

MODEL-BASED REINFORCEMENT LEARNING FOR TRUCK TRAILER
ROBOTICS VEHICLE'S TRAJECTORY PLANNING AND CONTROL

by

HAO YAN

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL AND COMPUTER ENGINEERING

2024

Oakland University
Rochester, Michigan

Doctoral Advisory Committee:

Mohamed Zohdy, Ph.D., Chair

Amanpreet Kaur, Ph.D.

Li Li, Ph.D.

Ryan Monroe, Ph.D.

© by Hao Yan, 2024
All rights reserved

To my family and friends
Your unwavering support has transformed
an impulsive beginning
into this momentous achievement
I am forever grateful

ACKNOWLEDGMENTS

First, I would like to show my deepest gratitude to my Ph.D. advisor and committee chair, Professor Mohamed Zohdy, from the Department of Electrical and Computer Engineering. His guidance and continue support have been so important in my Ph.D. academic journey.

I also appreciate all members from my Doctoral Advisory Committee: Dr. Amanpreet Kaur from the Department of Electrical and Computer Engineering, Dr. Li Li from the Department of Mathematics, and Dr. Ryan Monroe from the Department of Mechanical Engineering. They have provided diversified perspectives, valuable feedback, and constructive suggestions, which significantly enhanced the quality and depth of this research. The time, effort, and expertise from them have been crucial throughout the past years.

I would also like to thanks Steven Louis from the Department of Electrical and Computer Engineering, for creating the LaTeX template, which facilitated the formatting of this dissertation. And all other faculty and staff of Oakland University, their support and resources have been crucial in facilitating my research and academic growth.

Lastly, I am so grateful to the unconditional love, patience, and encouragement from my family, throughout my doctoral studies. Their belief in me has been my source of strength and motivation.

To everyone who has contributed to my academic journey, I want to express my heartfelt thanks. This research would not be possible without your support and guidance.

Hao Yan

ABSTRACT

MODEL-BASED REINFORCEMENT LEARNING FOR TRUCK TRAILER ROBOTICS VEHICLE'S TRAJECTORY PLANNING AND CONTROL

by

Hao Yan

Adviser: Mohamed Zohdy, Ph.D.

Advanced Driver Assistance Systems (ADAS) has become increasingly important in the automotive industry, because of customers' need for higher safety and efficiency for transportation. Truck-trailer systems, which occupies a large market share in both commercial and recreational sections, have been facing special challenges for ADAS due to the complex vehicle system dynamics and special skills required for maneuvering, especially in reverse. In the current market products, traditional control methods consisted of path planning and classical control methods have been widely used, however, they often struggle to adapt to the changing scenarios in real-world applications. On the other hand, with the development of Machine Learning, one of its research field, reinforcement learning, shows great potential in robotics and control area. The reinforcement learning algorithm is a semi supervised machine learning approach, and it can provide a flexible solution by enabling systems to learn and improve through experience, potentially conquer the limitations of traditional approaches. This dissertation explores the development of control systems for Truck-trailer Robotics Vehicle (TTRV) system, especially during reverse driving and parking in low-speed, closed, and unstructured environments. The research addressed challenges caused by the complex dynamics of TTRV systems, which is known for its nonlinearity, nonholonomic constraints, and under-actuation behaviours. And this research applied several modified reinforcement

learning approaches to trailer control, as well as traditional control methods, to show the improved ADAS capabilities of reinforcement learning approaches, and effectiveness of satisfying the growing need for intelligent transportation solutions.

This dissertation begins with a review of classical control methods which has been used for truck trailer system, including pole placement, Lyapunov controller, Proportional–Integral–Derivative (PID), Linear Quadratic Regulator (LQR), Model Predictive Control (MPC), and other nonlinear control methods. Multiple path planning approaches such as Dubins paths, Reeds-Shepp paths, and the A* algorithm are also applied and tested to the articulated vehicle. While these methods have shown capability in certain scenarios, changing environments, online calibration, and complex dynamics can make traditional algorithm struggle in real application situations. Moreover, these algorithms require pre-design and fine-tuning from engineers, which limits their application and self-evolution in real-world situations. All these problems slow down the popularity for advanced TTRV control systems in nowadays fast growing market.

To overcome these limitations mentioned above, an end-to-end Deep Reinforcement Learning (DRL) network solution for TTRV reverse autonomous control is proposed in this dissertation. The study compares several DRL algorithms with different computational complexities and performances, including both on-policy and off-policy approaches, all of them showed the effective of DRL in solving the trailer control problem.

Moreover, a hybrid model-based reinforcement learning approach is developed in this research, which integrates classical path planning techniques into the DRL algorithm to improve sample efficiency and performance. The proposed algorithm is tested in several obstacle avoidance simulations, demonstrating its ability to handle target chasing, obstacle avoidance, and passenger comfort simultaneously. Also, an Imitation Inspired Probabilistic Ensembles with Trajectory Sampling (IIPETS) algorithm is developed on the base of Probabilistic Ensembles with Trajectory Sampling (PETS) algorithm, this model

based approach significantly improved sample efficiency, and showed outstanding robustness and adaptability for controlling the truck trailer system.

The results of this study contribute to the development of intelligent ADAS technologies for truck trailer systems, which can potentially satisfy the growing demand for efficient and safe transportation market needs. The developed Reinforcement Learning (RL) based controllers demonstrated the adaptation capability to changing scenarios and overcoming the limitations of classical control methods, paving the way for a more robust and flexible TTRV intelligent control systems.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	iv
ABSTRACT	v
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS	xviii
CHAPTER ONE	
INTRODUCTION AND BACKGROUND	1
1.1. Background and Motivation	2
1.2. Adaptive and Self Learning System	4
1.3. Dissertation Objectives and Outlines	8
CHAPTER TWO	
MODELING OF TRUCK TRAILER ROBOTICS VEHICLES	11
2.1. Nonholonomic Systems and Vehicle Modeling	11
2.2. TTRV Kinematic Model	14
2.3. TTRV System Physical Constrains	18
2.4. Controllability and Reachability Analysis	23
2.5. Chapter Conclusion	28
CHAPTER THREE	
CLASSICAL TTRV PATH PLANNING METHODS	30
3.1. Path Planning Development History	30
3.2. Dubins Path Planning	32
3.3. Reeds-Shepp Path Planning	37

TABLE OF CONTENTS—Continued

3.4. Hybrid A* Path Planning	41
3.5. Kinodynamic RRT Path Planning	47
3.6. Chapter Conclusion	52
CHAPTER FOUR	
TRADITIONAL CONTROL METHODS FOR TTRV	53
4.1. Pole Placement and Gain Scheduler	53
4.2. Lyapunov Based Controller	60
4.3. Linear Quadratic Regulator Controller	65
4.4. Model Predictive Controller	72
4.5. Chapter Conclusion	82
CHAPTER FIVE	
MODEL FREE DEPP REINFORCEMENT LEARNING FOR TTRV REVERSE PARKING CONTROL	84
5.1. History of Reinforcement Learning	85
5.2. Graviation Potential Based Reward Function	90
5.3. On-Policy Approach: Proximal Policy Optimization Based TTRV Reverse Parking Control	93
5.3.1. Introduction of Proximal Policy Optimization	94
5.3.2. Implementation and Simulation	96
5.4. Off-Policy Approach: TD3 with Weight and Learning Rate Decay	103
5.4.1. Introduction to Twin Delayed Deep Deterministic Policy Gradient	104

TABLE OF CONTENTS—Continued

5.4.2. Enhanced TD3 with Adaptive Weight and Learning Rate Decay	107
5.4.3. Implementation and Simulation	109
5.5. Chapter Conclusion	113
CHAPTER SIX HYBRID MODEL-BASED REINFORCEMENT LEARNING FOR TTRV TRAJECTORY FOLLOWING AND PARKING CONTROL	115
6.1. Comparison Between Model-free And Model-based RL	115
6.2. Trajectory Planner Based on Hybrid A* algorithm	120
6.3. Integration of Model-based and Model-free Approaches	123
6.4. Reward Design for Truck-Trailer Path Following	126
6.5. Implementation and Simulation	129
6.6. Chapter Conclusion	136
CHAPTER SEVEN IMITATION INSPIRED PROBABILISTIC ENSEMBLES WITH TRAJECTORY SAMPLING FOR TTRV PARKING CONTROL	137
7.1. Introduction to Trajectory Sampling Methods	138
7.2. Learning from Imitation Inspiration	142
7.3. Implementation and Experimental Setup	145
7.4. Results and Discussion	149
7.5. Chapter Conclusion	156
CHAPTER EIGHT CONCLUSION AND FUTURE WORK	157
8.1. Conclusions	157

TABLE OF CONTENTS—Continued

8.2. Contributions	159
8.3. Future Work	160
REFERENCES	162

LIST OF TABLES

Table 5.1	Ford F-150 (Regular Cab, 6.5-foot box) and trailer Dimensions.	97
Table 5.2	Scania R 520 LA6x2/2MNA truck and trailer Dimensions.	100
Table 6.1	RL categories comparison based on the presence of a model and a global solution.	118

LIST OF FIGURES

Figure 1.1	Truck-trailer vehicles are widely used in logistics and transportation.	1
Figure 1.2	An simplified ADAS architecture.	5
Figure 1.3	Comparison of supervised, unsupervised, and reinforcement learning.	7
Figure 2.1	Kinematic full-track model of TTRV.	15
Figure 2.2	Simplified bicycle kinematic model of TTRV.	17
Figure 2.3	Physical representation of jack-knifing in truck-trailer systems.	20
Figure 2.4	Illustration showing the occurrence of jack-knifing in truck-trailer systems.	22
Figure 2.5	Open loop respond comparison between the positive and negative truck velocity.	29
Figure 3.1	Four patterns under CSC type.	32
Figure 3.2	Different Dubins contruction pattern for the same start and finish condition.	34
Figure 3.3	Dubins path starting from point (1, 1) with heading angle $-\frac{\pi}{4}$ and ending at (-3, -3) with heading angle 0.	36
Figure 3.4	Two Reeds-Shepp paths' illustration.	39
Figure 3.5	Comparison of A* and Hybrid A* search trees.	42
Figure 3.6	Hybrid A* Planning for truck-trailer parking.	43
Figure 3.7	Reeds-shepp based Hybrid A* Phases for truck-trailer parking.	46
Figure 3.8	The exploration stage of the Kinodynamic RRT* algorithm.	49
Figure 4.1	Pole placement diagram for TTRV reverse parking.	55
Figure 4.2	Pole placement simulation results for a 90 degree turn.	57

LIST OF FIGURES—Continued

Figure 4.3	Pole placement simulation results for a U turn.	58
Figure 4.4	Gain scheduling diagram.	59
Figure 4.5	Phase portrait for several stabilities.	61
Figure 4.6	Time domain plot for exponentially stable system.	62
Figure 4.7	Lyapunov-based controller Simulink diagram.	64
Figure 4.8	Lyapunov-based controller Simulation results.	65
Figure 4.9	LQR controller schematic diagram.	66
Figure 4.10	LQR controller reference states error.	68
Figure 4.11	The stacked simulation frames of using Dubins and LQR controller for TTRV reverse driving control.	70
Figure 4.12	LQR controller: trailer trajectory and desired parking path comparison.	70
Figure 4.13	LQR controller: control input of TTRV system.	71
Figure 4.14	LQR controller: trailer minimum lateral offset and steering angle comparison.	72
Figure 4.15	Function principle of a model-based predictive.	74
Figure 4.16	The stacked simulation frames of using Dubins and MPC controller for TTRV reverse driving control.	79
Figure 4.17	MPC controller: trailer trajectory and desired parking path comparison.	79
Figure 4.18	MPC controller: control input of TTRV system.	80
Figure 4.19	MPC controller: trailer minimum lateral offset and steering angle comparison.	81

LIST OF FIGURES—Continued

Figure 4.20	The stacked simulation frames of using RRT* and MPC controller for TTRV reverse driving control with obstacle avoidance.	82
Figure 5.1	The category of machine learning algorithms and reinforcement learning algorithms.	86
Figure 5.2	Markov Decision Process modeling decision-making in partly random situations.	88
Figure 5.3	Tabular learning methods store and update a table of values for each state-action pair.	88
Figure 5.4	Illustration of the gravity potential reward function, which guides the trailer towards the goal while avoiding obstacles.	91
Figure 5.5	Visualization of the clip function, illustrating how the probability ratio r is clipped at $1 - \varepsilon$ or $1 + \varepsilon$ depending on the sign of the advantage.	96
Figure 5.6	Structure of the PPO algorithm, including the DRL module and simulation framework.	97
Figure 5.7	TTRV perpendicular parking.	98
Figure 5.8	TTRV oblique parking demonstration.	99
Figure 5.9	Illustration of Scania R 520 LA6x2/2MNA truck and trailer in Unity simulation environment.	100
Figure 5.10	PyGame and Unity visualization environment showing trailer successfully parking.	101
Figure 5.11	Success rate of PPO algorithm.	102
Figure 5.12	Mean episode length of PPO algorithm.	103
Figure 5.13	Mean reward of PPO algorithm.	104
Figure 5.14	Structure of the TD3 algorithm, including the actor and critic networks.	106

LIST OF FIGURES—Continued

Figure 5.15	Unity visualization of trailer parking in obstacle-free environment using TD3WD.	110
Figure 5.16	Comparison of success rates between Twin Delayed Deep Deterministic Policy Gradient (TD3) and Weight Decayed Twin Delayed Deep Deterministic Policy Gradient (TD3WD).	111
Figure 5.17	Comparison of mean episode Lengths between TD3 and TD3WD.	112
Figure 5.18	Comparison of mean rewards between TD3 and TD3WD.	113
Figure 5.19	Diagram of imitation learning with expert demonstrations.	113
Figure 5.20	Unity visualization of trailer parking with obstacles in an evasive maneuver test using TD3WD.	114
Figure 6.1	Distinction between model-based and model-free reinforcement learning based on the presence of a model and a global solution.	116
Figure 6.2	Different stages in which the learned dynamics models can be employed.	119
Figure 6.3	Integration of trajectory state model-based reinforcement learning with PPO.	123
Figure 6.4	Vehicle trajectory state model considering preview reference points and deviation states.	125
Figure 6.5	Mean episode length over training steps.	130
Figure 6.6	Mean reward per episode over training steps.	130
Figure 6.7	Success rate over training steps.	131
Figure 6.8	Training loss over 3 million steps.	132
Figure 6.9	Explained variance over training steps.	133
Figure 6.10	Trajectory state model-based reinforcement learning agent maneuvering a "3" shaped path.	133

LIST OF FIGURES—Continued

Figure 6.11	Trajectory state model-based reinforcement learning agent maneuvering a double moose path.	134
Figure 6.12	Trajectory state model-based reinforcement learning agent maneuvering a narrow U-turn path with direction change.	135
Figure 7.1	Dyna algorithm architecture.	139
Figure 7.2	Probabilistic ensemble dynamics model for trajectory sampling.	141
Figure 7.3	Illustration of the IIPETS trajectory prediction process.	144
Figure 7.4	Overview of the IIPETS algorithm workflow.	146
Figure 7.5	Obstacle avoidance penalty function.	148
Figure 7.6	Overview of the IIPETS algorithm workflow.	149
Figure 7.7	Stacked screenshos of the IIPETS algorithm in an obstacle-free environment.	150
Figure 7.8	Trial rewards: original vs smoothed.	151
Figure 7.9	Validation scores: original vs smoothed.	151
Figure 7.10	Training loss: original vs smoothed.	152
Figure 7.11	Success episode: original vs smoothed.	153
Figure 7.12	Stacked screenshots of the IIPETS algorithm in an moost test environment.	154
Figure 7.13	Trial rewards with obstacles: original vs smoothed.	154
Figure 7.14	Success episode with obstacles: original vs smoothed.	155

LIST OF ABBREVIATIONS

ABS	Antilock Braking Systems
ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ESC	Electronic Stability Control
Hybrid A*	Hybrid A Star
IIPETS	Imitation Inspired Probabilistic Ensembles with Trajectory Sampling
Kinodynamic RRT*	Kinodynamic Rapidly-Exploring Random Tree
LQR	Linear Quadratic Regulator
LSTM	Long Short-Term Memory
MBPO	Model-based Policy Optimization
MBRL	Model-based Reinforcement Learning
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MFRL	Model-free Reinforcement Learning

LIST OF ABBREVIATIONS—Continued

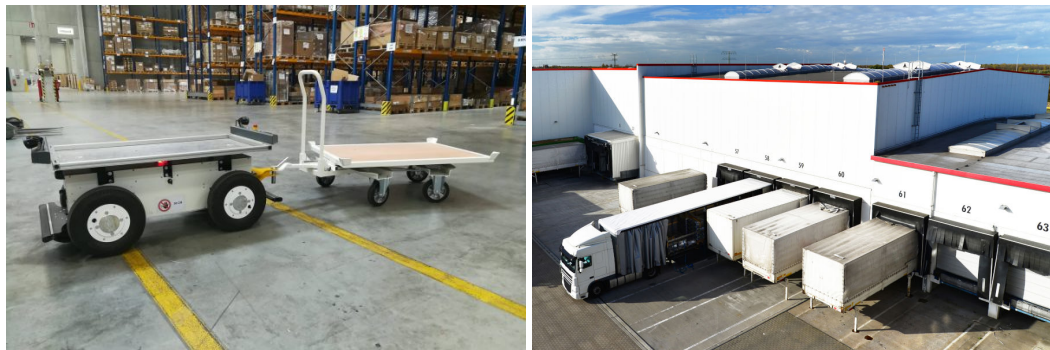
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MPC	Model Predictive Control
NN	Neural Network
PETS	Probabilistic Ensembles with Trajectory Sampling
PG	Policy Gradient
PID	Proportional–Integral–Derivative
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RRT	Rapidly-Exploring Random Tree
RRT*	Rapidly-Exploring Random Tree Star
RV	Recreational Vehicles
SAC	Soft Actor-Critic
SARSA	State-Action-Reward-State-Action
STEVE	Stochastic Trajectory Estimation with Value Expansion
TD3	Twin Delayed Deep Deterministic Policy Gradient
TD3WD	Weight Decayed Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
TTRV	Truck-trailer Robotics Vehicle

CHAPTER ONE

INTRODUCTION AND BACKGROUND

This chapter will give a general overview for the history of the TTRV control problem, the development of Advanced Driver Assistance Systems, and the history of Reinforcement Learning in vehicle automation. This chapter begins by offering essential background and motivation, underlining why these areas have gained so much research interest in recent years.

Section section 1.1 step into the critical challenges and opportunities inherent to truck trailer control, emphasizing its pivotal role in modern transportation logistics. In section 1.2, the focus shifts to the history of deep reinforcement learning, exploring how its adaptive capabilities can improve both vehicle safety and operational efficiency. Last part of this chapter is section 1.3, which outlines the core objectives of this dissertation, as well as the research questions that will be addressed.



(a) A small truck-trailer vehicle in a factory (b) A truck-trailer vehicle docking in a warehouse.

Figure 1.1: Truck-trailer vehicles are widely used in logistics and transportation.

1.1 Background and Motivation

Truck-trailer vehicle has been an important component for transportation industry nowadays, which has many user cases. In the logistics industry, heavy truck trailer facilitates the efficient transportation over long distances. Meanwhile, smaller trailers, such as Recreational Vehicles (RV) or box trailer, offer a cost-effective way for drivers to expand cargo capacity for daily travel, home improvements, and outdoor activities. However, these systems are facing challenges posed by complex road geometries, multi-agent interactions, and the need for precise integration between multiple sensors and perception, planning, and control algorithms. Particularly in the case of a TTRV system, the intricate dynamics characterized by nonlinearity, nonholonomic constraints, and under-actuation, compounded by the extensive blind zone at the trailer's rear end, make the development of a reverse driving control algorithm for TTRV exceptionally challenging[1][2]. Also during the COVID-19 pandemic, the truck driver shortage has caused significant downward effect on the global economy, which showed the critical role of the trucking industry in maintaining supply chains and the need for technologies that can help mitigate the impact of disruptions[3]. These challenges not only increase the difficulty of the career path for drivers but also diminish overall logistics time efficiency, affecting the productivity and profitability of the industry.

To address these challenges, researchers have been developing intelligent control strategies for TTRV systems. The goal is to reduce the load on human drivers, and potentially even replace the driver. This change is considered to bring lots of benefits. Safer driving experience as algorithms normally follow pre-defined patterns, and react faster than humans, also reduced greenhouse gas emissions and enhanced utilization of the overall vehicle fleet. The history of ADAS for vehicles can date back to the 1990s, at which time the passive-safety systems started to be installed on commercial vehicles,

including Antilock Braking Systems (ABS) and Electronic Stability Control (ESC)[4]. These safety features built the foundation for advanced technologies to improve of truck-trailer systems' safety and efficiency. According to the study from Insurance Institute for Highway Safety (IIHS), forward collision warning systems reduced rear-end crashes by 27%, while the addition of autonomous emergency braking further reduced these crashes by 50%[5]. For trucks, besides basic safety-related components such as brakes and tyres, advanced collision avoidance systems, including ACC and lane departure warnings, have been developed and widely installed. In Europe, truck-related fatalities decreased by 30% from 1995 to 2004. Human error causes 85.2% of truck accidents, with queue accidents being the second most common type. Main causes include non-adapted speed and insufficient safety distance[6]. The study aims to estimate the effectiveness of ADAS in reducing rear-end collisions involving trucks in queues.

