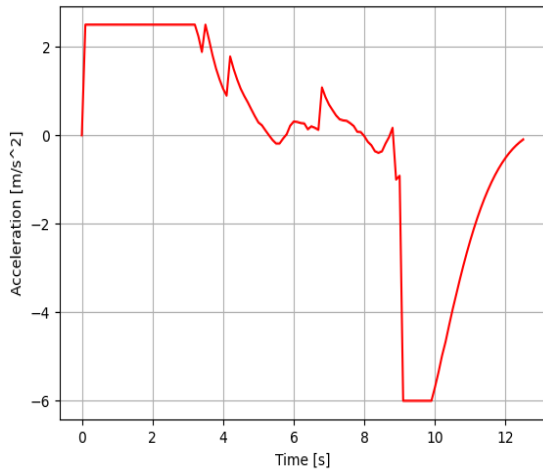


(d) Steering angle vs time

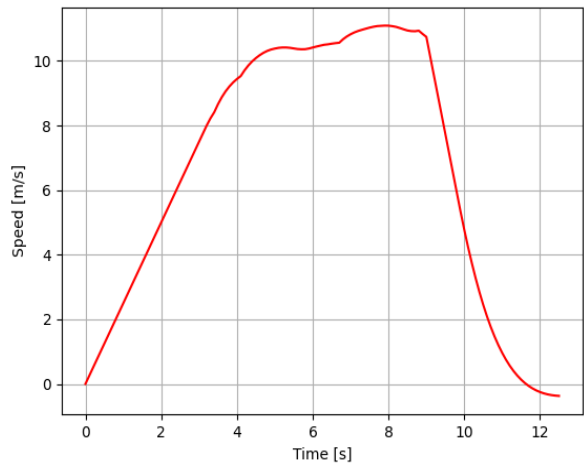
Fig. 5.39 Terminal cost weights high, path from (-20,20) to (40,0)



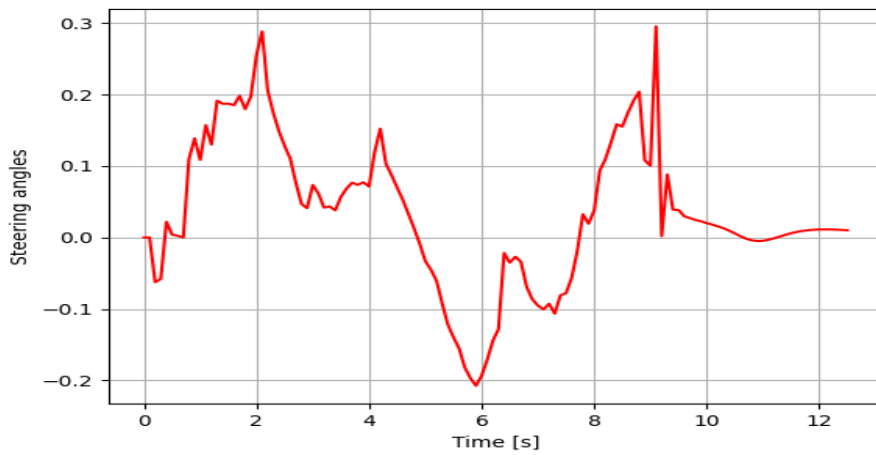
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

Fig. 5.40 Terminal cost weights high, path from (-30,-40) to (20,20)

From the simulation result, the choice of terminal cost weights (D) also has a significant impact on the behavior of the controller during the simulation of self-driving cars. The terminal cost weights determine the importance of the final state of the car compared to the intermediate states. A high value of D emphasizes the importance of the final state and penalizes deviations from it, while a low value of D places less emphasis on the final state.

In simulations with low terminal cost weights, the controller focuses less on achieving the desired final state and more on achieving intermediate states. This can result in more aggressive and dynamic driving behavior as the controller prioritizes reaching intermediate states quickly.

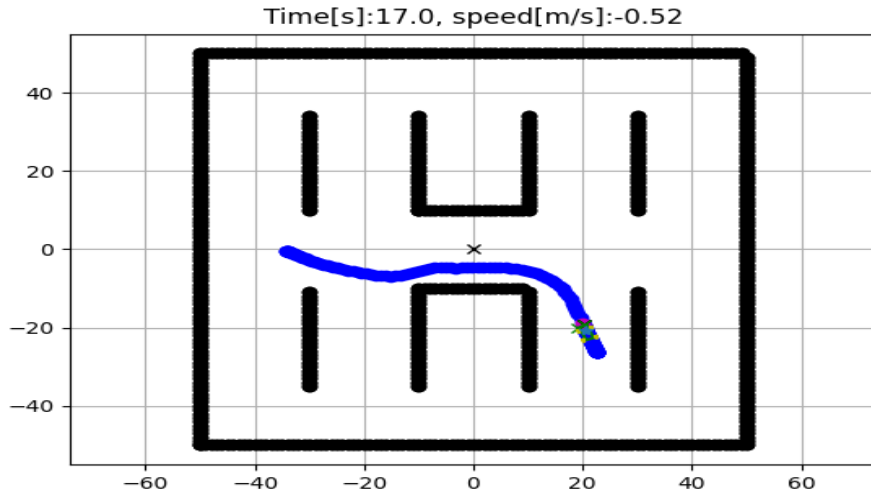
3- Horizon Length

The effect of horizon length on the behavior of the self-driving car controller during simulation is an important aspect to consider. The horizon length represents the number of future steps the controller considers while planning a trajectory for the car. The longer the horizon length, the more future steps the controller considers, resulting in a smoother and more optimal trajectory. However, this comes at the cost of increased computational complexity and longer planning times. The effect of the horizon length value was tested in two different paths.

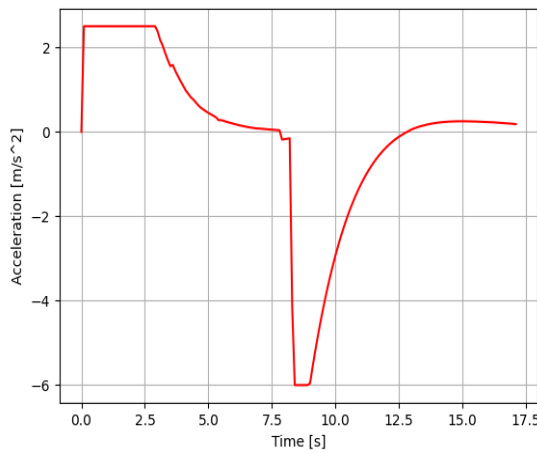
Fig. 5.41 and Fig. 5.42 show the response when the low value used to set the horizon length to 3.

Fig. 5.43 and Fig. 5.44 shows the response when a high value of 20 was used.