In past decades, numerous classical control methods, such as PID, LQR, and MPC, as well as rule-based algorithms, have been applied to ADAS algorithms for vehicle control. While these algorithms have shown effectiveness in well-defined environments and scenarios, they often face the adaptability issue to handle the changing dynamics of real-world situations, which makes it challenging for ADAS to improve its performance in complex user situations which further limits the user cases. Especially with the advancements in hardware technologies such like lidar, radar, which provides more accurate environment detection capability, but the increasing data size and dimension also makes it harder for traditional control algorithms to fully handle, or find the hidden pattern in these data using rule-based control methods. This limitation hints that intelligent algorithms which can adapt to the changing driving scenarios and be capable of dealing with high dimensional data are required in the current market.

In recent years, neural network algorithms have shown promising capabilities in solving complex tasks, which attracted many researcher's attention. Famous achievements

include neural network algorithms beating human champions in games like Dota 2, chess, and Jeopardy[7, 8, 9]. However, despite these impressive performance in structured environments, it is challenging to deploy neural network algorithms to real-world robotics problems. One of the reason is the inherent complexity in real world environment, as well as unpredictability, and high-dimensional state space, and the challenge to generalize the algorithm into changing environments[10]. This challenge is particularly important in safety critical systems such as ADAS, where the usage of neural networks started from handling basic single tasks such as lane detection and obstacle recognition, and then more and more involved in highly integrated components for real-time decision-making during autonomous driving[11]. Early ADAS implementations relied on simple supervised learning, but more advanced techniques, such as reinforcement learning, can now solve the complexities of dynamic environments in autonomous vehicle control[12]. Despite these advances of neural networks in industry control field, the gap between game-like environments and real-world scenarios remains a challenging research topic, especially in industries with high safety requirements, such as autonomous vehicle control and industry robotics control.

1.2 Adaptive and Self Learning System

Nowadays, ADAS algorithms has developed to be a complex system consisted of several components, such as communication, perception, planning, and control. The perception module is for processing sensor data and extracting useful environment information, such as the position of surrounding obstacle, road geometry, and other useful data for vehicle driving. The planning module consumes perception information to generate trajectories that the vehicle should follow. Last but not least, the control module will take trajectory and generate control commands sent to the vehicle's actuators to

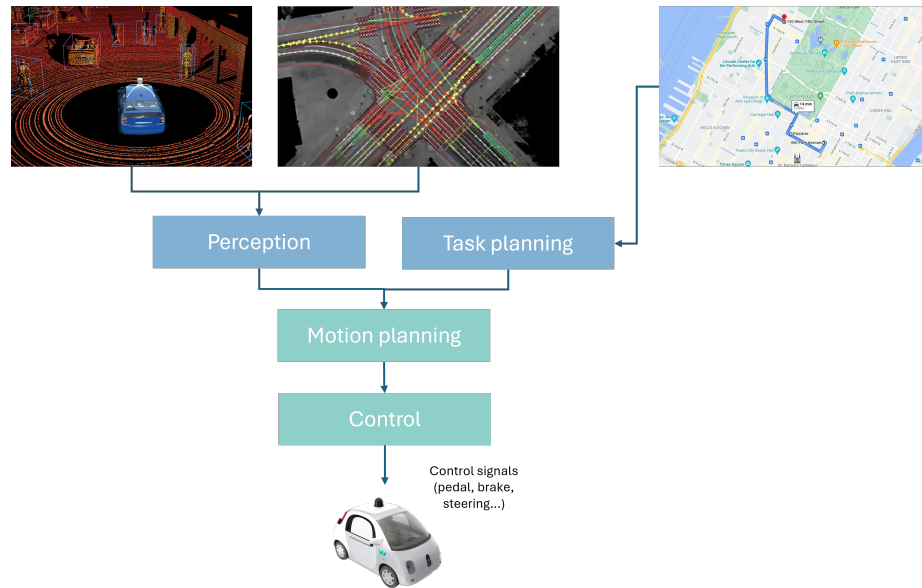


Figure 1.2: An simplified ADAS architecture. The perception module processes sensor data and extracts useful environment information. The planning module generates a trajectory based on this information. The control module generates control commands to execute the planned trajectory.

execute the planned trajectory. These subsystems communicate by transmitting and receiving information between each other using a well-defined middleware, and the performance of the ADAS highly depends on each component's accuracy and reliability, as well as the integration level between them. An overview of the mentioned structure can be seen in Figure 1.2.

Considering the complexity of manuvuering a vehicle in an urban environment: firstly the driver need to chcecking and tracking surrounding vehicles, and at the same time predicting their trajectories, and handling unexpected events, and the above steps are just a few of the many challenges. The modular ADAS approach, designed to handle predefined scenarios based on specific rules and algorithms, always show inadequacy in the real world if it cannot generalize to a unseen situation, adapt to different road

constraints, or react to unexpected events. This limitation hints that generalization of task solving capability can be even more important than specialization of task solving capability when it comes to real-world application. It has become clear that rule setting algorithm or control laws designed based on intricate assumptions for ADAS algorithm cannot meet modern market requirements; instead, the ability to generalize should draw more attention from researchers, allowing the system to reason on its own, evolve along with time, and act differently as the environment changes. The modular approach's limitations become evident when considering the large number of possible scenarios and the lack of adaptability when facing changing environments.

In recent years, neural network algorithms have shown promising capabilities in solving complex tasks, which attracted many researchers' attention. Famous achievements include neural network algorithms beating human champions in games like Dota 2, chess, and Jeopardy[7, 8, 9]. However, despite these impressive performances in structured environments, it is easy to deploy neural network algorithms to real-world robotics problems. One of the reasons is the real environment's inherent complexity, unpredictability, and high-dimensional state space, as well as the challenge to generalize the algorithm into changing environments[10]. The gap between game-like environments and real-world scenarios is particularly crucial in industries with high safety requirements, such as autonomous vehicle control. This trend represents a new direction toward adaptive and self-learning autonomous driving. By treating the entire control pipeline from sensor inputs to control commands as a single machine learning task, researchers want neural network models to have the potential to learn and adapt to a wide range of driving scenarios without too much manual engineering of each single sub-system. The self-learning nature of the neural network based autonomous driving systems is considered to be future mainstream, and companies like Tesla, Waymo, Huawei are all doing active research on this field. This allows the vehicle to handle changing driving

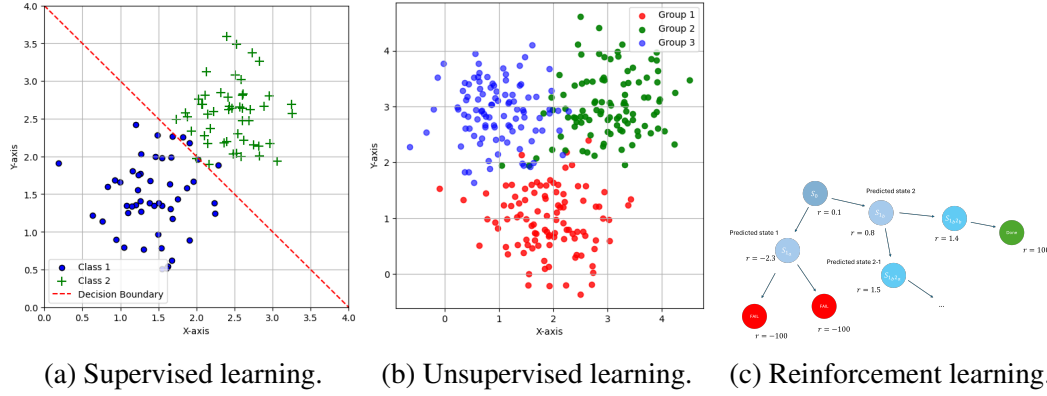


Figure 1.3: Comparison of supervised, unsupervised, and reinforcement learning.

scenarios which might not explicitly be considered or pre-programmed. And with the help of fast growing computational power, training a single model will becoming easierrn, and the algorithm evolution speed will even accelerate over time as more data becomes available.

Among all machine learning methods, reinforcement learning is known for its ability to develop adaptive and self-learning systems. Dated back to 1950s, RL is inspired from animal behaviorism and operant conditioning[13], and it has been developed greatly in the 1980s with the introduction of temporal difference learning by Sutton[14]. This was followed by the development of Q-learning by Watkins and Dayan in the 1990s[15]. A major breakthrough came in 2015 with the introduction of deep Q-networks by Mnih et al., enabling RL agents to achieve human-level performance in complex tasks like Atari games[16]. This makes RL a promising approach for developing adaptive and self-learning systems, such as autonomous driving. In RL, the driving agent learns to make decisions by interacting with the environment and receiving rewards or penalties based on its actions. Over time, the agent learns to optimize its behavior to maximize the cumulative reward. Also, benefited from the online data collection from many cars, the

large real driving experience dataset can be incorporated to the adaptive learning process, which can even fine tune the performance of deep reinforcement learning to better mimic human driving habitants. By learning from interactions with the environment and optimizing behavior based on rewards and penalties, RL agents can continuously improve their performance over time.

In conclusion, RL offers an effective approach for developing self-learning algorithms in autonomous driving control systems, and it enables the agent to improve continuously through exploration and exploitation as vehicle or robot interact with its surrounding environments. On the other hand, as research in RL for autonomous driving develops, it is important to address challenges such as improving safety level during agent's exploration, dealing with high-dimensional state and action spaces's effective representation, and developing stable and robust learning algorithms. By overcoming these limitations, RL has the potential to revolutionize the field of autonomous driving, which will lead a safer, more intelligent, and adaptable transportation systems, and eventually provides a potential solution for the complex challenges of autonomous vehicle control in real-world dynamics with changing physical constraints.

1.3 Dissertation Objectives and Outlines

The major objectives of this dissertation is to develop a deep reinforcement learning-based control system for TTRV that can learn effectively from collected driver experience trajectories, and adapt quickly to new and unseen driving scenarios. This research aims to present a comprehensive solution using modern reinforcement learning approaches to address the challenges posed by TTRV's nonholonomic systems. The proposed control system will be designed to handle multiple driving scenarios, not only limited to parking, but also includes lane following, obstacle avoidance, and other

common driving situations. The system will be evaluated in a simulation environment and visualized in Unity's 3D environment, providing a realistic representation of the vehicle's behavior in various driving scenarios.

The research has been structured to four stages. The first stage has its focus on providing background material on TTRV's nonholonomic systems, analyze the system's physical constraints and controllability, and then introduce several popular classical motion planning and feedback control methods, which has been used widely in current ADAS products. More detailing, this stage includes an in-depth discussion of kinematic model for TTRV, as well as classical methods such as Dubins and Reeds-Shepp paths, Hybrid A*, and Kinodynamic Rapidly-Exploring Random Tree (Kinodynamic RRT*). These foundations set the stage for understanding the complexities of such systems and the control techniques traditionally employed for vehicle trajectory planning and execution.

On the second stage, this dissertation introduces the concepts of model-free reinforcement learning methods, explores the pros and limitations of these approaches in the context of TTRV control. Especially, this dissertation will modify algorithms including Proximal Policy Optimization (PPO) and TD3 for the TTRV reverse control, and describe in details regarding their implementations and performance in both MATLAB and Python simulation environments. By showing the upto date reinforcement learning methods from both on-policy and off-policy, this stage's research demonstrates the capabilities of RL to handle complex control problems without intricate fine tune the system models.

The third stage followed by the model-free approaches is the exploration of model-based reinforcement learning techniques. In this stage, two different model-based RL approaches are introduced, which can be categorized as learning the policy over a known model, and jointly learning the model and policy. The trajectory-state-based approach is built on the idea of learning the path following policy over a known trajectory planning model, while the IIPETS approach is built on the idea of jointly learning the

model and policy using a probabilistic ensemble model, which gets the reward by predicting the future states, which is designed to achieve higher sample efficiency and better generalization capability.

Last but not least, this research also contributes a simulation environment for the TTRV control problem, which is highly extensible and modular. The simulation environment is built in Python using the Gymnasium robotics framework, and provides a flexible platform for simulating complex driving scenarios. In addition to this, a visualization framework is constructed in Unity for better 3D visualization of the vehicle's behavior, and it allows the user-interactive and realistic representation of the vehicle's behavior in different parking scenarios. These tools are important for testing and validating the performance of the proposed control algorithms in the virtual environment.

CHAPTER TWO

MODELING OF TRUCK TRAILER ROBOTICS VEHICLES

This chapter introduces the kinematic model approach used for motion planning and feedback control of wheeled vehicle. It begins with a brief overview of several popular modeling techniques in section 2.1. And in section 2.2, the nonholonomic system definition is discussed, as well as its characteristics and constraints. section 2.3 focuses on a simplified vehicle modeling method, kinematic bicycle model, which has been widely adopted for wheeled vehicles. Last but not least, section 2.4 performs the stability analysis of the proposed kinematic bicycle model, examining its equilibrium points, linearization, and stability criteria.

By discussing modeling concepts and techniques in this chapter, the theory foundation is built for the rest of this dissertation, allows deep understanding of motion planning and feedback control, as well as reinforcement learning approaches for the TTRV control problems.

2.1 Nonholonomic Systems and Vehicle Modeling

Wheeled vehicles can be modeled using different modeling techniques, and the choice of specific modeling technique depends on the application purpose or focus[17][18]. In general, a dynamic vehicle model is developed based on the concept of force balance, and a kinematic model is derived using velocity constraints as assumptions. For nonholonomic constrained systems, such as wheeled robots, the constraint equations have non-integrable properties, and the motion constraints include differential equations of generalized coordinates[19]. The kinematic models of these systems are derived from

the nonholonomic constraint matrix, which captures the relationship between the system's generalized coordinates and control inputs. On the other hand, the derivation of dynamic models falls within the realm of analytical mechanics, enabling an accurate description of the relationship between inputs and system variables.

The kinematic model of a wheeled vehicle is built assuming that the wheels roll without slippery on the surface. This no-slip condition leads to constraints on the vehicle's potential velocities, and limits vehicle's motion in certain direction range. Systems which have such velocity constraints, are called nonholonomic systems[20]. In nonholonomic systems, the velocity constraints are non-integrable, meaning they cannot be converted into equivalent position constraints[21]. This property shows the difference of holonomic systems, whose the constraints can be expressed solely in terms of the system's configuration variables. As a result, for a nonholonomic vehicle, to achieve desired position and orientation, it must perform certain complex maneuvers[22]. The study of nonholonomic systems is important for motion planning and control of wheeled vehicles, including robotic systems and autonomous cars. Researchers try to use different techniques, such as Lie algebraic methods and geometric control theory, to analyze and control these systems effectively[23].

Mathematically, a nonholonomic constraint can be expressed as:

$$\sum_{i=1}^n a_i(q) \dot{q}_i = 0 \quad (2.1)$$

where $q = (q_1, \dots, q_n)$ represents the generalized coordinates of the system, $\dot{q}_i$ is the time derivative of q_i , and $a_i(q)$ are functions of the generalized coordinates[24]. This equation shows a constraint on the velocities of the system, which cannot be integrated to obtain the equivalent position constraint. In other words, the constraint is non-integrable, and such system is called nonholonomic[20].

For the motion planning and control of nonholonomic systems, such non-integrability of the constraint equation has significant consequences: unlike holonomic systems, where the constraints can be explicitly expressed in terms of the system's configuration variables, nonholonomic systems are characterized by the presence of constraints that depend on the velocities. As a result, nonholonomic systems cannot instantly move in any arbitrary direction, and their motion is restricted to a subset of the possible velocity space[22].

Instead of specifying the velocity directions in which the vehicle is constrained, it is more practical to express the feasible directions of motion by transforming the constraints into an explicit form $\dot{q} = f(q, u)$. Assuming $m = n - k > 0$, let $g_1, \dots, g_m$ be selected as a basis of the right null-space of $\omega(q)$. By associating each $g_j \in \mathbb{R}^n$ with a control signal u_j , a control-affine driftless system can be obtained[1]:

$$\dot{q} = \sum_{j=1}^m g_j(q)u_j \quad (2.2)$$

where each $u_j, j = 1, \dots, m$, typically corresponds to a physical actuator. Nonholonomic systems are often characterized by underactuation, meaning that the number of control signals m is less than the dimension of the configuration space n . In general, a nonholonomic system can be modeled as a nonlinear system:

$$\dot{x} = f(x, u) \quad (2.3)$$

To analyze and control nonholonomic systems, various techniques have been developed. One prominent approach is the use of Lie algebraic methods, which exploit the underlying geometric structure of the system[25]. The Lie bracket operation, defined as:

$$[f, g] = \frac{\partial g}{\partial q} f - \frac{\partial f}{\partial q} g \quad (2.4)$$

where f and g are vector fields, plays a crucial role in characterizing the controllability and motion planning of nonholonomic systems. The Lie bracket essentially measures the degree of noncommutativity between two vector fields and provides insight into the system's controllability.

In conclusion, nonholonomic constraints are mathematically modelled as an non-integrable velocity constraints that restrict the motion of a system, and the study of nonholonomic systems has inspired the development of many motion planning algorithms, such as the Rapidly-Exploring Random Tree (RRT), which aim to efficiently explore the system's configuration space while respecting the nonholonomic constraints. These algorithms have been widely used in modern robotics, aerospace engineering, and other fields where modelled by nonholonomic systems.

2.2 TTRV Kinematic Model

The past research has developed several physical models for vehicle state analysis[26][27], including both kinematic and dynamic state models. The choice of model depends on the required accuracy level and the specific vehicle operating conditions. For example, in high-speed driving situations, the dynamic factors play a important role in motion planning and vehicle control; however, in low-speed driving scenarios, the kinematic constraints have a higher influence on the motion behavior, especially for the TTRV system. Because the TTRV reverse driving is normally operated with low-speed driving, the system model is hence built upon the kinematic relations. This research use the Ackermann chassis vehicle model for the host truck, and the trailer used a single-axle chassis. The two vehicles in the TTRV system are connected by the hitch ball, which is considered as a unique pivoting point. This pivot point enables these two vehicles to execute relative rotational motion between each other. From a mechanical structure

point of view, the truck and the trailer can be considered as two separate rigid bodies with the left and right sides being symmetrical on each single rigid body as shown in Figure 2.1. On flat non-slippery ground and under low-speed driving, the side-slip caused by tires, as well as the control delay caused by mechanical or electrical units can be ignored in the simulation[28][29].

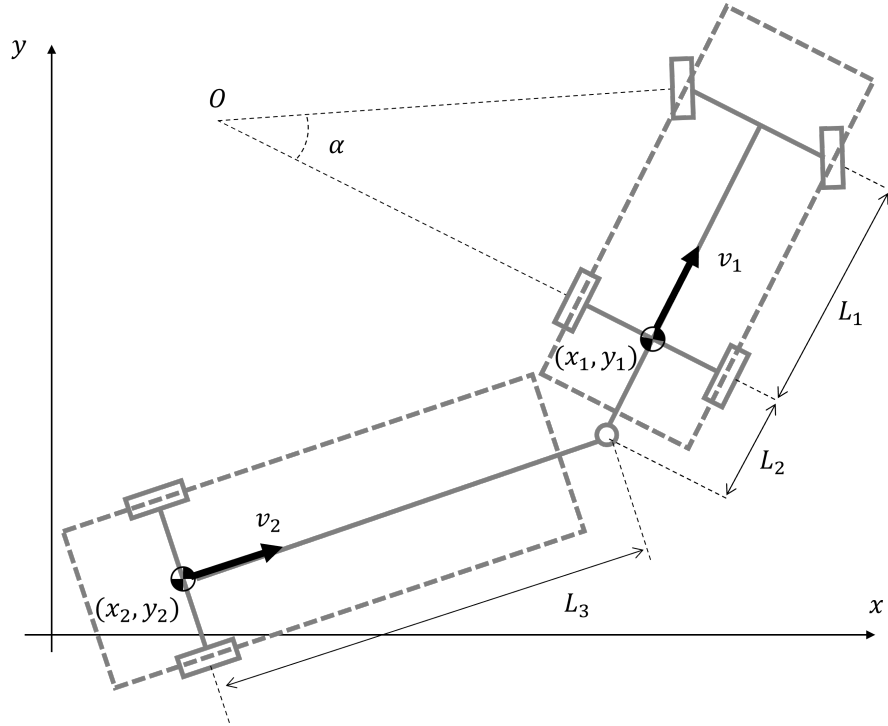


Figure 2.1: Kinematic full-track model of TTRV.

Because all the perception values are received in the vehicle coordinate, it is important to build the reference point coordinate transformation between the vehicle coordinate and the world coordinate. In this dissertation, only the translation from trailer

coordinate to world coordinate is explained, because the host coordinate transformation follows a similar computation.

Under the trailer coordinate, the reference points coordinates can be written as:

$$P_2 = \begin{bmatrix} x_{21} & x_{22} & \cdots & x_{2n} \\ y_{21} & y_{22} & \cdots & y_{2n} \end{bmatrix} \quad (2.5)$$

where P_2 is the coordinate vector matrix, each column vector of the matrix represents a reference point (e.g. surrounding obstacle detections, trailer corner coordinate and et al.) under the trailer coordinate system. $(x_{21}, x_{22}, \dots, x_{2n})$ and $(y_{21}, y_{22}, \dots, y_{2n})$ are X-coordinate and the Y-coordinate values respectively.

In linear algebra, the rotation matrix R_{world} and the translation matrix T_{world} can be known as:

$$R_{\text{world}} = \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 \\ \sin \theta_2 & \cos \theta_2 \end{bmatrix} \quad (2.6)$$

$$T_{\text{world}} = \begin{bmatrix} t_{x2} & t_{y2} \end{bmatrix}^T \quad (2.7)$$

where the (x_2, y_2) are the origin point of the trailer coordinate on the trailer rear axle center, and then the coordinate vector translation from trailer coordinate to world coordinate can be expressed as:

$$P_{\text{world}} = R_{\text{world}} * P_2 + T_{\text{world}} \quad (2.8)$$

In the world coordinate, the host vehicle state model is referred to as the front wheel Ackerman steering model, which allows control of the front wheel orientation relative to the heading of the vehicle. Due to limitations on the available computational power, vehicle dynamics are often highly simplified at the simulation phase[30]. Because the state model in this dissertation is carried out at low speed and with the assumption of

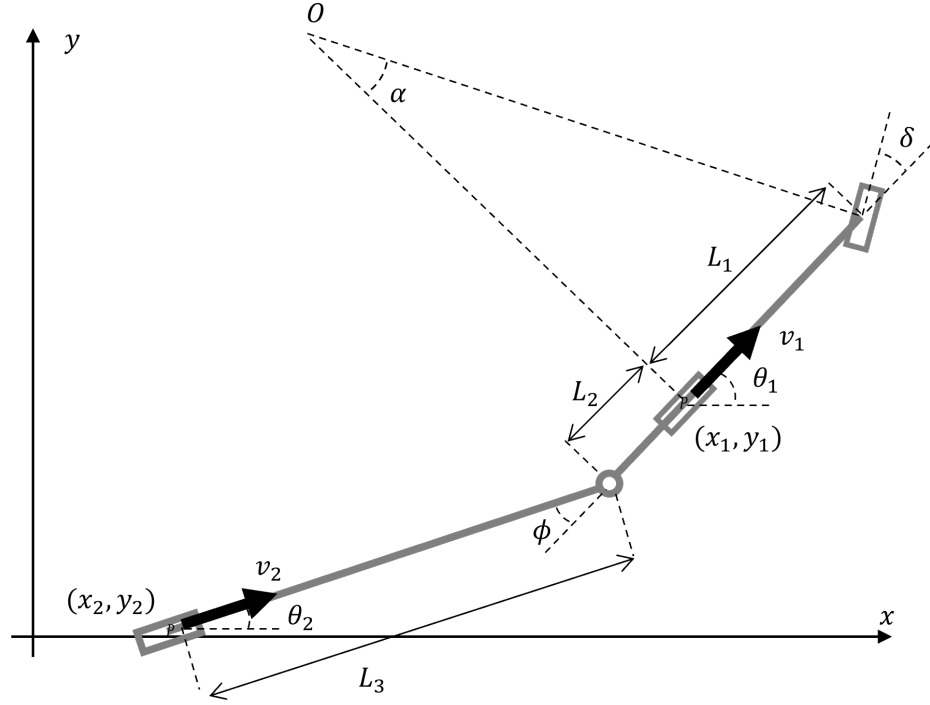


Figure 2.2: Simplified bicycle kinematic model of TTRV.

an ideally flat environment, the full track model represented in Figure 2.1 can be further simplified as the kinematic bicycle model in Figure 2.2, and the host vehicle kinematic equation can be expressed as:

$$\dot{x}_1 = v \cos \theta_1 \quad (2.9)$$

$$\dot{y}_1 = v \sin \theta_1 \quad (2.10)$$

$$\dot{\theta}_1 = \frac{v}{L_1} \tan \delta \quad (2.11)$$

The kinematic model of the trailer vehicle can be derived by propagating the state space of the host vehicle, as the trailer's kinematics are affected by the maneuver of the host vehicle. The longitudinal speed of the trailer can be estimated by projecting the

normal and longitudinal speed of the host vehicle onto the longitudinal axis of the trailer at the hitch point. Following the bicycle kinematic model, the states $[x_2, y_2, \theta_2]^T$ of the trailer can be expressed as:

$$\dot{x}_2 = v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \cos \theta_2 \quad (2.12)$$

$$\dot{y}_2 = v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \sin \theta_2 \quad (2.13)$$

$$\dot{\theta}_2 = -v \left(\frac{\sin \phi}{L_3} + \frac{L_2}{L_1 L_3} \cos \phi \tan \delta \right) \quad (2.14)$$

To evaluate the threat of jack-knifing, the host-trailer angle is also required for trailer state prediction, which is defined by the difference between Equation 2.14 and Equation 2.11:

$$\dot{\phi} = -\frac{v}{L_3} \sin \phi - \frac{v}{L_1} \left(1 + \frac{L_2 \cos \phi}{L_3}\right) \tan \delta \quad (2.15)$$

During reverse driving, the host-trailer system is limited to one degree of freedom in control, which is the host vehicle steering input, the system control is designed to minimize mainly the trailer's tracking error. So, the control reference variables are defined by the trailer's rear axle center position (x_2, y_2) , and heading θ_2 compared with the parking slot's reference position (x_{goal}, y_{goal}) , and heading angle θ_{goal} .

2.3 TTRV System Physical Constrains

Jack-knifing is a notorious constraint in the dynamics of truck-trailer wheeled robot systems. It occurs when the angle between the truck vehicle and trailer vehicle reaches the point where the control over the trailer's direction is lost during reverse motion. Instead of aligning to the desired trajectory, the truck trailer angle continues to

grow, eventually the trailer will collide with the truck. At this stage, the system enters an uncontrollable state, which is a critical failure mode in the control for TTRV. The physical representation of jack-knifing can be analyzed using the TTRV kinematic model. As shown in the model, the system input is the steering angle, provided by truck's front wheels, and then the trailer follows the truck's trajectory. But during reverse motion, the trailer's motion is very sensitive to the steering input of the truck. If the truck's steering angle is beyond a certain threshold, the trailer's angular velocity becomes larger than the truck's angular velocity, which will cause an unstable situation.

Jack-knifing occurs when the angle between the truck and trailer reaches a critical value during reverse motion, and this angle is denoted as β in Figure 2.3. At this point, the trailer's angular velocity $\dot{\beta}$ becomes greater than the truck's angular velocity $\dot{\alpha}$, even when the maximum steering input is given by the driver[31].

The critical angle when jack-knifing occurs under different steering angle can be calculated using the geometric properties of the TTRV. As shown in Figure 2.3, the critical angle is the sum of the resulting angles α and β when the two centers of rotation points coincide:

$$criticalangle = \alpha + \beta \quad (2.16)$$

The angles α and β can be expressed in terms of known length constraints (f , T and kerb diameter). Using the equations for $\dot{\phi}$ (Equation 2.15) and $\dot{\theta}_2$ (Equation 2.14), the expressions for α and β can be derived as:

$$\alpha = \arctan\left(\frac{L_2}{RC}\right) \quad (2.17)$$

$$\beta = \arcsin\left(\frac{L_3}{RD}\right) \quad (2.18)$$

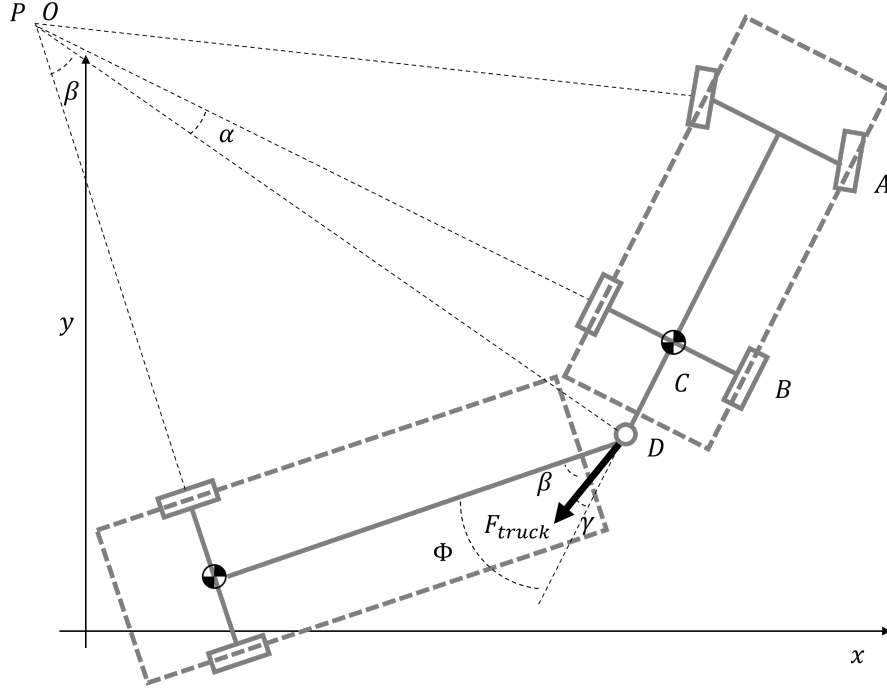


Figure 2.3: Physical representation of jack-knifing in truck-trailer systems.

The Pythagorean theorem can be employed to determine the distance to point D using the distances L_2 and RC :

$$RD = \sqrt{RC^2 + L_2^2} \quad (2.19)$$

The value of RC is obtained by subtracting half the track width, T , from RB :

$$RC = RB - \frac{T}{2} \quad (2.20)$$

Utilizing the Pythagorean theorem with L_1 and RA , the distance to point B can be calculated:

$$RB = \sqrt{RA^2 - L_1^2} \quad (2.21)$$

The kerb to kerb distance, defined as the diameter of the car's turning circle, is measured from the outer point of the wheel:

$$RA = \frac{K-f}{2} \quad (2.22)$$

By solving for α and β using the parameters RA , RB , RC , and RD :

$$\alpha = \arctan \left(\frac{L_2}{\sqrt{\left(\left(\frac{K-f}{2}\right)^2 - L_1^2 - \frac{T}{2}\right)}} \right) \quad (2.23)$$

$$\beta = \arcsin \left(\frac{L_3}{\sqrt{\left(\sqrt{\left(\left(\frac{K-f}{2}\right)^2 - L_1^2 - \frac{T}{2}\right)}}\right)^2 + L_2^2}} \right) \quad (2.24)$$

Consequently, the critical angle can be shown as the sum of α and β :

$$\begin{aligned} \text{critical angle} = & \arctan \left(\frac{L_2}{\sqrt{\left(\left(\frac{K-f}{2}\right)^2 - L_1^2 - \frac{T}{2}\right)}} \right) \\ & + \arcsin \left(\frac{L_3}{\sqrt{\left(\sqrt{\left(\left(\frac{K-f}{2}\right)^2 - L_1^2 - \frac{T}{2}\right)}}\right)^2 + L_2^2}} \right) \end{aligned} \quad (2.25)$$

The critical angle derived here is the theoretical maximum that ϕ can attain without being jack-knifed. If the angle β exceeds β_{critical} , the system becomes unstable, and jack-knifing occurs. As shown in Figure 2.4, once the TTRV passed the critical angle, the trailer vehicle will run a smaller circle than the truck vehicle; if the driver keep this steering angle, the trailer will eventually collide with the truck.

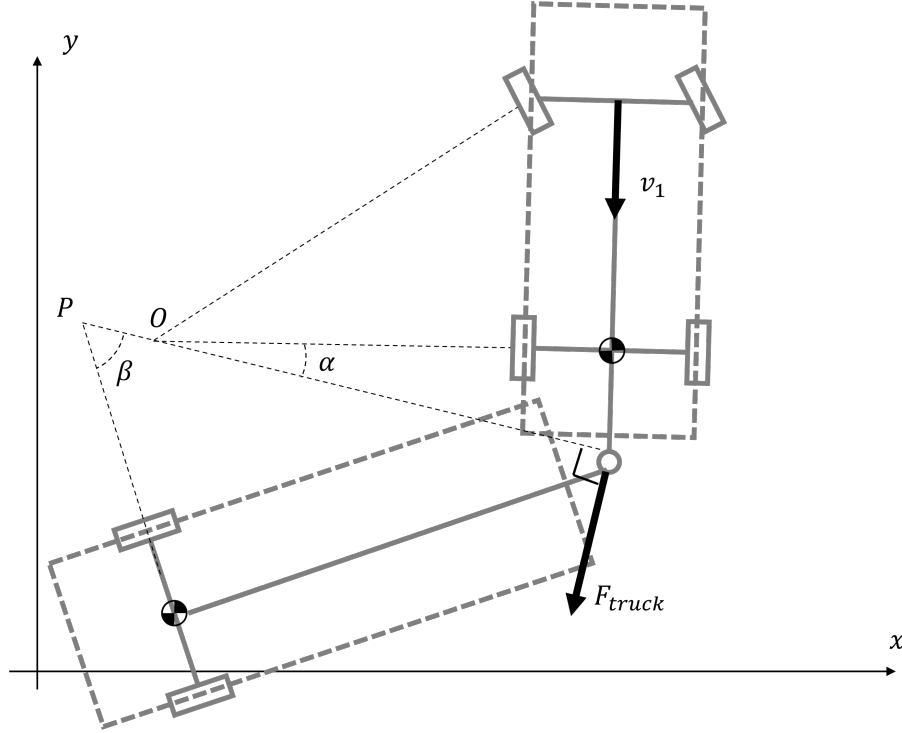


Figure 2.4: Illustration showing the occurrence of jack-knifing in truck-trailer systems.

To prevent jack-knifing, control strategies aim to maintain the stability of the truck-trailer system by limiting the steering input δ and ensuring that the angle β remains within the stable region. Feedback control techniques, such as nonlinear model predictive control and sliding mode control, can be employed to stabilize the system and avoid jack-knifing[31].

In conclusion, the physical representation and equations provided in this chapter can help the understand of the dynamics of jack-knifing under different steering angles. By analyzing the TTRV kinematic equations and analyzing the stability conditions, a clear explanation of jack-knifing is detailed, which provide the theoretical basis to develop effective control strategies to prevent this failure mode.

2.4 Controllability and Reachability Analysis

Controllability is a important feature for the control system, which can impact many control effects, including stabilizing the unstable systems through feedback control and optimizing control. Complete state controllability indicates that an external input, which are consisted of control variables, can be able to move the system's internal states from any starting states to any final states within finite time frame. In simplification, a system is considered controllable if there exists a control input that enables the transition from any initial state to any desired final state within a specified time interval.

Consider the continuous linear system

$$\begin{aligned}\dot{x}(t) &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)x(t) + D(t)u(t).\end{aligned}\tag{2.26}$$

where $x(t_0) = x^0$ is the initial states when the system starts.

The state pair sets (x, u) which solve Equation 2.26 for by setting the initial states (t_0, x^0) is called the behaviour $\mathcal{B}_{(A,B)}$ of Equation 2.26:

$$\mathcal{B}_{(A,B)} := \left\{ (x, u) \in X \times U \mid \exists (t_0, x^0) \in \mathbb{R} \times \mathbb{R}^n, \quad x(\cdot) = x(\cdot; t_0, x^0, u) \right\} \tag{2.27}$$

Here, X is a function space that contains all possible state trajectories of the system:

$$\left\{ x(\cdot; t_0, x^0, u) \mid (t_0, x^0) \in \mathbb{R} \times \mathbb{R}^n, \quad u \in U \right\} \subseteq X. \tag{2.28}$$

Regarding the concept of controllability, the system (A, B) is called

- Reachable at time $T > 0$ if for all $x^1 \in \mathbb{R}^n$ there exists $(x, u) \in \mathcal{B}_{(A,B)}$ such that $x(0) = 0$ and $x(T) = x^1$,
- Controllable at time T if for all $x^0, x^1 \in \mathbb{R}^n$ there exists $(x, u) \in \mathcal{B}_{(A,B)}$ such that $x(0) = x^0$ and $x(T) = x^1$,

- Null-controllable at time T if for all $x^0 \in \mathbb{R}^n$ there exists $(x, u) \in \mathcal{B}_{(A,B)}$ such that $x(0) = x^0$ and $x(T) = 0$.

To determine how easy the system can be controller, or how much effort needed to control the system, it need to test the eigen decomposition of the Controllability Gramian. This eigen decomposition can reveal relationships between the system states and how they respond to the control inputs. Here, the Controllability Gramian $W(t_0, t_1)$ can be defined as:

$$W(t_0, t_1) = \int_{t_0}^{t_1} \phi(t_0, t) B(t) B(t)^T \phi(t_0, t)^T dt \quad (2.29)$$

where ϕ is the state-transition matrix.

Equation 2.29 means if there exists a control u from state x_0 at time t_0 to state x_1 at time $t_1 > t_0$ if and only if $x_1 - \phi(t_0, t_1) x_0$ is in the column space of the Controllability Gramian.

In fact, if η_0 is a solution to $W(t_0, t_1) \eta = x_1 - \phi(t_0, t_1) x_0$ then a control given by $u(t) = -B(t)^T \phi(t_0, t)^T \eta_0$ would make the desired transfer.

Note that the matrix W has the following properties:

- $W(t_0, t_1)$ is symmetric
- $W(t_0, t_1)$ is positive semidefinite for $t_1 \geq t_0$
- $W(t_0, t_1)$ satisfies the linear matrix differential equation

$$\frac{d}{dt} W(t, t_1) = A(t) W(t, t_1) + W(t, t_1) A(t)^T - B(t) B(t)^T, W(t_1, t_1) = 0 \quad (2.30)$$

- $W(t_0, t_1)$ satisfies the equation[32]

$$W(t_0, t_1) = W(t_0, t) + \phi(t_0, t) W(t, t_1) \phi(t_0, t)^T \quad (2.31)$$

Gramians are also useful to determine the minimum-energy control $u(t)$ required to navigate the system to $x(t_f)$ at time t_f from $x(0) = 0$ [33]:

$$u(t) = B^* \left(e^{A(t_f-t)} \right)^* W_c(t_f)^{-1} x(t_f). \quad (2.32)$$

The total energy expended by this control law is given by

$$\int_0^{t_f} \|u(\tau)\|^2 d\tau = x^* W_c(t_f)^{-1} x. \quad (2.33)$$

If the controllability matrix is close to singular, more control energy will be required to drive the states towards certain direction. On the other hand, if the eigenvalues of W_c are all large, then the system can be controlled more easily in all directions, also requiring less energy to reach any given state.

It is generally impractical to compute the Gramians directly using Equation 2.29. Instead, the Controllability Gramian is the solution to the following Lyapunov equation:

$$A W_c + W_c A^* + B B^* = 0, \quad (2.34)$$

while the observability Gramian is the solution to

$$A^* W_o + W_o A + C^* C = 0. \quad (2.35)$$

Obtaining Gramians by solving a Lyapunov equation is typically quite expensive for high-dimensional systems. Instead of this computationally demanding approach, Gramians are frequently approximated empirically. This approximation is achieved using snapshot data gathered from both the direct system and the adjoint system, providing a more efficient means to estimate these important mathematical constructs.

In practice, the Controllability Gramian requires the integration of a system's state-transition matrix. For a simpler assessment of system controllability, a rank condition can be used that is similar to the Kalman rank condition found in time-invariant

systems, which shows that the Controllability of a LTI system is determined entirely by the column space of the controllability matrix $\mathcal{C}$:

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2 B & \dots & A^{n-1} B \end{bmatrix}. \quad (2.36)$$

If the matrix $\mathcal{C}$ has n linearly independent columns, meaning that it spans all of $\mathbb{R}^n$, then the system is considered controllable. The span of the columns of the controllability matrix $\mathcal{C}$ forms a Krylov subspace, which identifies the directions in $\mathbb{R}^n$ that can be manipulated with control. Consequently, controllability not only implies the possibility of arbitrary eigenvalue placement but also ensures that any state $\xi \in \mathbb{R}^n$ can be reached within a finite time using a specific actuation signal $u(t)$ [33].

In general, for a system (A, B) the following statements are equivalent:

- There exists $T > 0$ such that (A, B) is controllable at time T .
- $\text{im}(K(A, B)) = \mathbb{R}^n$
- $\text{rank}(K(A, B)) = n$
- For all $T > 0$ the system (A, B) is controllable at time T . It is said that (A, B) is controllable.