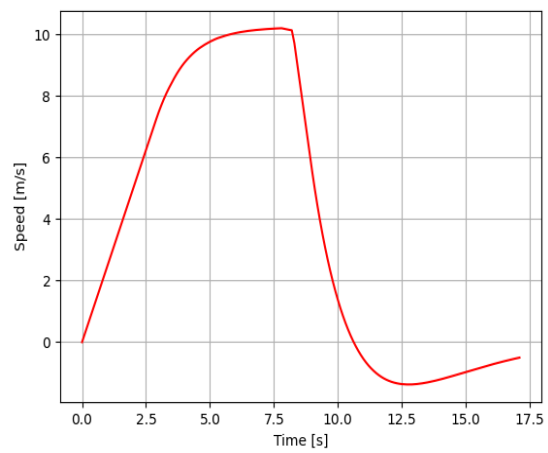
Fig. 5.45 and Fig. 5.46 shows the response when suitable value (8 to 12) was used.



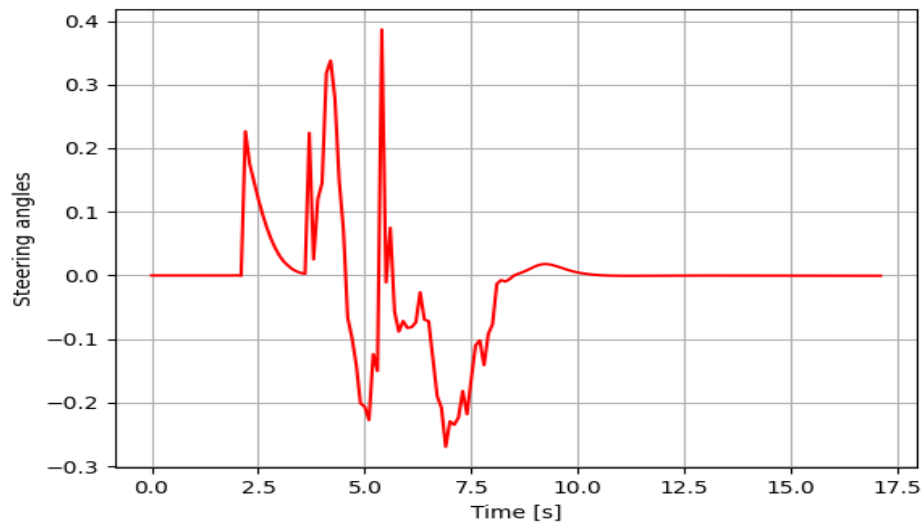
(a) Simulation of path following



(b) Acceleration vs time

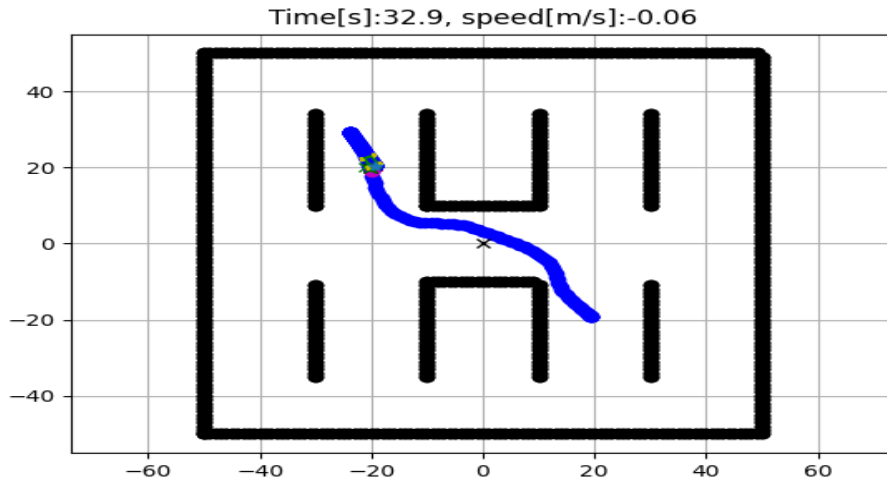


(c) Speed vs time

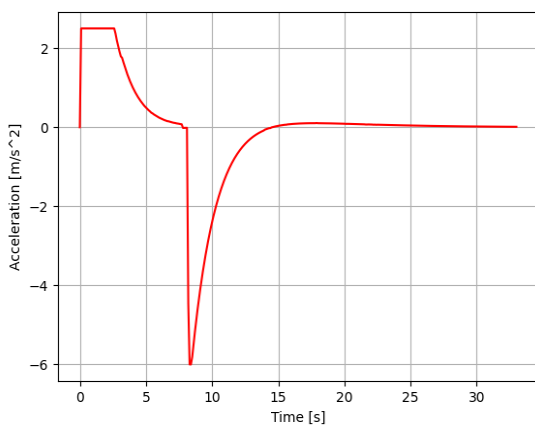


(d) Steering angle vs time

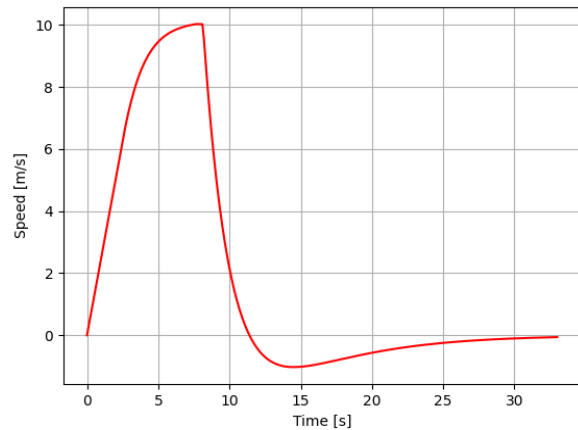
Fig. 5.41 Low horizon length effect, path from (-35,0) to (20, -20)