To identify the properties and conditions of the TTRV system, its equilibrium, observability, and controllability are analyzed. Let $x = [y_2, \theta_2, \phi]^\top$ be the trailer states obtained from Equation 2.12 - Equation 2.14 by neglecting the longitudinal component x_3 , the state space model can be written as[34]:

$$\dot{x} = v(\mathcal{A}(x) + \mathcal{B}(x; \delta)) \quad (2.37)$$

The equilibrium state of x is the origin $x_e = 0$, and then derive the trailer equilibrium state from Equation 2.15:

$$x_e = \{(y_2, \theta_2, \phi) \mid \delta = 0, y_2 \in \mathbb{R}, \theta_2 \in [-\pi, \pi), \phi = 0\} \quad (2.38)$$

The equilibrium set indicates during reverse driving, the trailer system is in an equilibrium state when the host vehicle and trailer are aligned in either direction, and there should be no steering angle input given by the host vehicle.

To check the system's internal asymptotic stability, substitute $\delta = 0$ into Equations (2.13) to (2.15), resulting in the following equation:

$$\dot{x} = v(\mathcal{A}x + \mathcal{B}\delta) \quad (2.39)$$

where

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{L_3} \\ 0 & 0 & -\frac{1}{L_3} \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} 0 \\ \frac{L_2}{L_1 L_3} \\ -\frac{L_3 + L_2}{L_1 L_3} \end{bmatrix} \quad (2.40)$$

By using the Cayley-Hamilton theorem, the controllability matrix $\mathcal{C} = [B, AB, AAB]$ is row full rank for state space in Equation 2.39, which shows the host-trailer system to be controllable out of the singular locus[35], and the origin of the nonlinear system can be made an asymptotically stable equilibrium by state feedback.

The stability of the system near its equilibrium point $\phi = 0$ can be analyzed by testing the linearized dynamics in Equation 2.15:

$$\dot{\phi} = -\frac{V}{L_3} \sin(\phi) \quad (2.41)$$

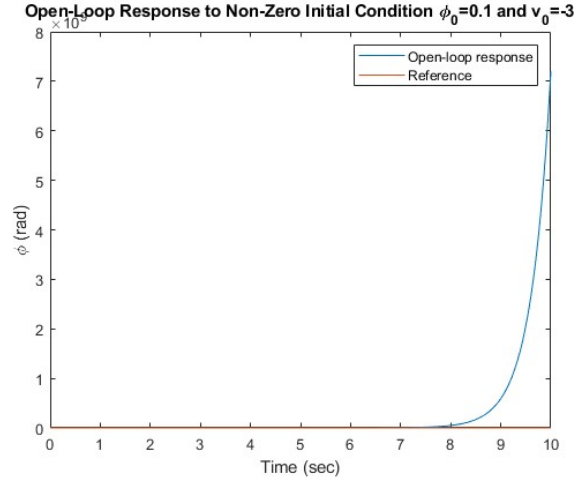
where the stability is determined by the sign of:

$$\frac{d}{d\phi} \left(-\frac{V}{L_3} \sin(\phi) \right) = -\frac{V}{L_3} \cos \phi \quad (2.42)$$

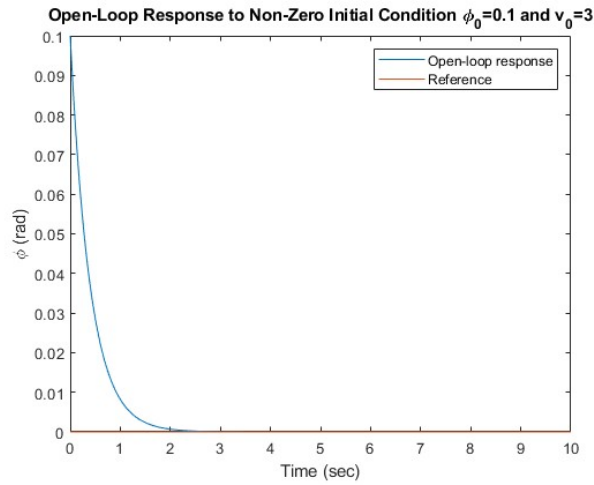
when $\phi = 0$, which can then be simplified to $-\frac{V}{L_3}$. When the system is driving forward, the system input $V > 0$, it will result in a negative value, indicating a stable equilibrium. In the opposite way, for $V < 0$ during backward driving, this equation will be a positive value, which is an unstable equilibrium. This aligns with driving experience: a trailer is inherently stable when moving forward, while always tends to jackknife itself when driving reversely, and it requires active control to maintain system stability, as shown in Figure 2.5.

2.5 Chapter Conclusion

In this chapter, the kinematic model of the TTRV system was introduced, and the physical constraints of the system were analyzed, especially for the root cause for trailer jackknifing. Then, to analysis the system stability and controllability, the controllability gramian and the system stability near the equilibrium point were discussed. The simulation shows that the TTRV system is controllable and stable when it is driving forward, but the system is unstable when it is driving reversely. This analysis provides the theoretical basis for the development of feedback control methods to stabilize the system and prevent jack-knifing. In the next chapter, the path planning and control methods for the TTRV system will be discussed, including the design of different feedback control system for the path following task.



(a) The positive velocity (forward driving) indicates stable system behaviour when the TTRV can correct its Φ value.



(b) The negative velocity (reverse driving) will cause unstable system performance, the Φ angle keeps increasing until it reaches jackknifing.

Figure 2.5: Open loop response comparison between the positive and negative truck velocity.

CHAPTER THREE

CLASSICAL TTRV PATH PLANNING METHODS

In the chapter and the following CHAPTER FOUR, a classical modular control system is developed as a baseline for future benchmark testing. The system consists of two main components: path planning and path following. This chapter focuses on the path planning module, which generates feasible paths for the truck-trailer system to follow.

This chapter first introduce the background and development history of vehicle path planning in section 3.1, and then introduce the Dubins path planning method in section 3.2, followed by the Reeds-Shepp path planning method which allows forward-backward adjustment driving in section 3.3. In section 3.4, the discrete path planning method Hybrid A* path planning is introduced, and finally, the Kinodynamic RRT* path planning method designed for continuous state space is introduced in section 3.5. All methods mentioned in this chapter are modified based on the kinematic model of the TTRV system, and simulation results are provided to demonstrate as well as compare the performance of different methods.

3.1 Path Planning Development History

Path planning is a basic module in robotics and autonomous vehicle navigation, especially for complex maneuvers such as offroad driving and parking. It's goal is to find a feasible and efficient route from the initial state to a goal state, while avoiding obstacles and satisfying various physical constraints[36]. The primary goals of path planning include generating collision-free trajectories, minimizing travel distance and time, and

ensure vehicle's smooth motion while considering vehicle kinematic or dynamic constraints[37].

Recent advancements in path planning algorithms have greatly improved the capabilities for autonomous vehicles during navigating complex environments. Classical approaches such as graph-based methods and sampling-based planners have been expanded with more complex techniques which take vehicle dynamics and real-world uncertainties into considerations[38]. These developments have led to more robust and efficient planning strategies, allowing to handle the challenges posed by autonomous parking scenarios nowadays.

Current research in path planning for autonomous parking focuses on several directions, including improving the computational efficiency of planning algorithms which optimize the real-time performance, incorporating realistic vehicle models to generate more feasible trajectories, and developing adaptive methods which handle dynamic environments with unexpected obstacles[39]. Additionally, many research tries to integrate machine learning approaches with traditional planning methods to enhance decision-making capabilities and generalize the method to changing parking scenarios[40].

Although many progress has been made, certain challenges remain in developing path planning algorithms which can consistently perform as good as or even better than human-level in a wide range of parking situations. Ongoing research aims to address issues such as maneuvering in tight spaces, considering multi-vehicle interactions, and generalize the algorithm with difference vehicle configurations. As autonomous driving technology continues to develop, future path planning algorithm advancements will help the ADAS maintain safe and efficient autonomous driving and parking performance.

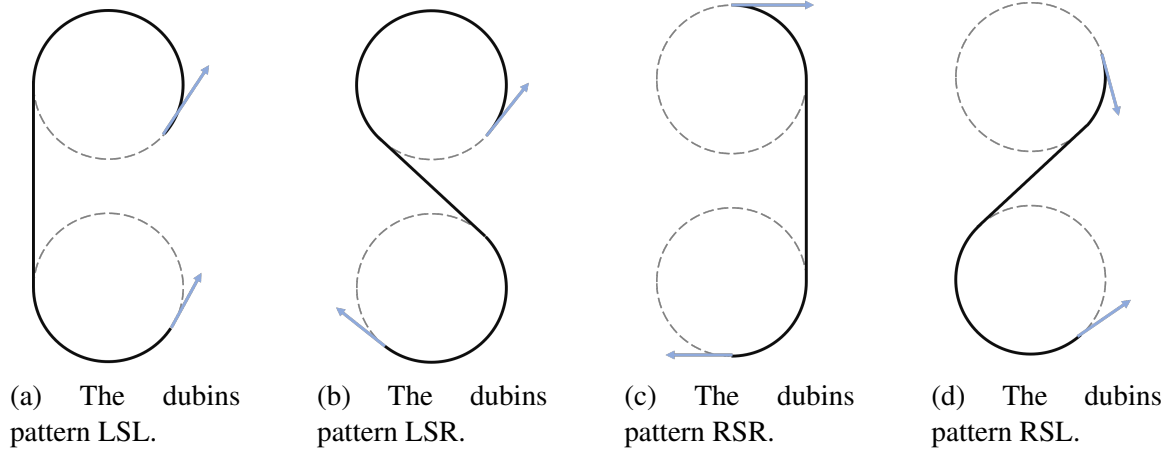


Figure 3.1: Four patterns under CSC type.

3.2 Dubins Path Planning

Dubins paths, introduced by L.E. Dubins in 1957[41], is the shortest paths between two points of specified initial and final orientations, for a vehicle moving at constant speed with a constrained turning radius. Dubins paths is consisted as the combinations of circular arcs (C) and straight line segments (S), provides an simple but effective solution for non-holonomic vehicle navigation.

The optimal Dubins path belongs to one of six possible types: RSR, RSL, LSR, LSL, RLR, or LRL, where R and L denote right and left turns respectively. These can be categorized into two main classes: CSC (Circle-Straight-Circle) and CCC (Circle-Circle-Circle), as illustrated in Figures 3.1.

The Dubins car model can be described by the kinematic bicycle model:

$$\dot{x} = v \cos(\theta) \quad \dot{y} = v \sin(\theta) \quad \dot{\theta} = \frac{v}{r} \quad \text{label}eq : \text{kinematicbicyclemodel} \quad (3.1)$$

where (x, y) is the position, θ is the orientation, v is the constant velocity, r is the minimum turning radius, and $u \in [-1, 1]$ is the control input.

The turning radius r is computed using vehicle's steering angle δ and wheelbase L by:

$$r = \frac{L}{\tan(\delta)} \quad (3.2)$$

To build Dubins paths starting with the initial pose $P_s(x_s, y_s, \theta_s)$ and ending with final pose $P_f(x_f, y_f, \theta_f)$, firstly the centers of the turning circles, $O_s(x_{cs}, y_{cs})$ and $O_f(x_{cf}, y_{cf})$, are computed based on the initial and final poses and the turning radius. The following algorithm algorithm 3.1 shows the logic of finding the optimal Dubins path:

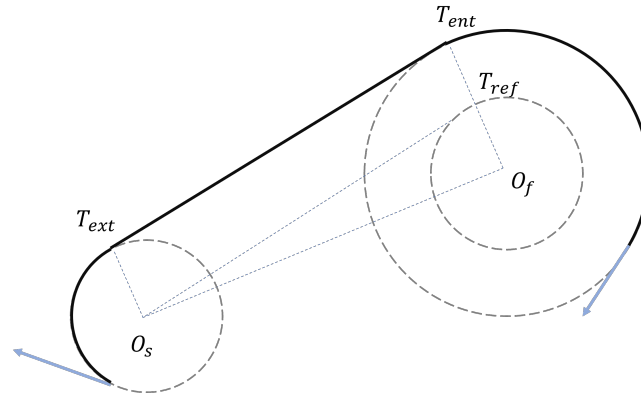
Algorithm 3.1: Dubins Path Planning.

Input: Start pose P_s , Final pose P_f , Turning radius r

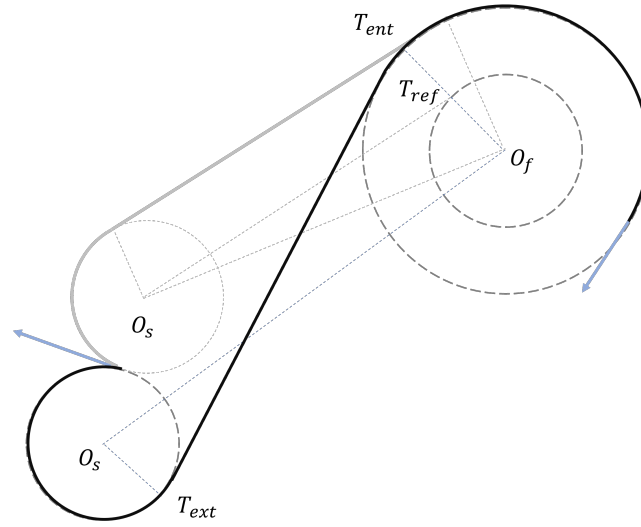
Output: Optimal Dubins path

1. **Function** DubinsPath(P_s, P_f, r):
 2. Compute turning circle centers O_s and O_f
 3. paths $\leftarrow \{\}$
 4. **foreach** path type in $\{RSR, RSL, LSR, LSL, RLR, LRL\}$ **do**
 5. path $\leftarrow$ ComputePath(P_s, P_f, r , path type); **if** path is valid **then**
 6. paths.append (path)
 7. **end**
 8. **end**
 9. **return** ShortestPath(paths)
-

The length of a Dubins path is calculated by summing the lengths of its straight segments and circular arcs, as shown in algorithm 3.3. And in the last, path with the minimum length is selected as the optimal path, as shown in algorithm 3.2. Figure 3.3



(a) Dubins construction following RSR pattern.



(b) Dubins construction following LSR pattern, the pattern for RSR is in grey color for comparison.

Figure 3.2: Different Dubins construction pattern for the same start and finish condition.

Algorithm 3.2: Find shortest path.

```
1. Function ShortestPath(paths):  
2.   shortest  $\leftarrow$  null  
3.   min_length  $\leftarrow \infty$   
4.   foreach path  $\in$  paths do  
5.     length  $\leftarrow$  CalculateLength(path)  
6.     if length < min_length then  
7.       shortest  $\leftarrow$  path  
8.       min_length  $\leftarrow$  length  
9.     end  
10.  end  
11.  return shortest
```

Algorithm 3.3: Calculate path length.

```
1. Function CalculateLength(path):  
2.   length  $\leftarrow$  0  
3.   foreach segment  $\in$  path do  
4.     if segment.type  $\in$  {S, S-} then  
5.       length  $\leftarrow$  length + |segment.length|  
6.     else  
7.       length  $\leftarrow$  length + r * |segment.angle|  
8.     end  
9.   end  
10.  return length
```

shows an example of a Dubins path, and Figure 3.2 illustrates how different patterns can be generated for the same start and end conditions.

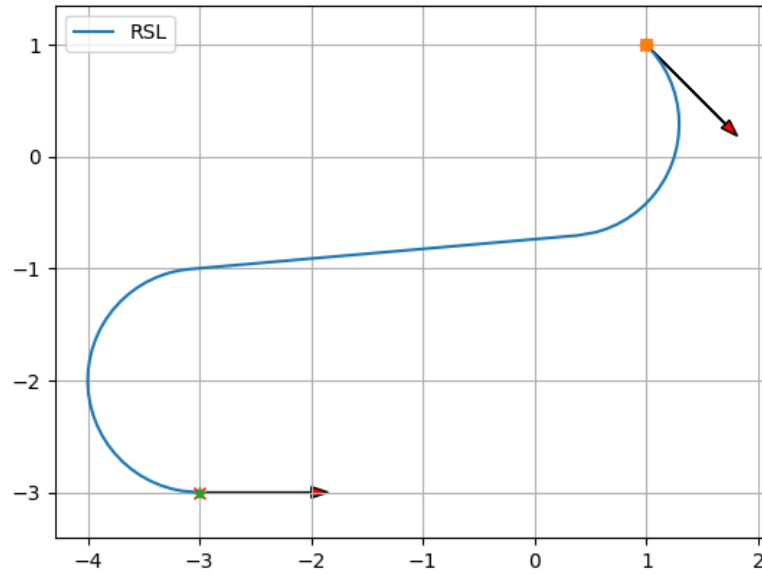


Figure 3.3: Dubins path starting from point (1, 1) with heading angle $-\frac{\pi}{4}$ and ending at (-3, -3) with heading angle 0.

In conclusion, Dubins path algorithm provides a fundamental method for path planning for non-holonomic vehicles, which considers the minimum turning radius constraint. However, this method is limited to one driving direction motion, This constraint can cause limited or even failure performance in many real-world scenarios, when the vehicle needs to forward-backward adjust its direction or change their speed. As a result, development of path planning techniques which allows driving direction change

or radius change is required, to provide flexibility in path planning for autonomous vehicles.

3.3 Reeds-Shepp Path Planning

In the previous chapter, Dubins path planning has been introduced, which computes optimal paths for vehicles moving in one direction with constrained turning radius. However, Dubins paths have a significant limitation: they assume the vehicle can only move in one direction. In real-world scenarios especially in tiny spaces, drivers need to move both forward and backward to successfully park the vehicle. In this chapter, Reeds-Shepp path is introduced, which extends the Dubins model to include path segments with opposite direction, thereby offering a powerful approach to path planning in limited freespace.

Reeds-Shepp paths were firstly introduced in 1990[42], are the shortest paths between two states (position and orientation) for vehicle which can move both forward and backward. Similar to Dubins paths, the Reeds-Shepp paths assume a constant turning radius, but this method allows the vehicle changing direction in certain path segments.

Similar to Dubins paths, Reeds-Shepp paths also use the kinematic bicycle model to describe the vehicle motion. and the key difference from the Dubins model is the introduction of u , allowing for both changing the velocity's v polarity in the kinematic bicycle model, as shown in algorithm 3.4 algorithm 3.5. This small modification can significantly expand the number of feasible paths, leading to more optimal trajectories.

Compared with Dubins path planning, Reeds and Shepp path planning proves that the shortest path between any two configurations consists of at most five segments, each of which is either a straight line (S) or an arc of a circle with the minimum turning radius

(C), which can make up 48 possible types of optimal paths, compared to Dubins' 6 different path types.

Algorithm 3.4: Main Reeds-Shepp path planning algorithm.

Input: Start configuration q_{start} , Goal configuration q_{goal} , Minimum turning radius r

Output: Optimal Reeds-Shepp path

```

1. Function ReedsSheppPath( $q_{start}, q_{goal}, r$ ):
2.    $paths \leftarrow \emptyset$ 
3.   foreach  $word \in REEDS\_SHEPP\_WORDS$  do
4.      $path \leftarrow \text{GeneratePath}(q_{start}, q_{goal}, r, word)$ 
5.     if  $path \neq null$  then
6.        $paths \leftarrow paths \cup \{path\}$ 
7.     end
8.   end
9.   return ShortestPath( $paths$ )

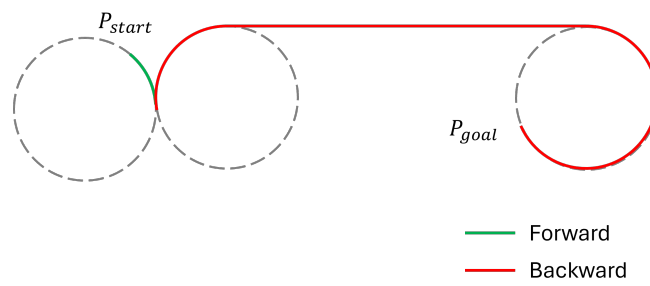
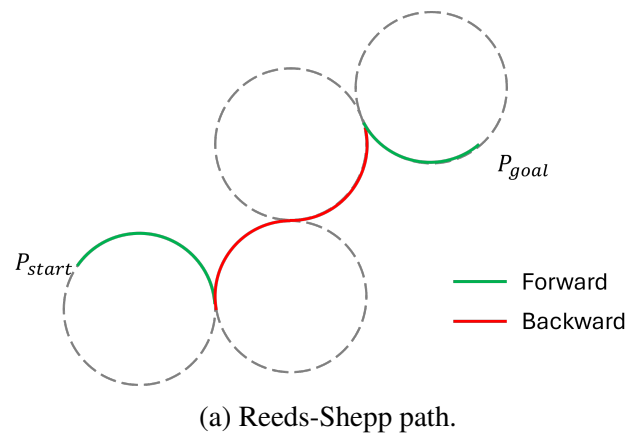
```

Similar to the algorithm used in previous algorithm 3.2 and algorithm 3.3, the length of a Reeds-Shepp path can be calculated using the following equation:

$$L = \sum_{i=1}^n |s_i| + r \sum_{j=1}^m |\theta_j| \quad (3.3)$$

where s_i represents the length of the i -th straight segment, θ_j is the angle subtended by the j -th circular arc, and n and m are the numbers of straight segments and circular arcs, respectively.