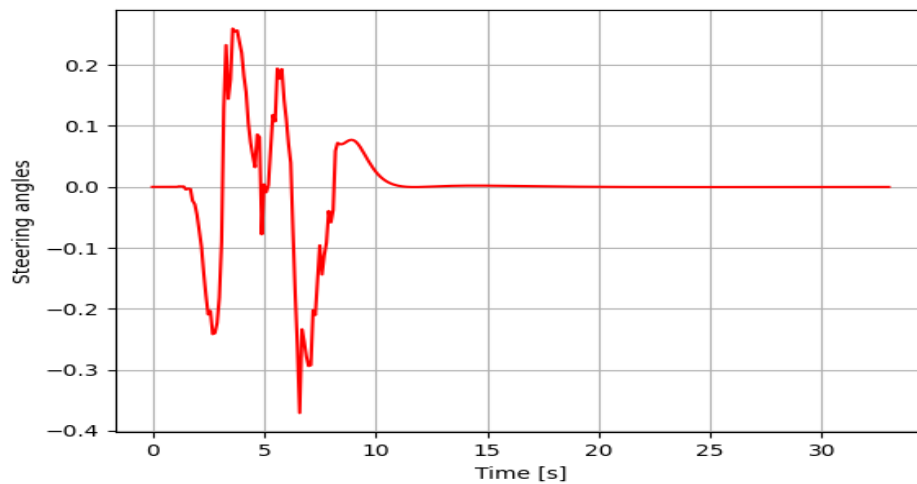
(a) Simulation of path following



(b) Acceleration vs time

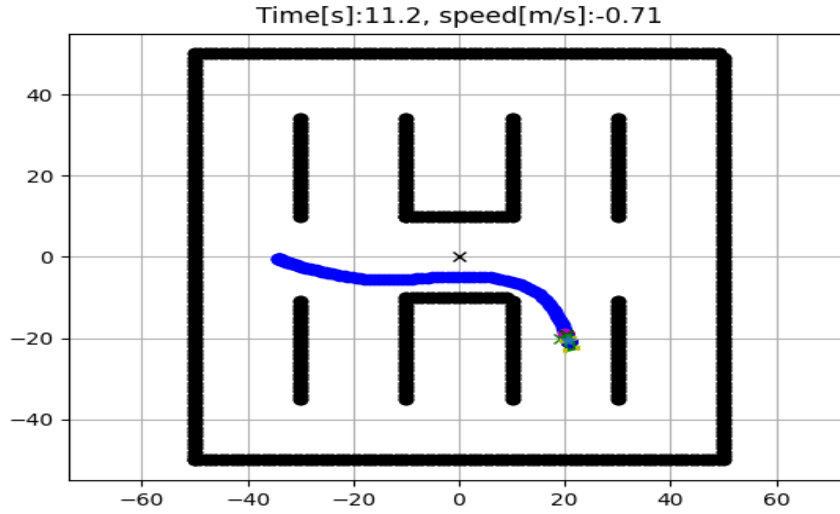


(c) Speed vs time

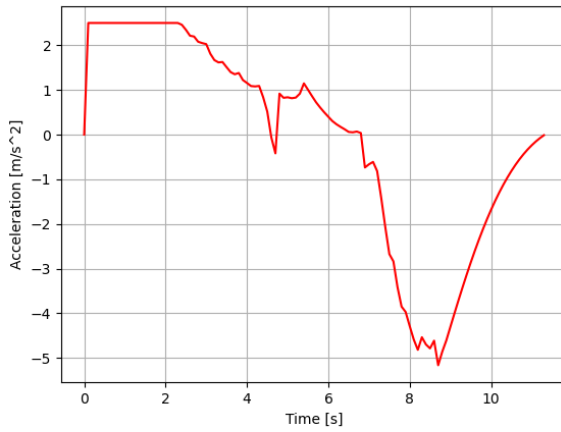


(d) Steering angle vs time

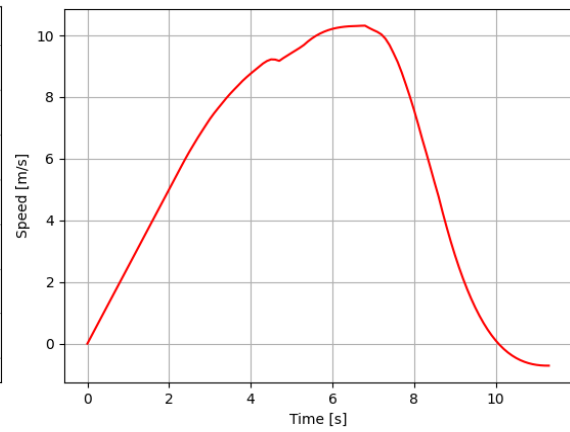
Fig. 5.42 Low horizon length effect, path from (20, -20) to (-20,20)



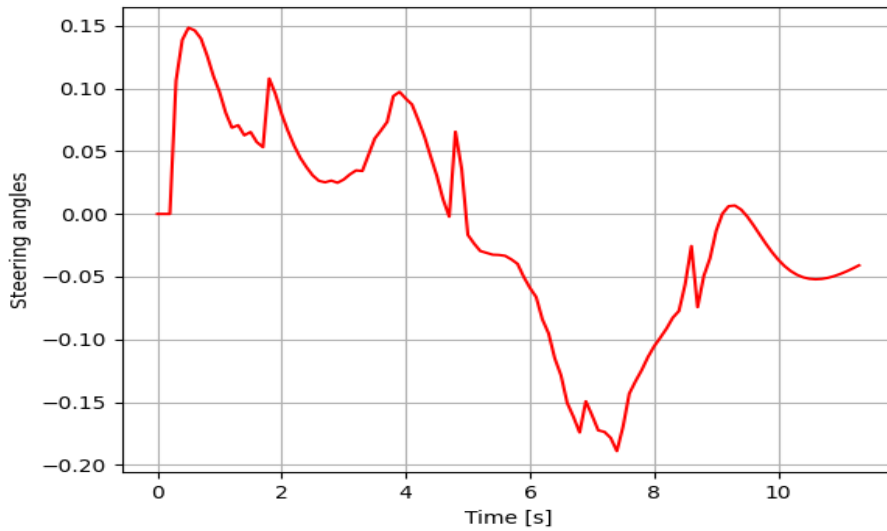
(a) Simulation of path following



(b) Acceleration vs time

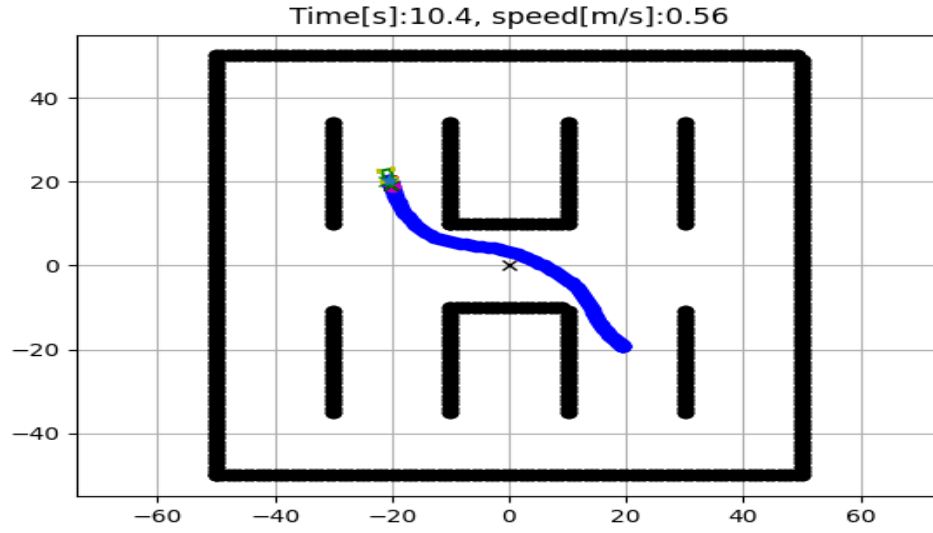


(c) Speed vs time

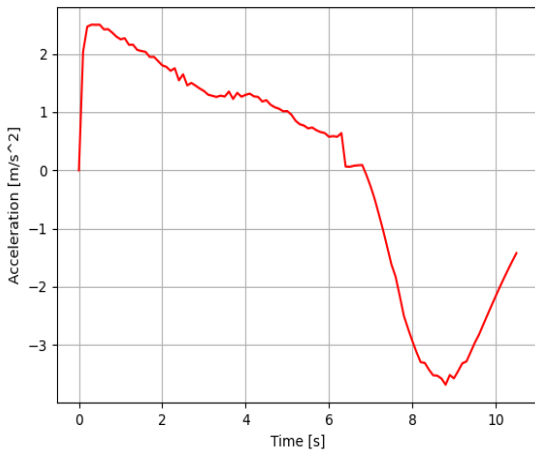


(d) Steering angle vs time

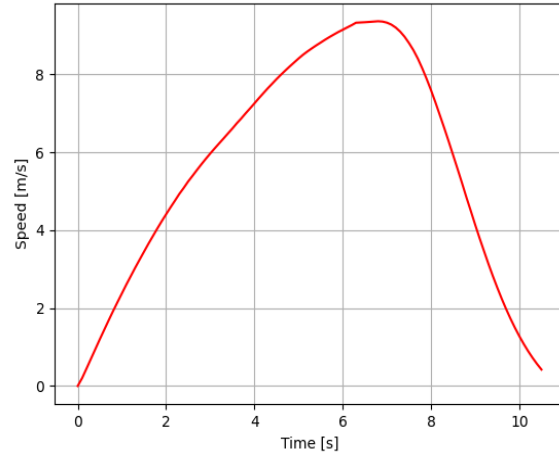
Fig. 5.43 High horizon length effect, path from (-35,0) to (20,-20)



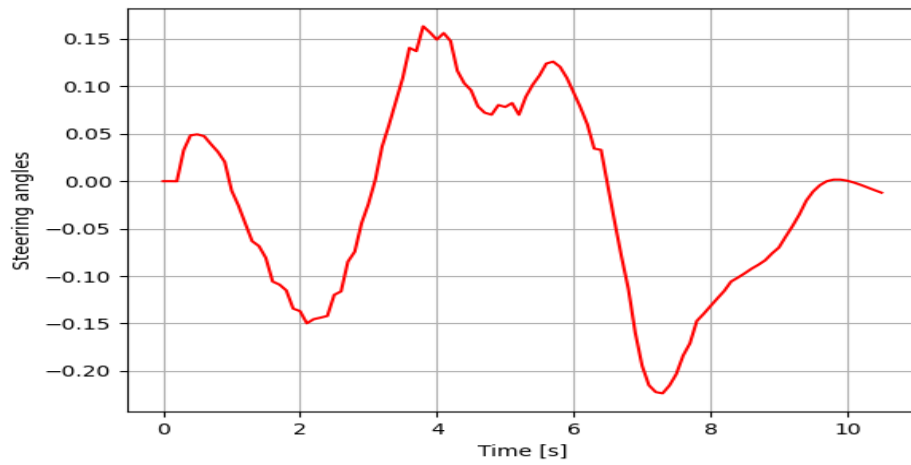
(a) Simulation of path following



(b) Acceleration vs time

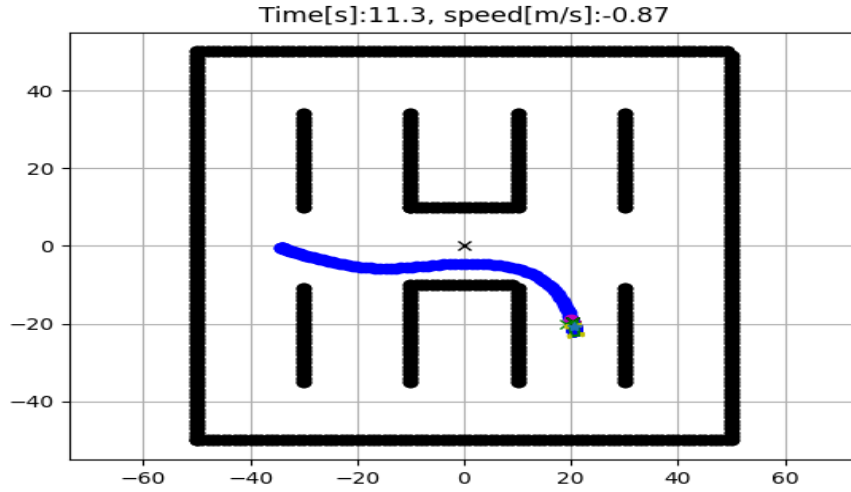


(c) Speed vs time

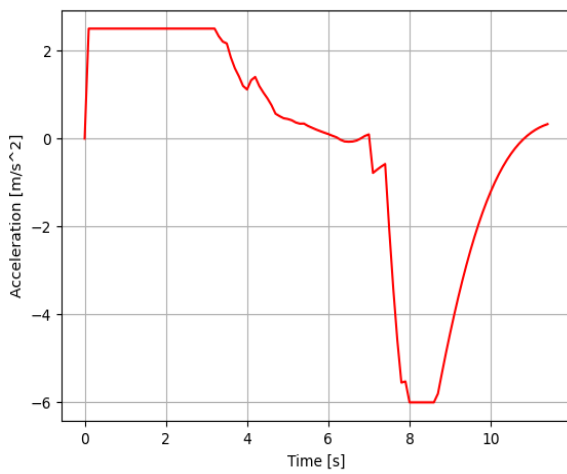


(d) Steering angle vs time

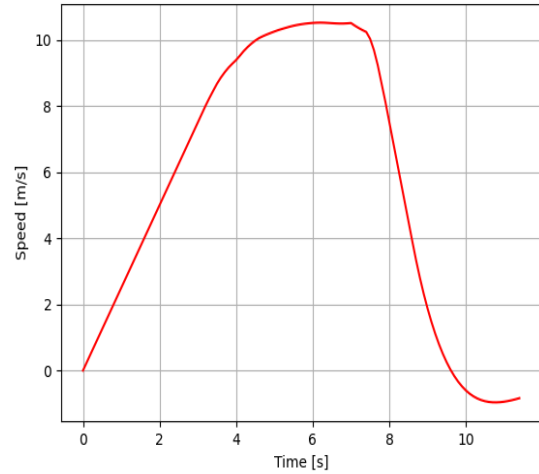
Fig. 5.44 High horizon length effect, path from (20, -20) to (-20,20)