One advantage of Reeds-Shepp path planning is its ability to generate shorter trajectories than Dubins paths in tight spaces. For instance, consider a vehicle that needs to parallel park, a Dubins path would require a large turning radius to approach the parking spot with only one possible final vehicle heading, a Reeds-Shepp path can reverse



(a) Reeds-Shepp path.

(b) Reeds-Shepp path with straight line segment.

Figure 3.4: Two Reeds-Shepp paths' illustration.

Algorithm 3.5: Generate path for Reeds-Shepp algorithm.

```
1. Function GeneratePath( $q_{start}, q_{goal}, r, word$ ):  
2.    $q \leftarrow q_{start}$   
3.    $path \leftarrow \emptyset$   
4.   foreach  $segment \in word$  do  
5.     if  $segment.type = S$  then  
6.        $q \leftarrow \text{MoveForward}(q, segment.length)$   
7.     else if  $segment.type \in \{L, R\}$  then  
8.        $q \leftarrow \text{Turn}(q, r, segment.angle, segment.type)$   
9.     else if  $segment.type \in \{S^-, L^-, R^-\}$  then  
10.       $q \leftarrow \text{ReverseMotion}(q, r, segment)$   
11.     end  
12.      $path \leftarrow path \cup \{segment\}$   
13.   end  
14.   if  $IsClose(q, q_{goal})$  then  
15.     return  $path$   
16.   else  
17.     return null  
18.   end
```

driving into this tight space. However, the increased flexibility of Reeds-Shepp path planning comes at the cost of higher computational complexity. The search space for optimal paths is larger, and the inclusion of direction changes requires more connection smooth effort. The time complexity for computing the optimal Reeds-Shepp path between two configurations is $O(n \log n + k)$, where n is the number of obstacles in the environment and k is the number of intersections between circular arcs and straight lines in the path.

In conclusion, Reeds-Shepp paths have found wide application in robotics, autonomous vehicles, and computer animation, which provide a more realistic model for vehicle to navigate and enable navigation in complex environments with better maneuvering efficiency.

3.4 Hybrid A* Path Planning

Hybrid A Star (Hybrid A*)[43] is a path planning algorithm that combines the discrete property of traditional A* search with continuous state space representation. This approach is especially suitable for non-holonomic vehicles, such as TTRV, where motion constraints is an critical limitation to be considered during path planning. This section explores how Hybrid A* can be adapted for a truck-trailer system, with a focus on generating efficient parking trajectories.

The core of Hybrid A* is to discretize the search space for efficiency and use a continuous state representation for accuracy. Each node in the search tree represents a continuous state of the vehicle, including position, orientation, and any other parameters from the vehicle model.

For this research, the TTRV state vector s can be defined as:

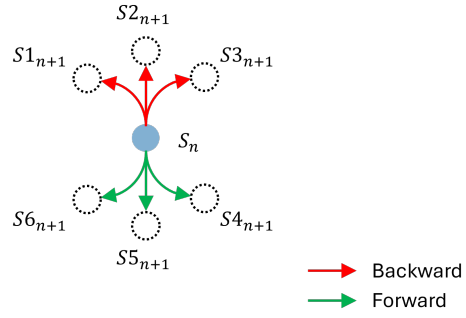
$$s = (x_2, y_2, \theta_{trailer}, \phi) \quad (3.4)$$

where (x_2, y_2) is the position of the truck's rear axle, $\theta_{trailer}$ is the orientation of the trailer, and ϕ is the steering angle. By using this 4 dimension states, it is possible to compute the full state representation of the truck-trailer system and generate feasible paths for parking maneuvers.

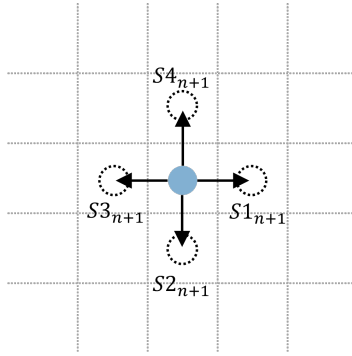
The Hybrid A* algorithm for the truck-trailer parking scenario proceeds as follows, an illustration has shown in Figure 3.6:

The f-cost of a node is calculated using the equation:

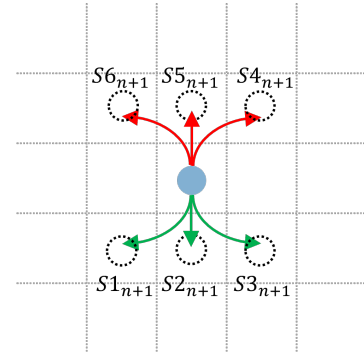
$$f(n) = g(n) + h(n) \quad (3.5)$$



(a) Vehicle potential moves in discretized space.



(b) A star search tree.



(c) Hybrid A* search tree.

Figure 3.5: Comparison of A* and Hybrid A* search trees.

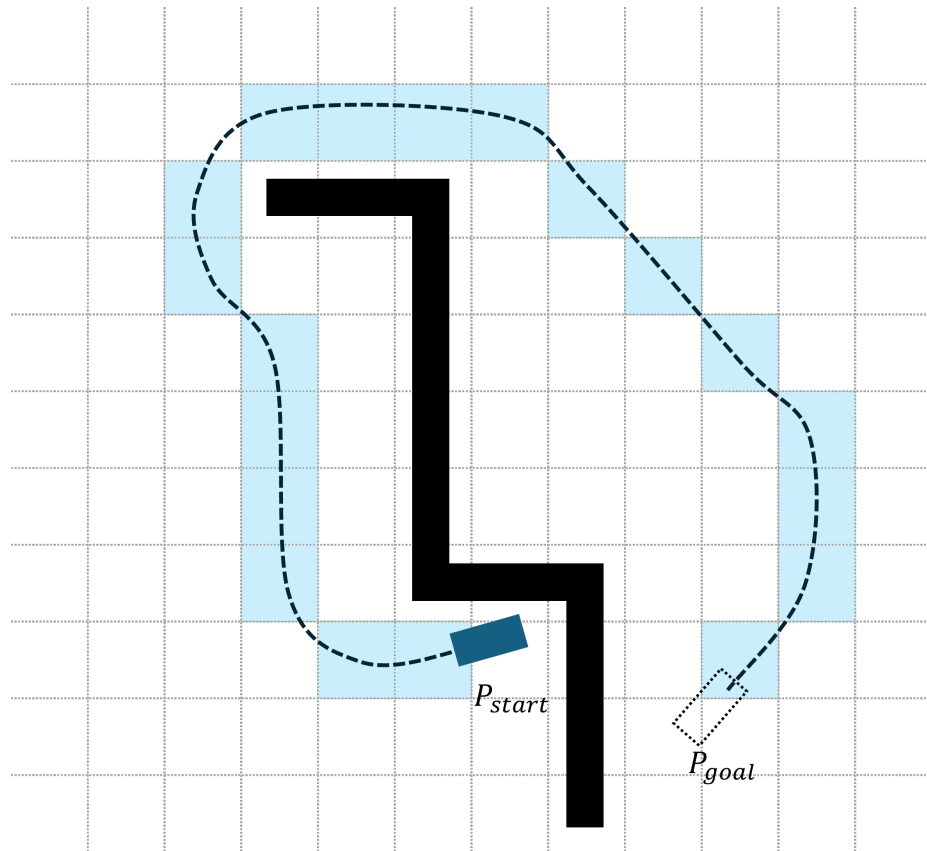


Figure 3.6: Hybrid A* Planning for truck-trailer parking.

Algorithm 3.6: Hybrid A* for truck-trailer parking.

Input: start_state, goal_region, obstacles

Output: Optimal path or failure

```
1. Function HybridAStar(start_state, goal_region, obstacles):
2.   open_list  $\leftarrow$  PriorityQueue()
3.   closed_list  $\leftarrow$  HashTable()
4.   open_list.push(start_state)
5.   while not open_list.empty() do
6.     current_node  $\leftarrow$  open_list.pop()
7.     if current_node in goal_region then
8.       | return ReconstructPath(current_node)
9.     end
10.    successors  $\leftarrow$  GenerateSuccessors(current_node)
11.    foreach successor in successors do
12.      | if CollidesWith(successor, obstacles) then
13.        | Continue;
14.      | end
15.      | if closed_list.contains(successor) and
16.        |   closed_list[successor].cost < successor.cost then
17.        |   Continue;
18.      | end
19.      | successor.f_cost  $\leftarrow$  CalculateFCost(successor)
20.      | open_list.push(successor)
21.    end
22.    closed_list.add(current_node)
23. end
return failure
```

where $g(n)$ is the cost to reach the node from the start, and $h(n)$ is a heuristic estimate of the cost from the node to the goal. In the specific implementation as shown in Figure 3.7, this research introduce a two-phase approach to enhance the efficiency of parking maneuvers:

- **Forward Driving Phase:** The algorithm first attempts to drive the truck-trailer system forward, approaching the general vicinity of the parking spot. This phase uses a simplified heuristic that encourages progress towards the parking area without considering the final orientation.
- **Reeds-Shepp Parking Phase:** Once the system reaches a position from which backward movement becomes necessary, the algorithm switches to a more complex planning mode. It identifies potential points from which a Reeds-Shepp path can generate a collision-free, forward-backward parking trajectory.

The transition between these phases is determined by evaluating the feasibility of Reeds-Shepp paths during the forward driving phase. When a feasible Reeds-Shepp path to the final parking position is found, the algorithm switches to the second phase in current node. Also, the heuristic function $h(n)$ is adjusted for each phase:

$$h(n) = \begin{cases} w_1 \cdot d_{\text{euclidean}}(n, \text{goal}) & \text{for forward phase} \\ w_2 \cdot d_{\text{reeds_shepp}}(n, \text{goal}) & \text{for parking phase} \end{cases} \quad (3.6)$$

where $d_{\text{euclidean}}$ is the Euclidean distance to the goal, $d_{\text{reeds_shepp}}$ is the length of the Reeds-Shepp path to the goal, and w_1, w_2 are weighting factors.

To handle the obstacle avoidance, this research use a hierarchical collision checking approach, and this two accuracy level approach helps maintain efficiency while ensuring safety:

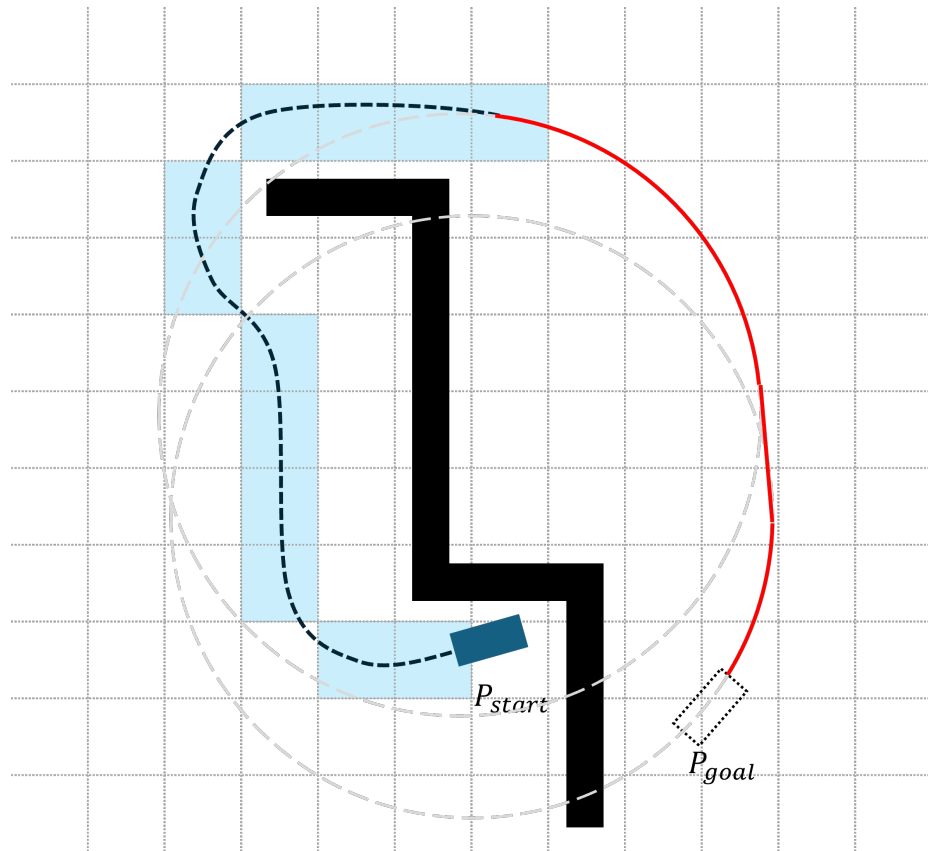


Figure 3.7: Reeds-shepp based Hybrid A* Phases for truck-trailer parking.

- **Bounding Box Check:** A fast, coarse check using bounding boxes is performed to eliminate obviously infeasible paths.
- **Polygon-Based Check:** More detailed collision checking is performed using the actual vehicle shapes to ensure safety.

To conclude, the Hybrid A* algorithm, shows a highly effective planning with complex parking maneuvers for TTRV. By continuously combining forward path exploration with Reeds-Shepp paths for the final parking and adjustment, the algorithm can generate smooth, collision-free trajectories even in tight spaces.

3.5 Kinodynamic RRT Path Planning

Kinodynamic RRT* is a continuously sampling based path planning algorithm that extends the traditional RRT algorithm to handle dynamic constraints. This extension making it particularly suitable for planning trajectories in dynamic environments with the consideration of vehicle physical limitations. By incorporating dynamic constraints, Kinodynamic RRT* can generate smooth and realistic paths that respect the vehicle kinematic limitations, such as maximum acceleration and steering angles.

Similar to Rapidly-Exploring Random Tree Star (RRT*), the core idea in Kinodynamic RRT* is to grow a tree with feasible trajectories in the state space, starting from the initial state and exploring towards the goal state. Unlike traditional RRT, which only considers the configuration space, Kinodynamic RRT* sample future states from the full state space, including both position and velocity components[44]. This comprehensive state sampling allows the algorithm to follow the dynamic behavior of the system more accurately, and generates trajectories with both collision-free and dynamically feasible.

The tree growth process in Kinodynamic RRT* is more complex than in standard RRT* due to the inclusion of dynamics. Each node in the tree represents a full state of the

system, such as position, orientation, velocities, and so on. The process of growing the tree involves the following steps:

- Random Sampling: sample a random state from the entire state space.
- Nearest Neighbor Search: find the nearest node in the existing tree to the sampled state.
- Steering: use a steering function that respects the system's dynamics to generate a trajectory from the nearest node towards the sampled state, which is one difference from the standard RRT algorithm, in which new nodes are created by simple linear interpolation
- Collision Checking: check for collisions with obstacles along the generated trajectory.
- Node Addition: new node is added to the tree if the trajectory is collision-free.

The key difference between RRT and RRT* is pruning and rewiring, which are crucial for achieving asymptotic optimality[45]. In Kinodynamic RRT*, these steps are adapted to handle the dynamic constraints:

- Near Neighbor Search: after adding a new node, the algorithm searches for a set of 'near' neighbors within a certain distance
- Choosing Parent: connect the new node to each near neighbor and chooses the parent that results in the lowest cost-to-come, which is the cost of reaching the new node from the start
- Rewiring: check the new node is reaching near neighbor would help lower the cost; if so, the tree 'rewire' by changing the parent of the near neighbor to the new node.

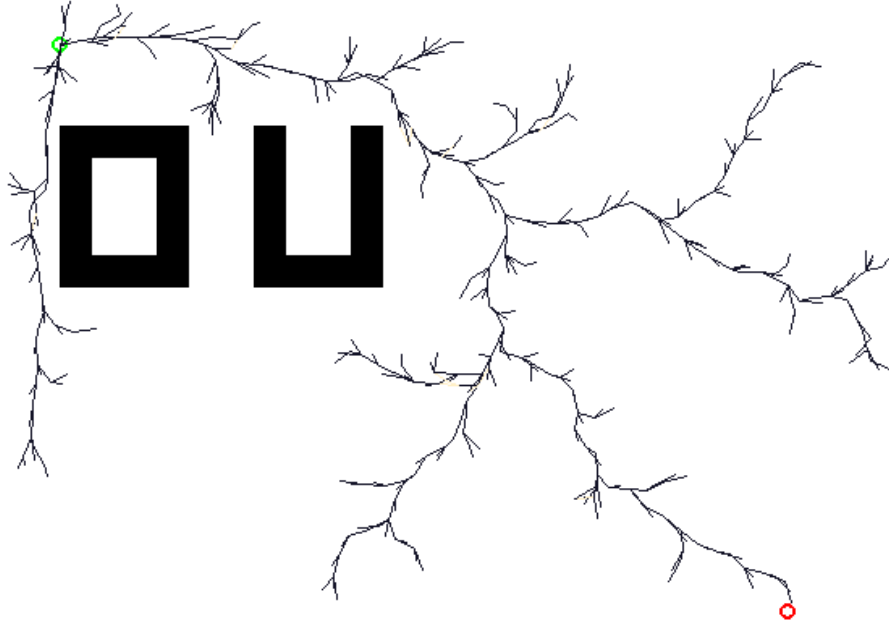


Figure 3.8: The exploration stage of the Kinodynamic RRT* algorithm.

These additional steps allow Kinodynamic RRT* to continuously improve the quality of the tree, converging towards an optimal solution over time. The pruning can make sure better paths are found during old, suboptimal branches of the tree are abandoned for efficiency.

The incorporation of system dynamics in Kinodynamic RRT* presents unique challenges, particularly in the steering and rewiring steps. The steering function must solve a two-point boundary value problem to generate dynamically feasible trajectories, which can be computationally expensive. Similarly, the rewiring step must ensure that any changes to the tree maintain dynamic feasibility. These challenges often necessitate the use of simplified dynamic models or efficient numerical integration techniques to maintain computational tractability.

As shown in section 2.2, the simplified truck-trailer system's state vector s can be defined as:

$$s = (x_{trailer}, y_{trailer}, \theta_{trailer}, \phi, v) \quad (3.7)$$

where $(x_{trailer}, y_{trailer})$ is the position of the trailer, $\theta_{trailer}$ is the orientations of the trailer, ϕ is the truck-trailer angle, and v is the truck's velocity, which is set as a constant in this case to simplify the state space. The full state space of the system includes the truck's position and orientation, the trailer's position and orientation, and the truck-trailer angle, which can be computed using the TTRV physical parameters like L_1, L_2, L_3 .

At the same time, for the reverse path following problem, the system control input u can be defined as:

$$u = (\delta_{truck}) \quad (3.8)$$

where δ_{truck} is the steering angle of the truck. The Kinodynamic RRT* algorithm for the truck-trailer parking is shown in algorithm 3.7.

The key challenge in Kinodynamic RRT* for truck-trailer systems is from the propagate function, which must accurately model the system dynamics. As shown in the previous chapter, the motion of the truck-trailer system can be described by Equations (2.9) to (2.11) and (2.13) to (2.15). To evaluate the occurrence of jack-knifing, it is important to consider the rate of change of the host-trailer angle in . The propagate function in the algorithm uses these equations to generate new states. It takes the current state and a control input $u = \delta_{truck}$, which is the steering angle of truck, and integrates these equations over a short time step to produce the next state.

The cost function for evaluating paths typically considers path length, smooth, distance to obstacles, and a penalty for large truck-trailer angles to avoid jack-knifing:

Algorithm 3.7: Kinodynamic RRT* for truck-trailer parking.

Input: start_state, goal_region, obstacles, max_iterations

Output: Optimal path or failure

```
1. Function KinodynamicRRTStar(start_state, goal_region, obstacles,  
   max_iterations):  
2.   tree  $\leftarrow$  Initialize(start_state)  
3.   for  $i = 1$  to max_iterations do  
4.     s_rand  $\leftarrow$  SampleState()  
5.     s_nearest  $\leftarrow$  Nearest(tree, s_rand)  
6.     u  $\leftarrow$  SelectInput(s_nearest, s_rand)  
7.     s_new  $\leftarrow$  Propagate(s_nearest, u)  
8.     if CollisionFree(s_new, obstacles) and not JackKnife(s_new) then  
9.       s_min  $\leftarrow$  ChooseParent(tree, s_new)  
10.      tree.AddVertex(s_new)  
11.      tree.AddEdge(s_min, s_new)  
12.      Rewire(tree, s_new)  
13.    end  
14.    if s_new in goal_region then  
15.      return ExtractPath(tree, s_new)  
16.    end  
17.  end  
18.  return failure
```

$$J = \int_0^T (w_1 + w_2|\kappa(t)| + w_3d_{obs}^{-1}(t) + w_4|\phi(t)|)dt \quad (3.9)$$

where w_1, w_2, w_3, w_4 are weighting factors, $\kappa(t)$ is the path curvature, $d_{obs}(t)$ is the distance to the nearest obstacle, and $\phi(t)$ is the truck-trailer angle at time t . Moreover, the algorithm includes a jackKnife check in addition to collision detection, which can ensure the generated trajectories not only avoiding obstacles but also maintain a safe truck-trailer angle throughout the maneuver.