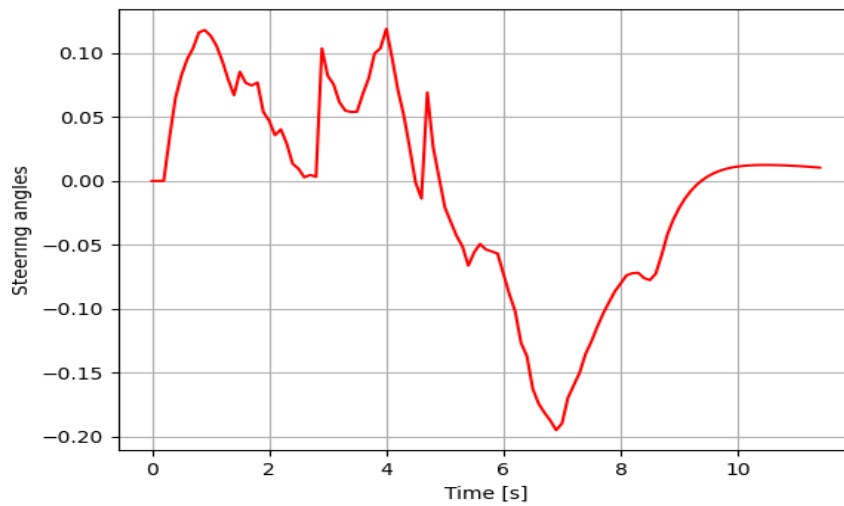
(a) Simulation of path following



(b) Acceleration vs time

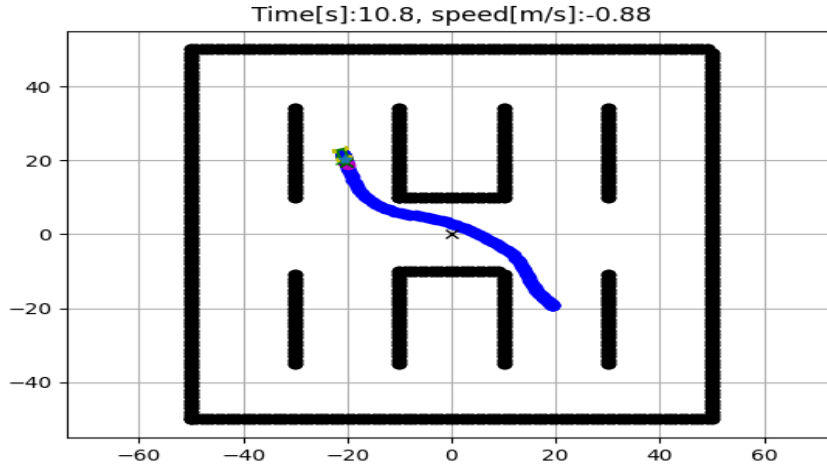


(c) Speed vs time

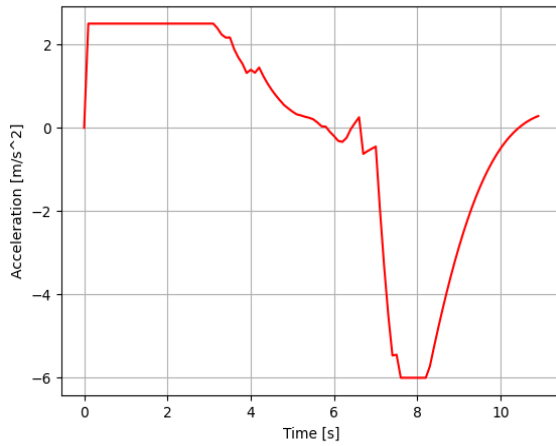


(d) Steering angle vs time

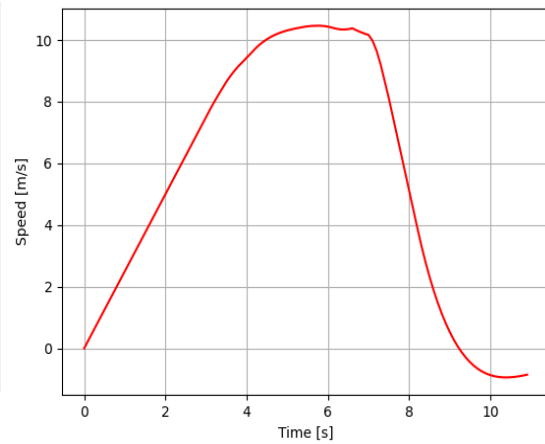
Fig. 5.45 Suitable horizon length effect, path from (-35,0) to (20, -20)



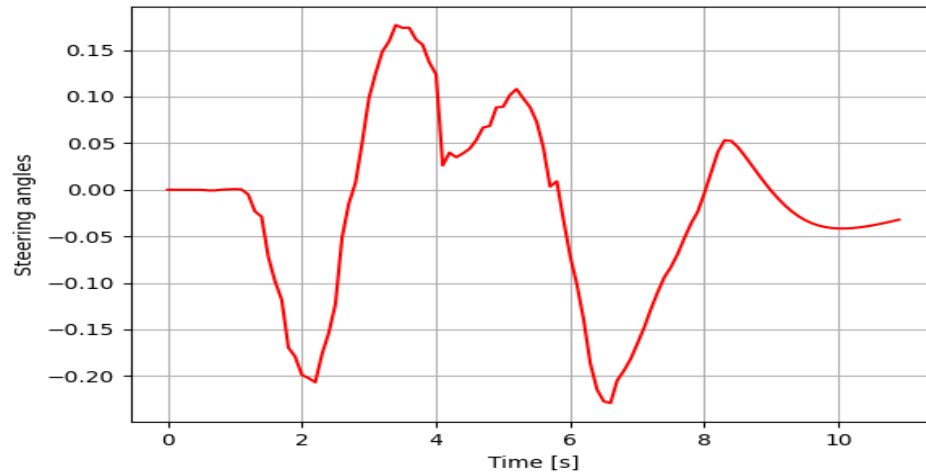
(a) Simulation of path following



(b) Acceleration vs time



(c) Speed vs time



(d) Steering angle vs time

Fig. 5.46 Suitable horizon length effect, path from (20, -20) to (-20,20)

Simulation results show that decreasing the horizon length results in a more responsive driving style, as the controller is only planning a few steps ahead and can react more quickly to changes in the environment. However, this may result in less optimal trajectories with more abrupt changes in control inputs, as the controller has less time to plan ahead. Additionally, a very low horizon length may not allow the MPC algorithm to consider the constraints on the control inputs over a long enough time interval, which is important for the safety and robustness of the control system.

Additionally, increasing the horizon length mean there is an increase in the computational effort required leads to smoother driving behavior with fewer abrupt changes in control inputs. This is because the controller has a better understanding of the future state of the car and can plan ahead to avoid sudden changes in direction or speed, allows the MPC algorithm to consider the constraints on the control inputs over a long time interval, which is important for the safety and robustness of the control system. However, this also results in a slower response time to unexpected changes in the environment, as the controller has already planned several steps ahead and may need to replan the trajectory.

Based on the simulation results and analysis, a horizon length of 8 to 12 was found to be an appropriate range for the self-driving car controller. This horizon length balances the trade-off between computational cost, performance, and driving comfort. The controller was able to accurately predict the car's future state and control inputs over a sufficient time interval, considering the constraints on the control inputs over a long enough time interval, which is crucial for safety and robustness of the control system.

Numerous studies are related to the proposed work. One such study, presented in [17], proposes a path planner based on Model Predictive Control (MPC) that incorporates a convex relaxation approach for both lane change and lane keeping maneuvers. The planner also employs a lane-associated potential field to generate natural and comfortable trajectories. However, it differs from the current work, which utilizes the A* algorithm for global path planning, the Potential field algorithm for local path planning, and N-MPC for motion planning. Despite the differences, both works demonstrate the effectiveness of their respective approaches in generating safe and comfortable paths for autonomous cars in various driving scenarios.