3.6 Chapter Conclusion

In this chapter, several path planning algorithms for autonomous vehicle parking control is introduced in details, including Dubins paths, Reeds-Shepp paths, Hybrid A*, and Kinodynamic RRT*. All these algorithms have been modified to fit the state space model of TTRV. As shown in previous sections, the geometric path planning algorithms, including Dubins and Reeds-Shepp paths, are fundamental methods which is suitable for generating feasible parking trajectories for non-holonomic vehicles. The grid based search algorithm like Hybrid A* can combines discrete and continuous state space representations to handle complex parking maneuvers, while the sampling based Kinodynamic RRT* algorithm extends the traditional RRT* algorithm to consider dynamic constraints, which allows generating smooth and realistic paths for parking in dynamic environments.

All these algorithms have their unique advantages and challenges, and the choice of algorithm depends on the specific requirements of the system or customer needs. In the next chapter, these algorithms is implemented in a simulation environment and tested by integrating with different control methods, to evaluate their performance in Unity simulation environment.

CHAPTER FOUR

TRADITIONAL CONTROL METHODS FOR TTRV

The development of effective autonomous vehicle's control methods is a crucial challenge for intelligent transportation systems, and as vehicles becoming autonomous, the ability to perform complex driving maneuver without human intervention has become a popular research topic[46].

This chapter introduces several popular traditional control methods for TTRV system, ranging from vehicle state stabilizing to vehicle path following. In section 4.1, the pole placement and gain scheduler controllers is analysed, and then the Lyapunov-based control methods is introduced and applied to the simulation of TTRV in section 4.2. section 4.3 investigates the application of LQR control to TTRV, especially for the path following tasks. And section 4.4 introduced MPC controller, and explores how MPC's prediction capability can integrate the TTRV constraints to solve complex path following tasks. All control strategy are modified to fit the TTRV state space, and the simulation results are presented to demonstrate the effectiveness of the control methods. This chapter's simulation result also serves as the benchmark for the following chapter's RL control methods comparison, to evaluate the performance of modern advanced neural network based control algorithms.

4.1 Pole Placement and Gain Scheduler

The pole placement is developed in the 1960s as part of modern control theory[21], and it is considered to be a simple yet powerful method for stabilizing linear systems and fulfilling desired closed-loop dynamics. The method is derived from the

concept that a system's dynamic behavior is characterized by the poles from its transfer function. By properly placing poles of the transfer function, engineers can shape system's response, including stability, settling time, and overshoot. For TTRV reverse driving, this approach can be applied to design a controller which ensures stable and efficient steering maneuvering. To implement pole placement for TTRV reverse driving, the system model needs to be linearized around the Φ angle.

Consider the linearization around truck-trailer angle $\phi = 0$ and steering angle $\delta = 0$, the state-space equation for Equation 2.15 can then be linearized and expressed as:

$$\Delta\dot{\phi} = \frac{V}{L_3}\Delta\phi + \frac{V}{L_1}\left(1 + \frac{L_2}{L_3}\right)\Delta\delta \quad (4.1)$$

From this equation, the system can be identified to follow the standard state-space form $\dot{x} = Ax + Bu$:

$$A = \frac{V}{L_3}, \quad B = \frac{V}{L_1}\left(1 + \frac{L_2}{L_3}\right), \quad x = \Delta\phi, \quad u = \Delta\delta \quad (4.2)$$

where x represents the state vector of deviations from the operating point, u is the input vector, and A and B are the system state and input matrices, respectively.

The pole placement theorem states that for a controllable system, arbitrary poles can be assigned to the closed-loop system through state feedback[47]. Assuming a zero reference $r = 0$, the control law for state feedback is given by:

$$u = -Kx \quad (4.3)$$

where K is the feedback gain matrix. The closed-loop system dynamics then become:

$$\begin{aligned} \dot{x} &= Ax + B(-Kx) = (A - BK)x \\ y &= Cx \end{aligned} \quad (4.4)$$

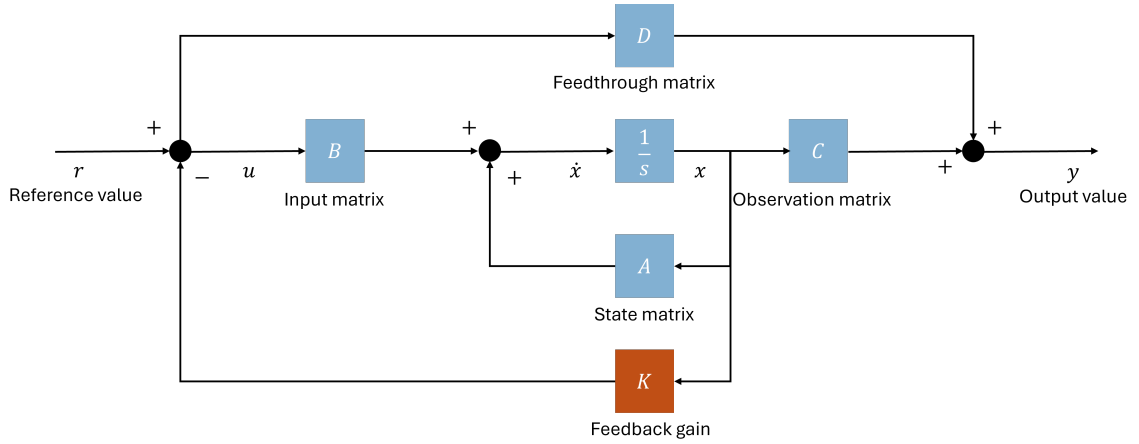


Figure 4.1: Pole placement diagram for TTRV reverse parking.

The eigenvalues of the new state transfer matrix ($A - BK$) determine the poles of the closed-loop system. By selecting eigenvalues of this new matrix, desired system characteristics such as stability, response speed, and damping ratio can be achieved.

For the TTRV system, the selection of poles should have several considerations. Firstly, all poles should have negative real parts to ensure systems' asymptotic stability. Also, the magnitude of these real parts affects the speed of the system response, which allows tuning of the controller's responsiveness. Complex conjugate poles introduce oscillatory behavior, which should be minimized to achieve smooth parking maneuvers for glsttrv[47]. Additionally, the pole locations must be within the physical limitations of the TTRV system, such as maximum steering angle or jackknifing avoidance. After balancing these considerations, a pole placement strategy can be designed to provide optimal performance within the system's constraints, ensuring stable and efficient control of the truck-trailer system during reverse parking maneuvers.

The characteristic equation of this closed-loop system is:

$$\det(sI - (A - BK)) = 0 \quad (4.5)$$

Once the desired poles are selected, the feedback gain matrix K can be computed using techniques such as Ackermann's formula or MATLAB's place function. The Ackermann's formula simplifies the computation of K^T as follows:

$$\begin{aligned} K^T &= \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix} \mathcal{C}^{-1} \Delta_{\text{new}}(A) \\ &= [0 \ 0 \ \dots \ 1] [B \ AB \ A^2B \ \dots \ A^{n-1}B]^{-1} p(A) \end{aligned} \quad (4.6)$$

where $\mathcal{C} = [B \ AB \ A^2B \ \dots \ A^{n-1}B]$ is the controllability matrix of the system, $p(A) = \Delta_{\text{new}}(A)$ is the desired characteristic polynomial evaluated at matrix A . Then, the generated pole placement can be applied to the TTRV to maneuver the trailer along the desired trajectory with stability. The following MATLAB simulations are conducted to demonstrate the effectiveness of the pole placement controller to stabilize the truck trailer angle Φ during reverse driving.

In the Figure 4.2, the right plot shows the desired reference value, while the left plot displays the simulation results of the TTRV system under pole placement control. The system's response closely follows the desired reference value, demonstrating the effectiveness of the pole placement controller in guiding the TTRV vehicle through the reverse driving. The initial offset represents an initial ϕ angle, a small delay in response can be observed, and asymptotic stable steady-state tracking during the plateau phase.

In the Figure 4.3, the closed-loop response of the pole placement controller during a simulated U-turn maneuver is observed. The graph shows the articulation angle ϕ over time, comparing the controller's response (blue line) to the reference trajectory (orange line). The U-turn is represented by an extended curve, with the angle reaching

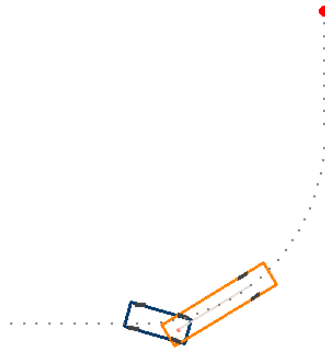
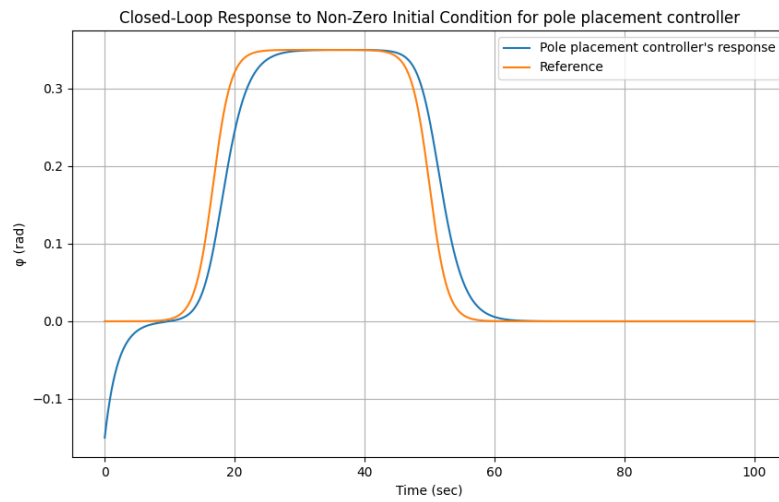


Figure 4.2: Pole placement simulation results for a 90 degree turn.

approximately 0.4 radians for about 20 seconds. The controller's response closely tracks the reference, demonstrating effective performance. These two simulations show the pole placement controller's capability in handling complex maneuvers while maintaining stability.

While pole placement is effective for linear systems, the TTRV system's nonlinearity poses challenges for direct application. To address this, gain scheduling

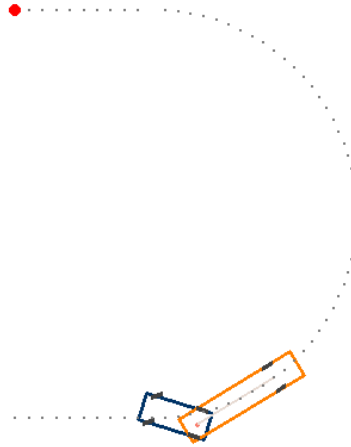
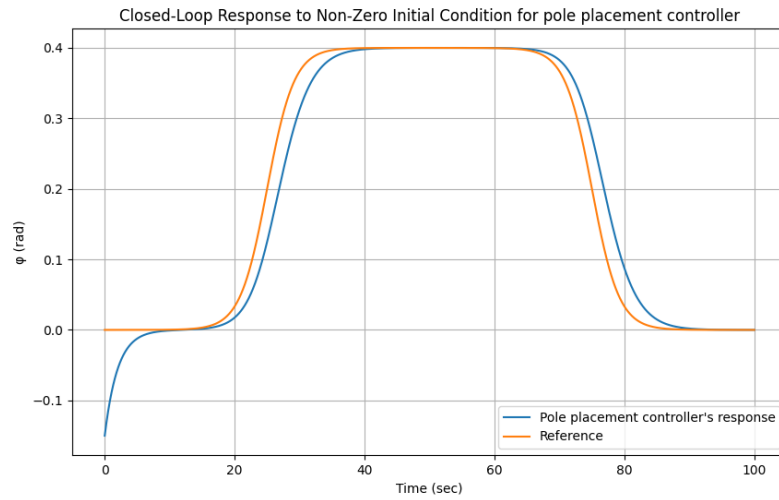


Figure 4.3: Pole placement simulation results for a U turn.

extends linear control methods to nonlinear systems by adjusting controller parameters based on the current operating condition.

For the TTRV system, the nonlinearity primarily arises from the varying trailer angle ϕ . A gain scheduling approach involves linearizing the system around multiple operating points, designing pole placement controllers for each linearized model, and interpolating between these controllers based on the current ϕ . The gain scheduler

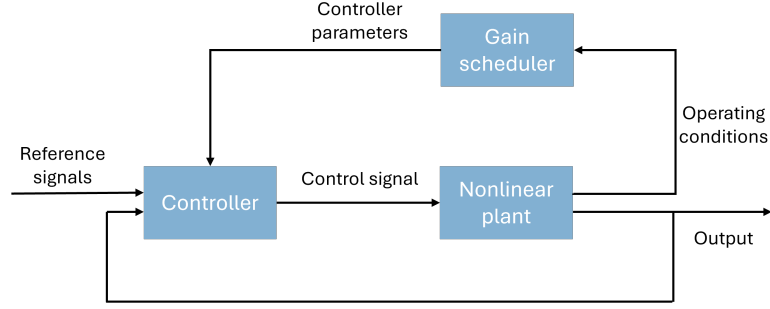


Figure 4.4: Gain scheduling diagram.

continuously monitors ϕ and adjusts the controller parameters accordingly, allowing effective control across the entire operating range of the TTRV system, as shown in Figure 4.4.

Mathematically, this can be represented as:

$$K(\phi) = f(\phi, K_1, K_2, \dots, K_n) \quad (4.7)$$

where $K(\phi)$ is the gain matrix as a function of ϕ , and $K_1, K_2, \dots, K_n$ are gain matrices designed for specific operating points.

In conclusion, this chapter has explored the application of pole placement and gain scheduling techniques to the TTRV system. Although pole placement provides a foundation for linear control, gain scheduler method extends its applicability to the nonlinear dynamics of the truck-trailer system. By combining these methods, robust control can be achieved throughout the TTRV system's operating range under different conditions stable and efficient.

4.2 Lyapunov Based Controller

Lyapunov-based controller is a feedback control method proposed by Aleksandr Lyapunov's, and it is from his work on stability analysis[48][49]. This method introduces key stability concepts illustrated in Figure 4.5: a system is Lyapunov stable if solutions near an equilibrium point remain in its vicinity indefinitely. Asymptotic stability is a stronger stability condition, it occurs when solutions converge to the equilibrium over time. For practical applications like TTRV control, exponential stability is often desired, which can guarantee a minimum convergence rate to the equilibrium point, as illustrated in Figure 4.6. These stability concepts are pretty important for nonlinear systems like the truck-trailer, where traditional linear control methods may not be sufficient enough or fail to provide the desired performance.

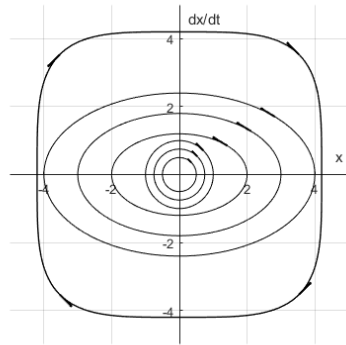
The Lyapunov theory uses functions similar to energy in physical systems, which is why it's often explained using energy analogies. These functions decrease along system trajectories for stable systems. One of the commonly used Lyapunov function candidate based on the error between the current and desired trailer angle can be described as following:

$$V = \frac{1}{2}e^2 \quad (4.8)$$

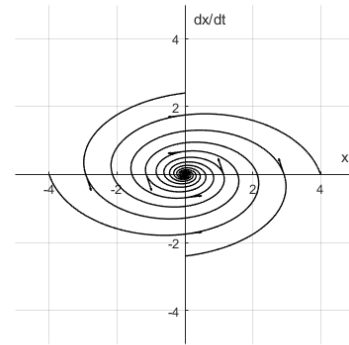
In the feedback control problem for TTRV, the error e can be defined as:

$$e = \phi - \phi_{ref} \quad (4.9)$$

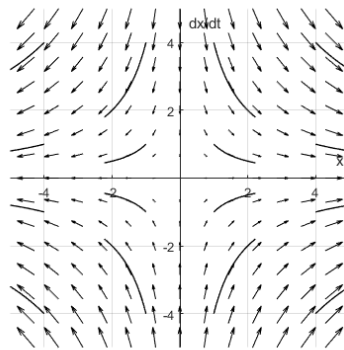
where ϕ from Equation 2.15 represents the current trailer angle and ϕ_{ref} is the desired truck trailer angle reference. The time derivative of the error is given by:



(a) Lyapunov stable.



(b) Asymptotically stable.



(c) Unstable.

Figure 4.5: Phase portrait for several stabilities.

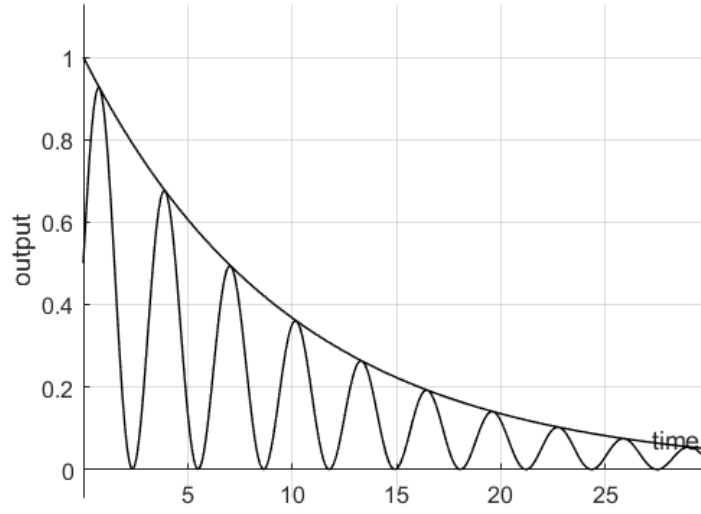


Figure 4.6: Time domain plot for exponentially stable system.

$$\dot{e} = \dot{\phi} \quad (4.10)$$

To ensure asymptotic stability, the Lyapunov function must satisfy the following conditions[48]:

1. $V(0) = 0$
2. $V(x(t)) > 0$ for $x \neq 0$
3. $\dot{V}(x(t)) \leq 0$ for all $t > 0$

The proposed Lyapunov function in Equation 4.8 inherently satisfies the first two conditions. To satisfy the third condition, the time derivative of V needs to be tested:

$$\dot{V} = e\dot{e} = e\dot{\phi} \quad (4.11)$$

Then substitute the expression for $\dot{\phi}$ from Equation 2.15, the following equation is obtained:

$$\dot{V} = e \left(-\frac{v}{L_3} \sin \phi + \frac{v}{L_1} \left(1 + \frac{L_2 \cos \phi}{L_3} \right) \tan \delta \right) \quad (4.12)$$

To ensure stability, a control law to make $\dot{V}$ negative semi-definite, and one selection is:

$$\dot{V} = -k \cdot e^2 \leq 0 \quad (4.13)$$

In this case, the $\dot{\phi}$ can be expressed as:

$$\dot{\phi} = -k \cdot e \quad (4.14)$$

Then $\dot{\phi}$ new equation can be used to replace Equation 4.13. A control law of the following form can then be proposed:

$$\tan(\delta) = \frac{-k \cdot e - \frac{V}{L_3} \cdot \sin(\phi)}{\frac{V}{L_1} \cdot \left(1 + \frac{L_2}{L_3} \cdot \cos(\phi) \right)} \quad (4.15)$$

The control law in Equation 4.15 can be implemented directly to steer the TTRV as shown in Figure 4.7, and the final steering angle δ can be computed as:

$$\delta = \arctan \left(\frac{-k \cdot e - \frac{V}{L_3} \cdot \sin(\phi)}{\frac{V}{L_1} \cdot \left(1 + \frac{L_2}{L_3} \cdot \cos(\phi) \right)} \right) \quad (4.16)$$

To evaluate the performance of the Lyapunov-based controller, a simulation is conducted to observe the system's response to a step input. As shown in Figure 4.8, the initial ϕ offset is 0.1 radians, and the controller successfully guides the trailer to the

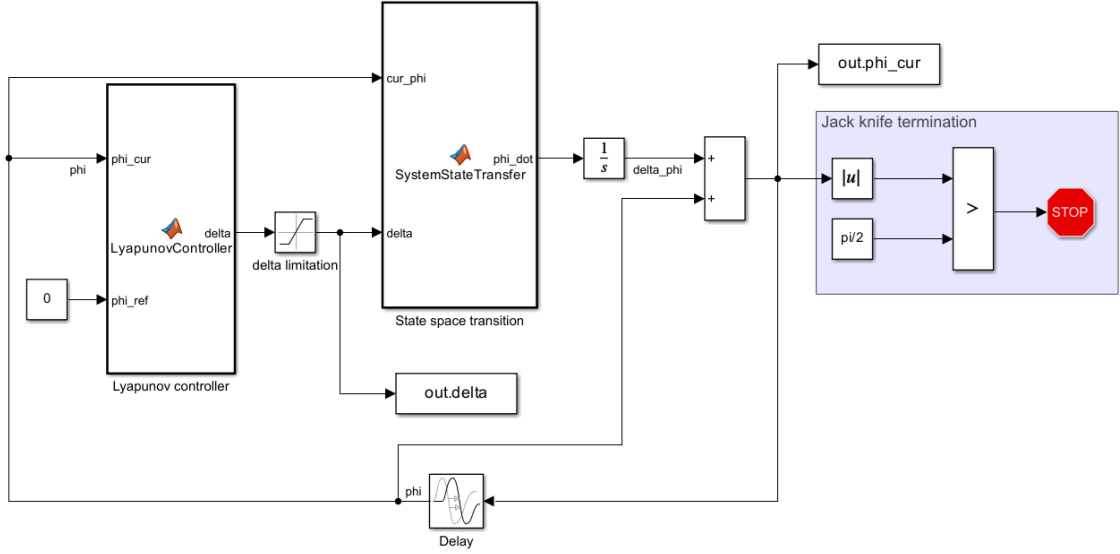


Figure 4.7: Lyapunov-based controller Simulink diagram.

desired angle, demonstrating the effectiveness of the Lyapunov-based controller for TTRV reverse ϕ correction and jackknifing avoidance.

In conclusion, the Lyapunov-based controller provides a systematic approach to stability and direct system parameters integration for TTRV reverse driving control. However, similar to other feedback control methods, one of the challenge is the gain tuning and handling complex scenarios like trajectory following and obstacle avoidance, which impacts its real-time implementation on embedded systems. Although current industry products often apply more established methods like LQR or MPC because of the flexibility of these methods, Lyapunov theory is still valuable for system design, especially it is incorporated into hybrid control methods to improving the system's stability.

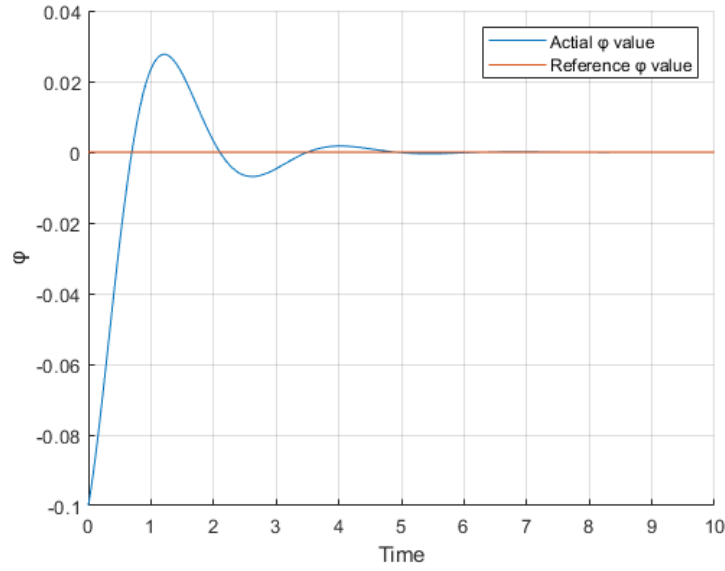


Figure 4.8: Lyapunov-based controller Simulation results.

4.3 Linear Quadratic Regulator Controller

LQR is an optimal control method designed for linear systems with quadratic cost functions, which goal is to minimize a cost function and at the same time regulating the system to a desired state. The LQR algorithm computes a feedback gain matrix K , and apply this gain into the control law $u = -Kx$, which optimize the system control effort and state deviation. As shown in section 2.4, for a controllable system $(\mathcal{A}, \mathcal{B})$, it is possible to manipulate the eigenvalues of the closed-loop system $(\mathcal{A} - \mathcal{B}K)$ through this state feedback loop. LQR is particularly useful for the Multiple Input Multiple Output (MIMO) systems and help system to improve its inherent robustness with guaranteed gain and phase margin[50].

The LQR is designed based on the cost function, and for a infinite-horizon, continuous-time system described in Equation 2.26, the cost function is defined as:

$$J = \int_0^{\infty} \left(x^T Q x + u^T R u + 2x^T N u \right) dt \quad (4.17)$$

where the matrices Q and R weight the cost of state deviation and actuation cost respectively, moreover, the matrix Q is positive semi-definite, and R is positive definite; these two matrices are diagonal matrices, and the diagonal elements can be tuned to change the relative importance of the control objectives. The weight matrix is useful to make choices to optimize the control law, for example, a larger value in a particular diagonal entry of Q matrix means that deviations in the corresponding state will be penalized more heavily in the cost function, which leads the controller to put more effort to regulate this particular state. When the Q matrix is set to be zero, the controller will not attempt to regulate the state at all, only focus on minimizing control effort. The matrix R determines how much the control actions are penalized in the cost function. A larger value in a diagonal entry of R indicates the corresponding control action will be penalized more, which discourages aggressive control actions and can be used to generate smoother control signals.

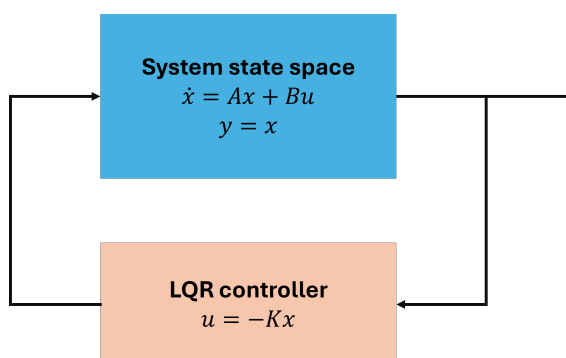


Figure 4.9: LQR controller schematic diagram.

As shown in Figure 4.9, the LQR control law $u = -Kx$ is designed to minimize the cost function $J = \lim_{t \rightarrow \infty} J(t)$, and the analytical solution for the optimal controller gain K can be computed following:

$$K = R^{-1} (B^T P + N^T) \quad (4.18)$$

where the variable P is computed by solving the continuous time algebraic Riccati equation:

$$A^T P + PA - (PB + N)R^{-1} (B^T P + N^T) + Q = 0 \quad (4.19)$$

which can be also written as:

$$\mathcal{A}^T P + P\mathcal{A} - PBR^{-1}B^T P + \mathcal{Q} = 0 \quad (4.20)$$

where $\mathcal{A} = A - BR^{-1}N^T$ and $\mathcal{Q} = Q - NR^{-1}N^T$.

To apply LQR control to the TTRV system, the trajectory is firstly computed, which has been introduced in chapter THREE. The reference trajectory is then used to compute the reference states error, which is the difference between the current states and the reference states as shown in Figure 4.10, and the state space model can be formulated as:

$$\begin{bmatrix} \Delta \dot{\theta}_1 \\ \Delta \dot{\theta}_2 \\ \Delta \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{v}{L_3} & -\frac{v}{L_3} & 0 \\ 0 & V & 0 \end{bmatrix} \begin{bmatrix} \Delta \theta_1 \\ \Delta \theta_2 \\ \Delta y \end{bmatrix} + \begin{bmatrix} \frac{v}{L_1} \\ -\frac{L_2}{L_1 L_3} v \\ 0 \end{bmatrix} \delta \quad (4.21)$$

where the reference states error consist of $\Delta \theta_1$, $\Delta \theta_2$, and Δy , which is the angle between truck and the reference point tangent, the angle between trailer and reference point tangent, and the lateral distance between trailer and refernece point respectively.

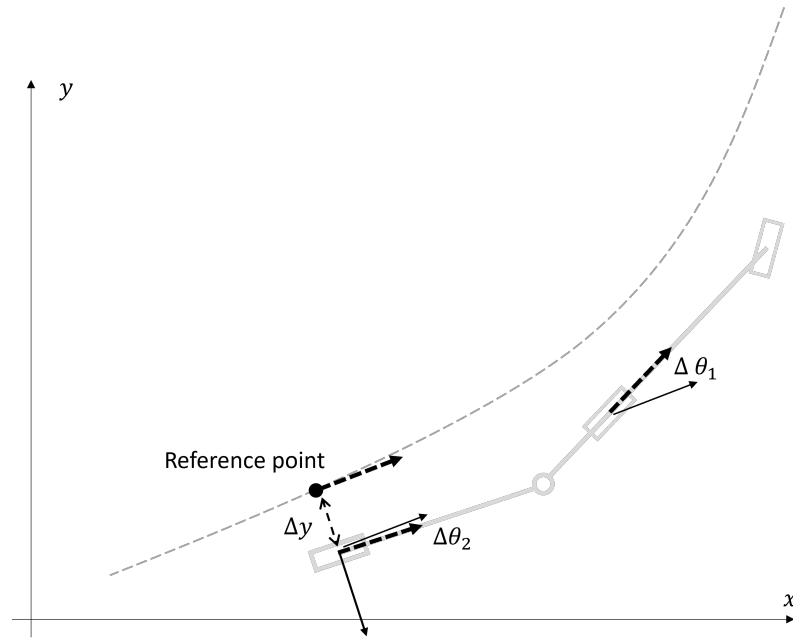


Figure 4.10: LQR controller reference states error.

To compute the optimal gain of the TTRV controller, MATLAB provides built-in functions which can solve Riccati equation efficiently by providing the system state space model, weight matrix Q and R , the code is as following:

```

1
2 % Linearized State Space
3 A = [0      0      0;
4      vx./L3 -vx./L3 0;
5      0      vx     0];
6
7 B = [vx./L1;
8      -L2*vx ./ (L1*L3);
9      0];

```

```

10
11 C = eye(3);
12 D = zeros(3, 1);
13
14 % state states x = [\theta1, \theta2, y_err]
15 sys = ss(A, B, C, D);
16
17 % State weight matrix
18 Q = [1/(deg2rad(2).^2)    0    0;
19      0    1/(deg2rad(2).^2)    0;
20      0    0    1/(1.^2)];
21 % Input weight matrix for steering angle
22 R = 1 / (deg2rad(steer_max).^2);
23
24 % Output K is the LQR gain
25 [K,S,P] = lqr(sys, Q, R)

```

The computed LQR gain matrix K can then be used to compute the control input $u = -Kx$ in every step.

The simulation of the LQR controller for TTRV reverse path following control are shown in Figure 4.11, the grey line illustrates the trajectory generated by Dubins path planner, and the truck starts to maneuver the trailer with its initial states $(40, 30, -\frac{\pi}{2})$ and ends with final states $(0, 0, 0)$. The stacked simulation frames show the truck-trailer system's movement during the reverse parking maneuver, with the trailer following the desired path closely as shown in Figure 4.12, which is the comparison between trailer's history trajectory and the reference trajectory.

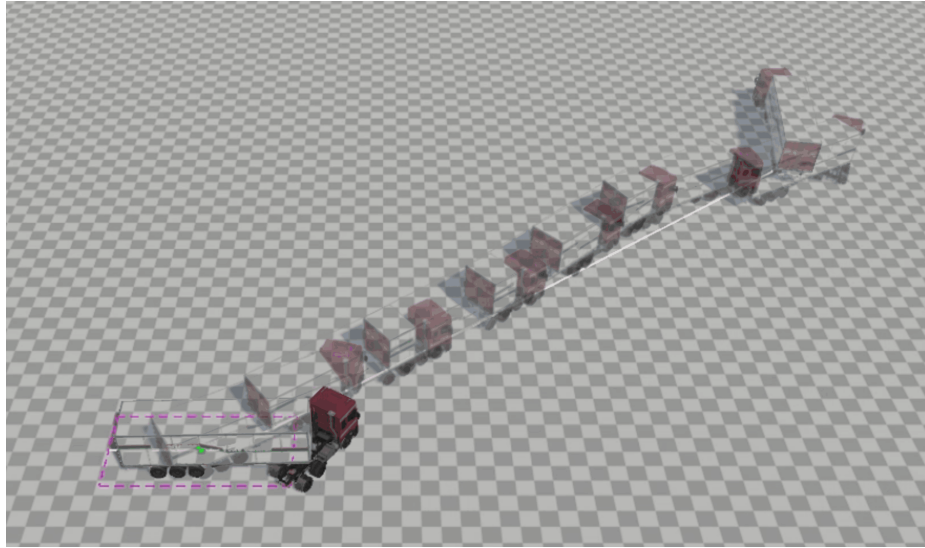


Figure 4.11: The stacked simulation frames of using Dubins and LQR controller for TTRV reverse driving control.

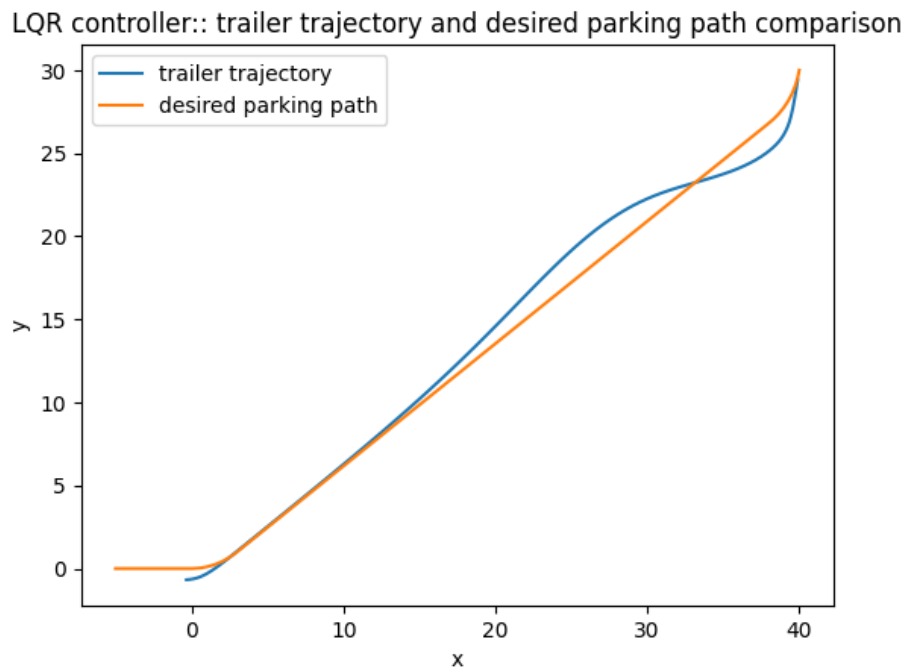


Figure 4.12: LQR controller: trailer trajectory and desired parking path comparison.

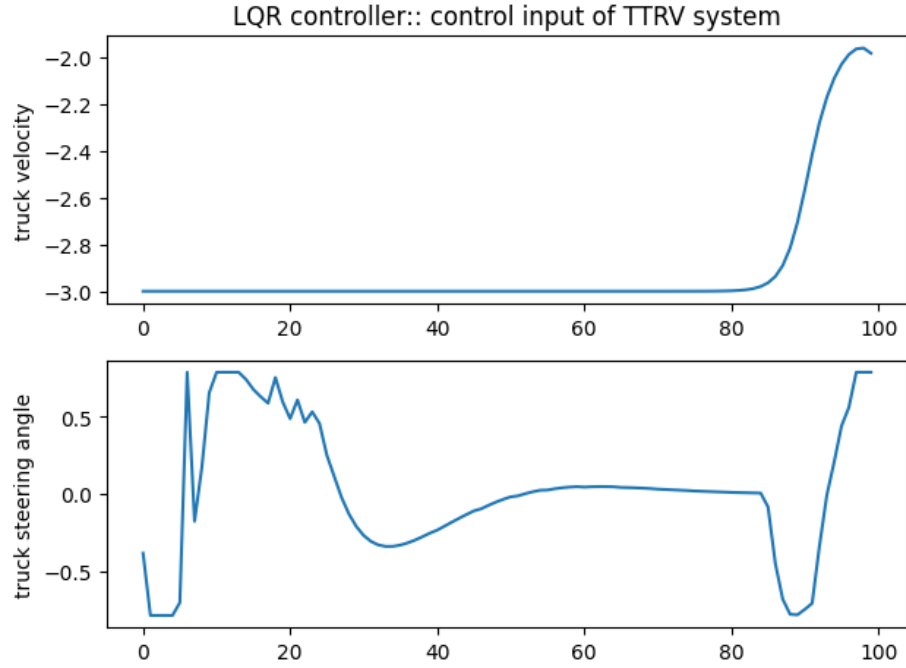


Figure 4.13: LQR controller: control input of TTRV system.

Figure 4.13 shows the control inputs for TTRV system with LQR controller. The top figure shows the truck velocity, which remains constant at about -3 m/s for most of the simulation before gradually decreasing to -2 m/s near the end, and the bottom graph represents the truck steering angle through the parking process.

Figure 4.14 shows two evaluation parameters for the LQR. The top figure shows the trailer's lateral offset from the reference trajectory, which is smaller than 1.5m through the parking process. The bottom figure shows the Φ difference, representing the heading angle of the trailer. A relatively large deviation happened during the 90-degree turn segment, but the controller quickly corrected it, and the trailer's heading angle closely followed the reference trajectory.

In conclusion, this section introduced the application of LQR to the TTRV system. LQR provides an optimal control solution for linear systems by minimizing a quadratic

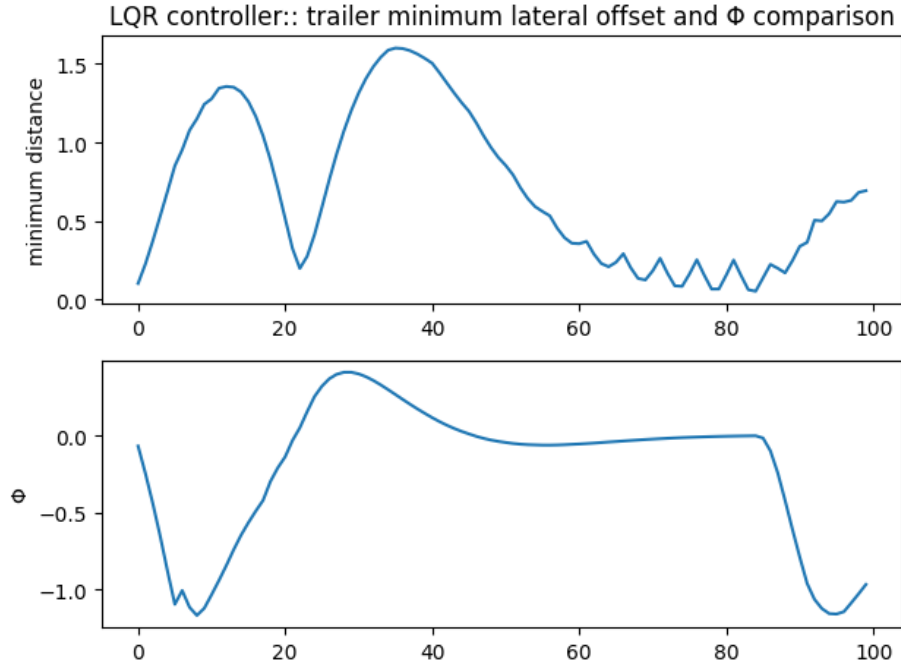


Figure 4.14: LQR controller: trailer minimum lateral offset and steering angle comparison.

cost function, balancing control effort and state regulation. Its ability to handle MIMO systems and inherent robustness make it one of the most popular control methods for current ADAS systems. The simulation results show the effectiveness of the LQR controller in guiding the trailer along the desired path, which proves that the LQR controller to be a robust and efficient solution for TTRV reverse parking control, with the consideration of safety in complex scenarios.

4.4 Model Predictive Controller

Model Predictive Control can be seen as a finite horizon, constrained version of LQR for linear systems. Similar to LQR, it is designed to optimize a quadratic cost function that balances state regulation and control effort. However, MPC extends the

concept by operating on a finite prediction horizon with the consideration of system constraints, by performing finite-horizon optimization for the system model at each time step [51].

The main idea of MPC involves using a model of the system to predict its future behavior over a given horizon. Based on the predicted states, the control inputs are determined by minimizing the cost function, similar with LQR's idea, the cost function considers both state errors and control efforts:

$$J = \sum_{k=0}^{N-1} \left(x(k)^T Q x(k) + u(k)^T R u(k) \right) + x(N)^T P x(N) \quad (4.22)$$

where J is the cost to be minimized, N is the prediction horizon, $x(k)$ is system states at time step k , $u(k)$ is control inputs at time step k , Q and R are weight matrix for system states and control inputs.