Chapter Six: Conclusions and Recommendations

6.1 Conclusions

The aim of this study was to develop an intelligent control system that could support motion planning in self-driving cars. The system relies on various path planning algorithms and a predictive controller to achieve motion planning for an autonomous car. Global path planning was done using the A* algorithm, while local path planning was done using the Potential field algorithm. These algorithms provided a reference trajectory for the car while updating the path in response to moving obstacles. Moreover, a predictive controller was used to optimize the car's trajectory, ensuring safe and efficient driving by computing a set of future states based on a given set of control input values.

The system model utilized the kinematic model of a non-holonomic car to approximate the car's state and input. The neural network was used to predict the required weight for the control inputs of self-driving cars, while the MPC algorithm was used to optimize these control inputs over a specified time horizon. The MPC algorithm achieved this by minimizing the cost function and satisfying the constraints. The results showed that the proposed approach was successful in navigating through obstacles in both static and dynamic environments. The study also demonstrated the importance of carefully selecting the cost function weight and horizon length for the predictive controller to achieve optimal performance. This research provides valuable insights into the use of path planning algorithms and predictive controllers for motion planning in autonomous cars.

However, the proposed approach has some limitations, and further research is needed to optimize the proposed approach and integrate it with

other components of self-driving cars, such as perception, localization, and decision-making. Overall, this thesis has made significant contributions to the field of autonomous driving and has paved the way for future innovations in this area.

6.2 Future Work

The following recommendations can be considered for future work:

- Exploring the use of deep learning for path planning. This can be achieved by training a neural network to learn the optimal path in various environments.
- Integrating real-time sensor data into the system by incorporating sensors such as LIDAR, cameras, or radar. These sensors can detect obstacles and update the car's path and control inputs in real-time.
- Incorporating machine learning techniques to optimize the car's control inputs based on real-time sensor data.
- Evaluating the system's performance in more complex environments to identify areas where the system may need improvement or optimization for different scenarios.
- Improving communication between autonomous cars to ensure effective communication as more autonomous cars are introduced onto the roads.

References

- [1] N. Highway Traffic Safety Administration and U. Department of Transportation, “Early Estimate of Motor Vehicle Traffic Fatalities for the First 9 Months (January–September) of 2021,” 2021.
- [2] “What Percentage of Car Accidents Are Caused by Human Error? | Pittsburgh Law Blog.” <https://www.cbmclaw.com/what-percentage-of-car-accidents-are-caused-by-human-error/> (accessed Jun. 16, 2022).
- [3] “SAE Levels of Driving Automation™ Refined for Clarity and International Audience.” <https://www.sae.org/blog/sae-j3016-update> (accessed Jun. 14, 2022).
- [4] Y. Nishio, K. Nonaka, and K. Sekiguchi, Moving Obstacle Avoidance Control by Fuzzy Potential Method and Model Predictive Control. 2017. doi: 10.0/Linux-x86_64.
- [5] M. Nolte, M. Rose, T. Stolte, and M. Maurer, “Model Predictive Control Based Trajectory Generation for Autonomous Vehicles-An Architectural Approach,” 2017.
- [6] A. Franco and V. Santos, “Short-term Path Planning with Multiple Moving Obstacle Avoidance based on Adaptive MPC,” 2019.
- [7] D. González, J. Pérez, V. Milanés, and F. Nashashibi, “A Review of Motion Planning Techniques for Automated Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016, doi: 10.1109/TITS.2015.2498841.
- [8] P. Falcone, H. Eric Tseng, F. Borrelli, J. Asgari, and D. Hrovat, “MPC-based yaw and lateral stabilisation via active front steering and braking,” in *Vehicle System Dynamics*, 2008, pp. 611–628. doi: 10.1080/00423110802018297.

- [9] V. T. Minh, “Nonlinear Model Predictive Controller and Feasible Path Planning for Autonomous Robots,” *Open Computer Science*, vol. 6, no. 1, pp. 178–186, 2016, doi: 10.1515/comp-2016-0015.
- [10] A. Koga, H. Okuda, Yuichi Tazaki, and et al., “Realization of Different Driving Characteristics for Autonomous Vehicle by Using Model Predictive Control*,” *IEEE Intelligent Vehicles Symposium*, 2016.
- [11] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles,” Apr. 2016, [Online].
- [12] Christian Gotte, Martin Keller, Till Nattermann, Alois Seewald, and et al., “A Real-Time Capable Model Predictive Approach to Lateral Vehicle Guidance,” *IEEE*, 2016.
- [13] M. Bojarski et al., “End to End Learning for Self-Driving Cars,” Apr. 2016, [Online]. Available: <http://arxiv.org/abs/1604.07316>
- [14] J. Rios-Torres and A. A. Malikopoulos, “A Survey on the Coordination of Connected and Automated Vehicles at Intersections and Merging at Highway On-Ramps,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1066–1077, May 2017.
- [15] G. Bresson, Z. Alsayed, L. Yu, S. Glaser, and L. Yu, “Simultaneous Localization And Mapping: A Survey of Current Trends in Autonomous Driving,” vol. XX, 2017, doi: 10.1109/TIV.2017.2749181i.
- [16] C. Bila, F. Sivrikaya, M. A. Khan, and S. Albayrak, “Vehicles of the Future: A Survey of Research on Safety Issues,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1046–1065, May 2017, doi: 10.1109/TITS.2016.2600300.
- [17] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, “Path Planning for Autonomous Vehicles using Model Predictive Control,” *IEEE*, 2017.

- [18] R. Guidolini, A. F. de Souza, F. Mutz, and Badue Claudine, “Neural-Based Model Predictive Control for Tackling Steering Delays of Autonomous Cars*,” IEEE, 2017.
- [19] S. di Cairano and I. v Kolmanovsky, “Real-time optimization and model predictive control for aerospace and automotive applications,” MITSUBISHI ELECTRIC RESEARCH LABORATORIES, 2018, [Online]. Available: <http://www.merl.com>
- [20] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, “Information Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving,” IEEE Xplore, 2018.
- [21] C. Marina Martinez, M. Heucke, F. Y. Wang, B. Gao, and D. Cao, “Driving Style Recognition for Intelligent Vehicle Control and Advanced Driver Assistance: A Survey,” IEEE Transactions on Intelligent Transportation Systems, , pp. 666–676, Mar. 2018.
- [22] M. v. Smolyakov, A. I. Frolov, V. N. Volkov, and I. v. Stelmashchuk, “Self-Driving Car Steering Angle Prediction Based on Deep Neural Network An Example of CarND Udacity Simulator,” in IEEE 12th International Conference on Application of Information and Communication Technologies, AICT 2018 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Oct. 2018.
- [23] J. Wang, J. Liu, and N. Kato, “Networking and Communications in Autonomous Driving: A Survey,” IEEE Communications Surveys and Tutorials, vol. 21, no. 2, pp. 1243–1274, Apr. 2019.
- [24] Y. Wang, D. Zhang, Y. Liu, B. Dai, and L. H. Lee, “Enhancing transportation systems via deep learning: A survey,” Transportation Research Part C: Emerging Technologies, vol. 99. Elsevier Ltd, pp. 144–163, Feb. 01, 2019. doi: 10.1016/j.trc.2018.12.004.

- [25] J. Gwak, J. Jung, R. D. Oh, M. Park, M. A. K. Rakhimov, and J. Ahn, "A review of intelligent self-driving vehicle software research," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 11, pp. 5299–5320, Nov. 2019, doi: 10.3837/tiis.2019.11.002.
- [26] A. Reda, A. Bouzid, and J. Vásárhelyi, "Model Predictive Control for Automated Vehicle Steering," *Acta Polytechnica Hungarica*, vol. 17, no. 7, pp. 2020–163.
- [27] K. Muhammad, A. Ullah, J. Lloret, J. del Ser, and V. H. C. de Albuquerque, "Deep Learning for Safe Autonomous Driving: Current Challenges and Future Directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, Jul. 2021, doi: 10.1109/TITS.2020.3032227.
- [28] Y. Xu, W. Tang, B. Chen, L. Qiu, and R. Yang, "A model predictive control with preview-follower theory algorithm for trajectory tracking control in autonomous vehicles," *Symmetry (Basel)*, vol. 13, no. 3, pp. 1–16, Mar. 2021, doi: 10.3390/sym13030381.
- [29] S. Kolachalama and H. Malik, "Intelligent vehicle drive mode which predicts the driver behavior vector to augment the engine performance in real-time," *Data-Centric Engineering*, vol. 3, no. 10, Apr. 2022, doi: 10.1017/dce.2022.15.
- [30] J. L. Vazquez, A. Liniger, W. Schwarting, D. Rus, and L. van Gool, "Deep Interactive Motion Prediction and Planning: Playing Games with Motion Prediction Models," Apr. 2022, [Online]. Available: <http://arxiv.org/abs/2204.02392>
- [31] X. Jin, Z. Yan, G. Yin, S. Li, and C. Wei, "An Adaptive Motion Planning Technique for On-Road Autonomous Driving," *IEEE Access*, vol. 9, pp. 2655–2664, 2021, doi: 10.1109/ACCESS.2020.3047385.

- [32] S. N. Vassilyev, A. Y. Kelina, Y. I. Kudinov, and F. F. Pashchenko, “Intelligent Control Systems,” in *Procedia Computer Science*, Elsevier B.V., 2017, pp. 623–628. doi: 10.1016/j.procs.2017.01.088.
- [33] K. M. Passino, “Intelligent Control: An Overview of Techniques *.”, 2015.
- [34] P. Antsaklis, “Intelligent Control.” Jan. 01, 1994. Accessed: Jan. 02, 2023.
- [35] S. Azadi, R. Kazemi, and H. R. Nedamani, “Trajectory planning of tractor semitrailers,” *Vehicle Dynamics and Control*, pp. 429–478, Jan. 2021, doi: 10.1016/B978-0-323-85659-1.00010-0.
- [36] H. Liu, *Robot Systems for Rail Transit Applications*. Elsevier, 2020. doi: 10.1016/B978-0-12-822968-2.01001-9.
- [37] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H. M. Tai, “Autonomous local path planning for a mobile robot using a genetic algorithm,” *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, vol. 2, pp. 1338–1345, 2004, doi: 10.1109/CEC.2004.1331052.
- [38] “An Introduction to Dijkstra’s Algorithm: Theory and Python Implementation | by Andreas Soularidis | Python in Plain English.” <https://python.plainenglish.io/dijkstras-algorithm-theory-and-python-implementation-c1135402c321> (accessed Apr. 08, 2023).
- [39] S. K. Debnath et al., “A review on graph search algorithms for optimal energy efficient path planning for an unmanned air vehicle,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 15, no. 2, pp. 743–750, Aug. 2019, doi: 10.11591/ijeecs.v15.i2.pp743-749.

- [40] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, “A Survey of Path Planning Algorithms for Mobile Robots,” *Vehicles*, vol. 3, no. 3, pp. 448–468, Aug. 2021, doi: 10.3390/vehicles3030027.
- [41] I. M. Zidane and K. Ibrahim, “Wavefront and a-star algorithms for mobile robot path planning,” *Advances in Intelligent Systems and Computing*, vol. 639, pp. 69–80, 2018, doi: 10.1007/978-3-319-64861-3_7.
- [42] A. Stentz, “Optimal and Efficient Path Planning for Unknown and Dynamic Environments,” 1993.
- [43] Z. Zhang, J. Wu, J. Dai, and C. He, “A Novel Real-Time Penetration Path Planning Algorithm for Stealth UAV in 3D Complex Dynamic Environment,” *IEEE Access*, vol. 8, pp. 122757–122771, 2020, doi: 10.1109/ACCESS.2020.3007496.
- [44] S. Singh, R. Simmons, T. Smith, A. Stents, and V. Verma, “Recent Progress in Local and Global Traversability for Planetary Rovers,” *IEEE*, 2000.
- [45] M. Pivtoraiko and A. Kelly, “EFFICIENT CONSTRAINED PATH PLANNING VIA SEARCH IN STATE LATTICES”, 2005.
- [46] W. Nghah WAJ, “A Simple Local Path Planning Algorithm for Autonomous Mobile Robots”, 2011.
- [47] F. Arambula Cosío and M. A. Padilla Castañeda, “Autonomous robot navigation using adaptive potential fields,” *Math Comput Model*, vol. 40, no. 9–10, pp. 1141–1156, 2004, doi: 10.1016/j.mcm.2004.05.001.
- [48] “Model Predictive Control 20.1 OVERVIEW OF MODEL PREDICTIVE CONTROL From online version of book by Seborg et al. (2011) on ‘Process Dynamics and Control.’”

- [49] K. P and B. R, "Design of PID and Model Predictive Controller for Three Phase Flow (Crude Oil+Water+Air) through Helical Coil and Control Valve in Series," *International Journal of Chemical Sciences*, vol. 15, no. 1, p. 114, Mar. 2017, Accessed: Apr. 08, 2023.
- [50] P. E. Orukpe, "MODEL PREDICTIVE CONTROL FUNDAMENTALS," *Nigerian Journal of Technology (NIJOTECH)*, vol. 31, no. 2, pp. 139–148, 2012.
- [51] E. F. Camacho and C. Bordons, *Model Predictive Control*. Springer, 2013.
- [52] I. Aşar, "“MODEL PREDICTIVE CONTROL (MPC) PERFORMANCE FOR CONTROLLING REACTION SYSTEMS”" A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF MIDDLE EAST TECHNICAL UNIVERSITY," 2004.
- [53] A. Khalaf MAlmaliki, N. Abdul Wahab, and S. AL-Haddad, "Model-based Predictive Control of a Tower Crane," *MACE Technical Journal (MTJ)*, 2019.
- [54] M. F. Manzoor and Q. Wu, "Control and Obstacle Avoidance of Wheeled Mobile Robot," in *Proceedings - 7th International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2015*, Institute of Electrical and Electronics Engineers Inc., Oct. 2015, pp. 235–240. doi: 10.1109/CICSyN.2015.48.
- [55] S. Lek and J. F. Guégan, "Artificial neural networks as a tool in ecological modelling, an introduction," 1999.
- [56] E. Castillo, "Functional Networks Related papers Functional Networks: A New Network-Based Methodology Enrique Castillo Functional Networks," 1998.

- [57] K. (Kevin N.) Gurney, An introduction to neural networks. UCL Press, 1997.
- [58] M. T. Hagan, H. B. Demuth, and O. De Jesus, “AN INTRODUCTION TO THE USE OF NEURAL NETWORKS IN CONTROL SYSTEMS,” *International Journal of Robust and Nonlinear Control: IFAC-Affiliated*, 2002.
- [59] A. Vasičkaninová, M. Bakosova, and M. Bakošová, “Neural Network Predictive Control Of A Chemical Reactor,” *Acta Chimica Slovaca*, vol. 2, no. 2, pp. 21–36, 2009, doi: 10.7148/2009-0563-0569.
- [60] J. Feng, X. He, Q. Teng, C. Ren, H. Chen, and Y. Li, “Reconstruction of porous media from extremely limited information using conditional generative adversarial networks,” Sep. 2019.
- [61] “Derivative of Neural Activation Function | by Yash Garg | Medium.” <https://yashgarg1232.medium.com/derivative-of-neural-activation-function-64e9e825b67> (accessed Dec. 06, 2022).
- [62] L. Tang, F. Yan, B. Zou, K. Wang, and C. Lv, “An improved kinematic model predictive control for high-speed path tracking of autonomous vehicles,” *IEEE Access*, vol. 8, pp. 51400–51413, 2020, doi: 10.1109/ACCESS.2020.2980188.
- [63] P. R. B. Monasterios and P. A. Trodden, “Model predictive control of linear systems with preview information: Feasibility, stability, and inherent robustness,” *IEEE Trans Automat Contr*, Sep. 2019
- [64] “Convex optimization explained: Concepts & Examples - Data Analytics.” <https://vitalflux.com/convex-optimization-explained-concepts-examples/> (accessed Dec. 02, 2022).

الخلاصة

بيئة حركة المرور على الطرق متغيرة للغاية ولا يمكن التنبؤ بها. قد تواجه السيارات ذاتية القيادة التي تعمل في مثل هذه البيئات سيناريوهات حرجة غير متوقعة، حيث يزداد خطر وقوع حادث بسرعة مقارنة بمواقف القيادة العادية. قد تنشأ هذه السيناريوهات بسبب سلوك غير متوقع من مستخدمي الطريق الآخرين أو ظهور عوائق على الطريق. في مثل هذه الظروف الحرجة، يكون الهدف الأساسي للتحكم في حركة السيارة هو تقليل خطر وقوع حادث وشيك.

الغرض من هذه الدراسة هو تطوير نظام يمكن أن يساعد في منع الحوادث في بيئات المرور على الطرق التي لا يمكن التنبؤ بها والمتغيرة من خلال معالجة مشكلة تخطيط الحركة والتحكم فيها في المواقف الحرجة للسيارات المستقلة. يولد النظام المسارات المثلى ومدخلات التحكم للسيارة لاتباعها مع تجنب العوائق واتباع مركز المسار بشكل متوقع. لتحقيق هذا الهدف، تم تقديم تقنية تخطيط الحركة للسيارات ذاتية القيادة.

تعتمد تقنية تخطيط الحركة المستخدمة في هذه الدراسة على خوارزميات (potential field , A*) لتخطيط المسار، مع وحدة تحكم ذكية تتكون من prediction المدعوم بتقنية neural network . تتنبأ model predictive controller النموذجية بمستقبل السيارة لأفق زمني محدود باستخدام نموذج رياضي للسيارة. تستخدم وحدة التحكم نموذج دراجة حركي خطي ومنفصل كنموذج سيارة داخلي. يتم استخدام مسار الإستراتيجية A* لأنه أداة قوية لحل مشاكل تحديد المسار نظرًا لما يتمتع به من أمثلية وكفاءة وقبول ومرونة. ويتم استخدام potential field لتخطيط المسار في بيئة بها عوائق بسبب بساطتها وأمانها وتكلفة حسابية منخفضة. مدخلات التحكم هي زاوية توجيه السيارة والتسارع.

تعمل model predictive controller في النموذج على حل مشكلة التحسين باعتبارها مشكلة برمجة تربيعية تقلل من دالة التكلفة مع تلبية مجموعة من القيود. تُستخدم neural network لضبط معدل تغيير قيمة زاوية التوجيه، ويعد تعديل معدل زاوية التوجيه جزءًا مهمًا من تحسين أداء السيارة ذاتية القيادة وضمان سلامتها وموثوقيتها على الطريق. بدون تعديل معدل زاوية التوجيه، قد تتأثر قدرة السيارة على الدوران الدقيق والتوقف بدقة، مما قد يؤدي إلى زيادة مخاطر وقوع الحوادث وتقليل كفاءة استهلاك الوقود. تتضمن وظيفة التكلفة مجموعة من الأهداف، بما في ذلك الأخطاء في الحالات المرغوبة والحالية، والمدخلات، ومعدل تغيير المدخلات، لتوجيه السيارة ذاتية القيادة بعيدًا

عن المناطق عالية التكلفة. تحدد وحدة اتخاذ القرار مسار العمل التالي للسيارة، بعد تحديد المناورة اللاحقة يتم إنشاء ملف تعريف السرعة ومرجع مركز المسار لتتبع السيارة ذاتية القيادة.

تم حل مشكلة البرمجة التربيعية باستخدام python convex optimisation في أداة التحسين، بوقت عينة يبلغ 0.1s. يتم ضبط معلمات التحكم، بما في ذلك أوزان دالة التكلفة وطول الأفق، لجعل المسار آمنًا ومريحًا. يتراوح طول الأفق المحدد من 8m إلى 12m، وفيه تضمن وحدة التحكم أن السيارة تتبع المسار المقصود مع تجنب العوائق.

تم تحقيق جميع النتائج ضمن قيود السيارة المحددة والتي هي، أقصى سرعة 15 m/s، أقصى سرعة عكسية 5 m/s، أقصى زاوية توجيه 45°، أقصى معدل توجيه 30°، أقصى تباطؤ 6 m/ S² وأقصى تسارع 2.5 m/ S².



جمهورية العراق
وزارة التعليم العالي و البحث العلمي
جامعة كربلاء
كلية الهندسة
قسم الهندسة الكهربائية والالكترونية

نموذج المسيطر التنبؤي المستند على الشبكة العصبية
لتتبع السيارة ذاتية القيادة

رسالة مقدمة الى مجلس كلية الهندسة / جامعة كربلاء وهي جزء من متطلبات نيل درجة الماجستير
في علوم الهندسة الكهربائية

من قبل:

فاطمه منير عبدالرسول

باشراف :

أ.م.د. احمد عبدالهادي احمد

أ.م.د. حيدر جليل كامل

نيسان - 2023

رمضان - 1444