Unlike LQR, which computes a static feedback gain, MPC solves this optimization problem at each step, which allows the controller to adapt to changing conditions and handle constraints on both states and inputs, and this feature makes MPC works very well for complex systems with dynamic constraints such as TTRV.

At every time step t_k , the system is being sampled, and a control strategy is derived for the control horizon $[t_k, t_k + N_C]$ to minimize the deviation from the desired trajectory over the prediction horizon $[t_k, t_k + N_P]$. Beyond N_C control moves, the control input remains constant in the rest of the prediction horizon. The system's future states is predicted using the state space model, and different optimization algorithms including Quadratic Programming can be applied to minimizes the control cost. Only the first control input from the control sequence is used at each time step t_k . This step is then repeated in every cycle, and the principle of the MPC algorithm is shown in Figure 4.15.

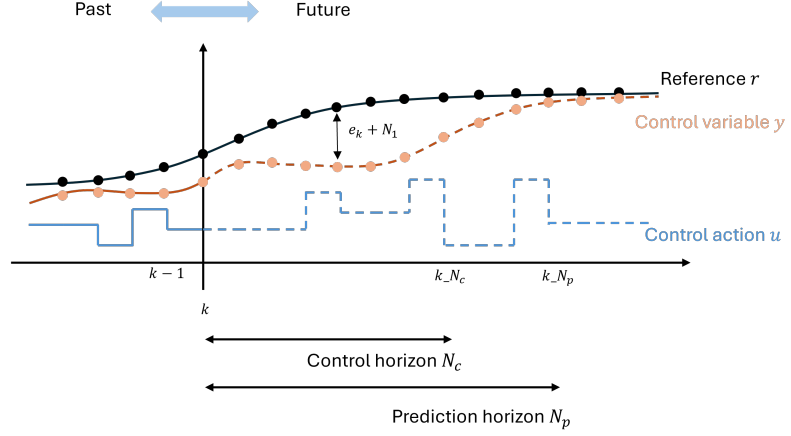


Figure 4.15: Function principle of a model-based predictive.

To describe the TTRV state space transfer function, a new state variable v_1 is added for referencing beyond the Equations (2.9) to (2.11) and (2.13) to (2.15), and then the state vector becomes $X = [x_1, y_1, \theta_1, x_2, y_2, \theta_2, \phi, v]$, and the input states change from $U = [v, \delta]$ to $U = [a, \delta]$ where a is the acceleration of the truck vehicle. The new state representation is as following:

$$\begin{aligned}
 \dot{x}_1 &= f_1(X, U) = v \cos \theta_1 \\
 \dot{y}_1 &= f_2(X, U) = v \sin \theta_1 \\
 \dot{\theta}_1 &= f_3(X, U) = \frac{v}{L_1} \tan \delta \\
 \dot{x}_2 &= f_4(X, U) = v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \cos \theta_2 \\
 \dot{y}_2 &= f_5(X, U) = v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \sin \theta_2 \\
 \dot{\theta}_2 &= f_6(X, U) = -v \left(\frac{\sin \phi}{L_3} + \frac{L_2}{L_1 L_3} \cos \phi \tan \delta\right) \\
 \dot{\phi} &= f_7(X, U) = -\frac{v}{L_3} \sin \phi - \frac{v}{L_1} \left(1 + \frac{L_2 \cos \phi}{L_3}\right) \tan \delta \\
 \dot{v}_{11} &= f_8(X, U) = a
 \end{aligned} \tag{4.23}$$

To get the Jacobian matrix A' of the TTRV system, compute the partial derivatives of state transfer function f_1 to f_8 with respect to each elements in the state vector X :

$$A' = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial y_2} & \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_1}{\partial \phi} & \frac{\partial f_1}{\partial v_1} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial y_2} & \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \phi} & \frac{\partial f_2}{\partial v_1} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial y_1} & \frac{\partial f_3}{\partial \theta_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial y_2} & \frac{\partial f_3}{\partial \theta_2} & \frac{\partial f_3}{\partial \phi} & \frac{\partial f_3}{\partial v_1} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial y_1} & \frac{\partial f_4}{\partial \theta_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial y_2} & \frac{\partial f_4}{\partial \theta_2} & \frac{\partial f_4}{\partial \phi} & \frac{\partial f_4}{\partial v_1} \\ \frac{\partial f_5}{\partial x_1} & \frac{\partial f_5}{\partial y_1} & \frac{\partial f_5}{\partial \theta_1} & \frac{\partial f_5}{\partial x_2} & \frac{\partial f_5}{\partial y_2} & \frac{\partial f_5}{\partial \theta_2} & \frac{\partial f_5}{\partial \phi} & \frac{\partial f_5}{\partial v_1} \\ \frac{\partial f_6}{\partial x_1} & \frac{\partial f_6}{\partial y_1} & \frac{\partial f_6}{\partial \theta_1} & \frac{\partial f_6}{\partial x_2} & \frac{\partial f_6}{\partial y_2} & \frac{\partial f_6}{\partial \theta_2} & \frac{\partial f_6}{\partial \phi} & \frac{\partial f_6}{\partial v_1} \\ \frac{\partial f_7}{\partial x_1} & \frac{\partial f_7}{\partial y_1} & \frac{\partial f_7}{\partial \theta_1} & \frac{\partial f_7}{\partial x_2} & \frac{\partial f_7}{\partial y_2} & \frac{\partial f_7}{\partial \theta_2} & \frac{\partial f_7}{\partial \phi} & \frac{\partial f_7}{\partial v_1} \\ \frac{\partial f_8}{\partial x_1} & \frac{\partial f_8}{\partial y_1} & \frac{\partial f_8}{\partial \theta_1} & \frac{\partial f_8}{\partial x_2} & \frac{\partial f_8}{\partial y_2} & \frac{\partial f_8}{\partial \theta_2} & \frac{\partial f_8}{\partial \phi} & \frac{\partial f_8}{\partial v_1} \end{bmatrix} \quad (4.24)$$

The result of the partial derivatives is computed below, where the partial derivatives not listed are equal to zero:

$$\begin{aligned}
\frac{\partial f_1}{\partial \theta_1} &= -v_1 \sin \theta_1 \\
\frac{\partial f_1}{\partial v_1} &= \cos \theta_1 \\
\frac{\partial f_2}{\partial \theta_1} &= v_1 \sin \theta_1 \\
\frac{\partial f_2}{\partial v_1} &= \sin \theta_1 \\
\frac{\partial f_3}{\partial v_1} &= \frac{\tan \delta}{L_1} \\
\frac{\partial f_4}{\partial \theta_2} &= v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \delta \tan \phi\right) (-\sin \theta_2) \\
\frac{\partial f_4}{\partial \phi} &= v(-\sin \phi) \left(1 - \frac{L_2}{L_1} \tan \delta \tan \phi\right) \cos \theta_2 \\
&\quad + v \cos \phi \left(-\frac{L_2}{L_1} * \sec \phi \tan \delta\right) \cos \theta_2 \\
\frac{\partial f_4}{\partial v_1} &= \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \cos \theta_2 \\
\frac{\partial f_5}{\partial \theta_2} &= v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \delta \tan \phi\right) \cos \theta_2 \\
\frac{\partial f_5}{\partial \phi} &= v(-\sin \phi) \left(1 - \frac{L_2}{L_1} \tan \delta \tan \phi\right) \sin \theta_2 \\
&\quad + v \cos \phi \left(-\frac{L_2}{L_1} * \sec \phi \tan \delta\right) \sin \theta_2 \\
\frac{\partial f_5}{\partial v_1} &= \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \sin \theta_2 \\
\frac{\partial f_6}{\partial \phi} &= -v_1 \left(\frac{\cos \phi}{L_3} - \frac{L_2}{L_1 L_3} \sin \phi \tan \delta\right) \\
\frac{\partial f_6}{\partial v_1} &= -\left(\frac{\sin \phi}{L_3} + \frac{L_2}{L_1 L_3} \cos \phi \tan \delta\right) \\
\frac{\partial f_7}{\partial \phi} &= -\frac{v}{L_3} \cos \phi - \frac{v}{L_1} \left(1 - \frac{L_2 \sin \phi}{L_3}\right) \tan \delta \\
\frac{\partial f_7}{\partial v_1} &= -\frac{1}{L_3} \sin \phi - \frac{1}{L_1} \left(1 + \frac{L_2 \cos \phi}{L_3}\right) \tan \delta
\end{aligned} \tag{4.25}$$

And similarly, the matrix B' can be written as:

$$B' = \begin{bmatrix} \frac{\partial f_1}{\partial \delta} & \frac{\partial f_1}{\partial a} \\ \frac{\partial f_2}{\partial \delta} & \frac{\partial f_2}{\partial a} \\ \frac{\partial f_3}{\partial \delta} & \frac{\partial f_3}{\partial a} \\ \frac{\partial f_4}{\partial \delta} & \frac{\partial f_4}{\partial a} \\ \frac{\partial f_5}{\partial \delta} & \frac{\partial f_5}{\partial a} \\ \frac{\partial f_6}{\partial \delta} & \frac{\partial f_6}{\partial a} \\ \frac{\partial f_7}{\partial \delta} & \frac{\partial f_7}{\partial a} \\ \frac{\partial f_8}{\partial \delta} & \frac{\partial f_8}{\partial a} \end{bmatrix} \quad (4.26)$$

where the partial derivatives of equations f_1 to f_8 with respect to input variables can be written as the following (similarly, the rest of the partial derivatives is 0):

$$\begin{aligned} \frac{\partial f_8}{\partial a} &= 1 \\ \frac{\partial f_3}{\partial \delta} &= \frac{v_1}{L_1} \sec \delta \\ \frac{\partial f_4}{\partial \delta} &= -v_1 \cos \phi \frac{L_2}{L_1} \tan \phi \sec \delta \cos \delta \\ \frac{\partial f_5}{\partial \delta} &= -v_1 \cos \phi \frac{L_2}{L_1} \tan \phi \sec \delta \sin \delta \\ \frac{\partial f_6}{\partial \delta} &= -v_1 \frac{L_2}{L_1 L_3} \cos \phi \sec \delta \\ \frac{\partial f_7}{\partial \delta} &= -\frac{v_1}{L_1} \left(1 + \frac{L_2}{L_3} \cos \phi\right) \sec \delta \end{aligned} \quad (4.27)$$

Then, using the methods of Forward Euler Discretization with sampling time dt , the state transfer equation can be written as:

$$X_{k+1} = X_k + f(X_k, U_k) dt \quad (4.28)$$

Using first degree Tayer expansion around $\bar{X}$ and $\bar{U}$

$$\begin{aligned} X_{k+1} &= X_k + (f(\bar{X}, \bar{U}) + A'z_k + B'U_k - A'\bar{X} - B'\bar{U}) dt \\ X_{k+1} &= (I + dtA') X_k + (dtB') U_k + (f(\bar{X}, \bar{U}) - A'\bar{X} - B'\bar{U}) dt \end{aligned} \quad (4.29)$$

The Equation 4.29 can be transferd to the form following state space control schema:

$$x_{k+1} = Ax_k + BU_k + C \quad (4.30)$$

where the A , B , and C can be replaced using the derived Jacobian matrix:

$$\begin{aligned} A &= (I + dtA') \\ B &= dtB' \\ C &= (f(\bar{z}, \bar{u}) - A'\bar{z} - B'\bar{u}) dt \end{aligned} \quad (4.31)$$

As shown in Figure 4.16 shows the reverse parking trajectory of MPC in Unity simulated environment. When compared to LQR simulation results in previos section, the MPC approach demonstrates significant improvements, and the trajectory is closer to the given Dubins path, and the final parking state is closer to desired goal state.

The performance of the MPC approach for the truck-trailer reverse parking problem is illustrated in Figure 4.17, Figure 4.18, and Figure 4.19. These results demonstrate the effectiveness of MPC in handling the complex dynamics of the truck-trailer system during reverse parking maneuvers.

Figure 4.17 shows the comparison between the trailer's actual trajectory and the desired parking path under MPC control. The MPC-controlled trajectory closely follows the desired path, exhibiting minimal deviations. This performance surpasses that of the LQR controller (shown in Figure 4.12), where larger deviations were observed, particularly in the middle section of the path.

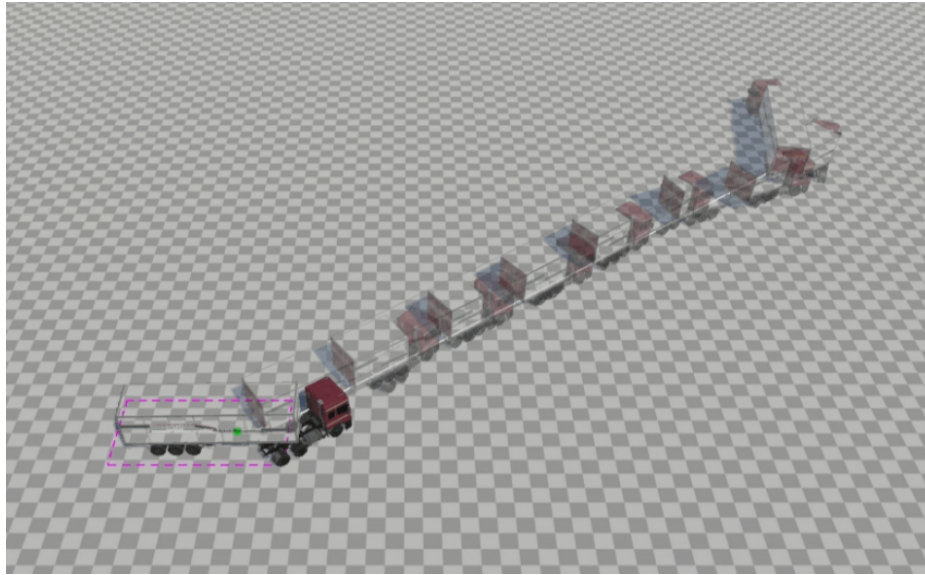


Figure 4.16: The stacked simulation frames of using Dubins and MPC controller for TTRV reverse driving control.

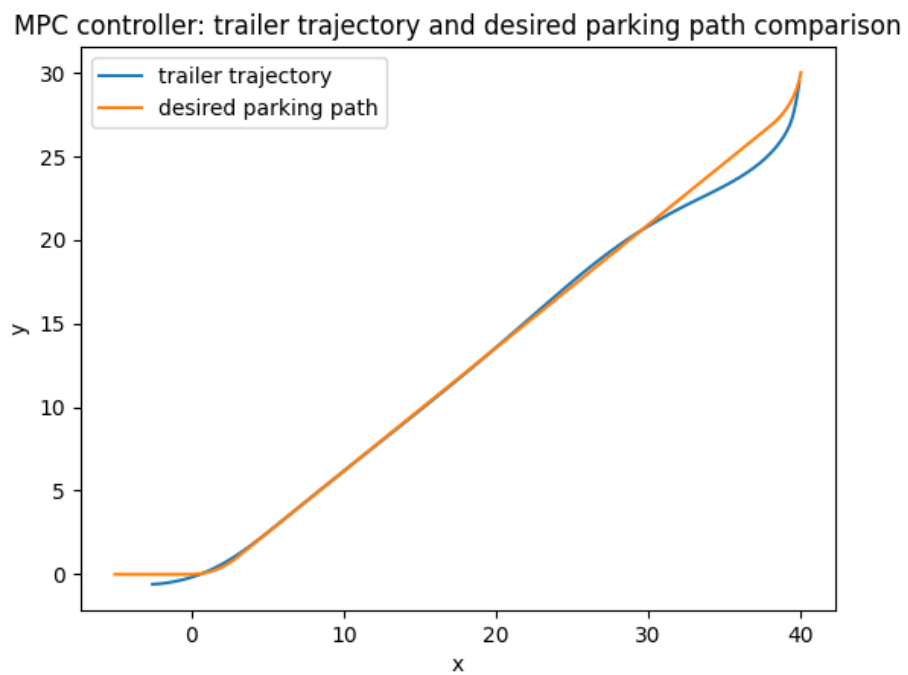


Figure 4.17: MPC controller: trailer trajectory and desired parking path comparison.

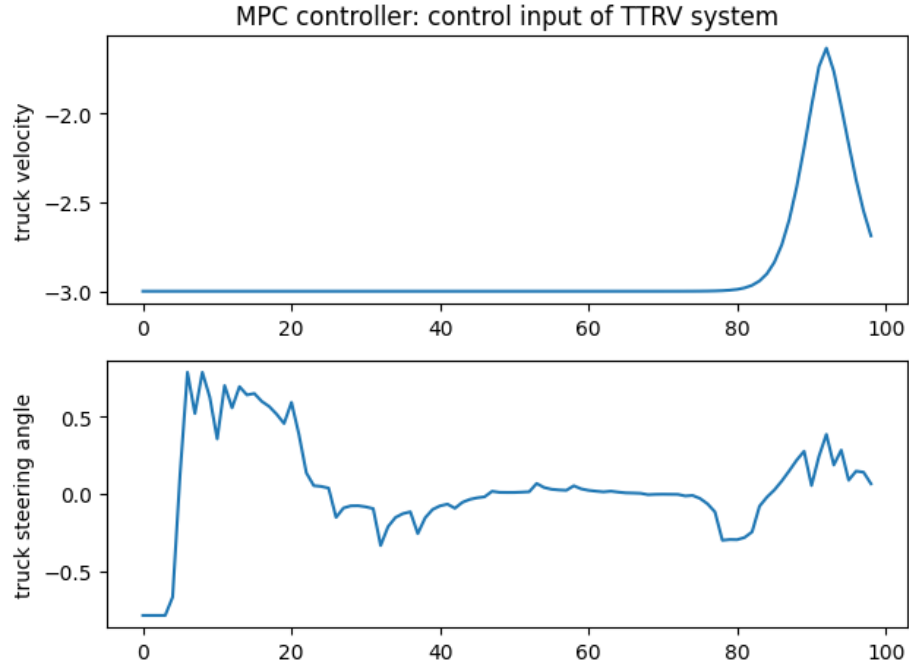


Figure 4.18: MPC controller: control input of TTRV system.

The control inputs for TTRV system under MPC are presented in Figure 4.18. The truck velocity (top graph) shows a smoother profile compared to the LQR results (Figure 4.13), with gradual changes rather than abrupt shifts. The steering angle (bottom graph) also demonstrates more controlled and less oscillatory behavior than its LQR counterpart. These smoother control inputs contribute to more stable and efficient maneuvering of the truck-trailer system.

The trailer's lateral offset and steering angle under MPC control are shown in Figure 4.19. The lateral offset (top graph) exhibits smaller peak values and less fluctuation compared to the LQR results (Figure 4.14), indicating better path-following performance. The steering angle (bottom graph) shows a smoother profile with fewer sharp changes, contributing to more stable control of the truck-trailer system.

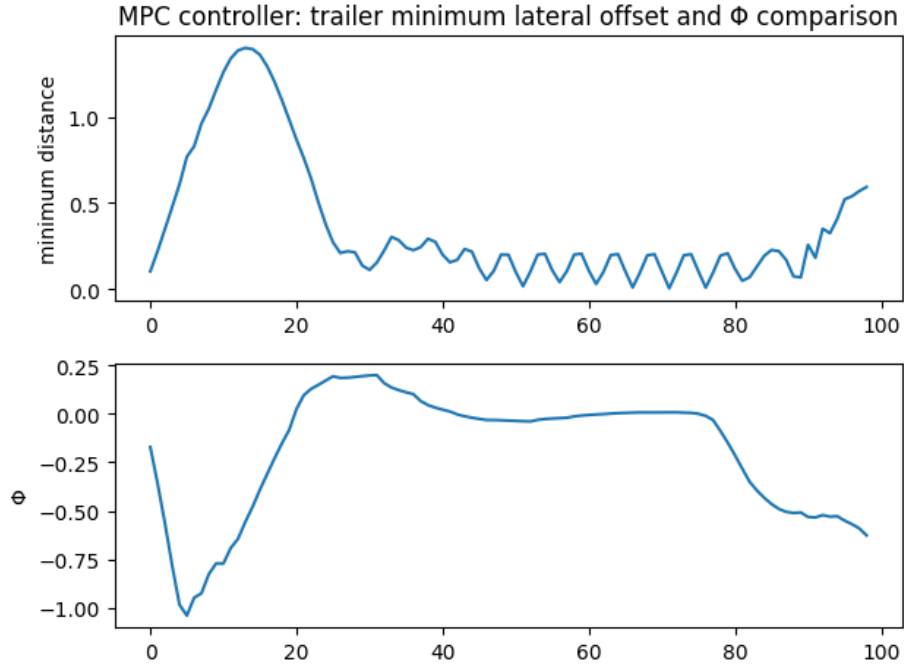


Figure 4.19: MPC controller: trailer minimum lateral offset and steering angle comparison.

Future more, the MPC controller has also been tested in obstacle avoidance scenarios, where the truck-trailer system needs to navigate around several stationary obstacles while reverse parking. The simulation results are shown in Figure 4.20, which demonstrates the MPC controller's ability to handle complex environments and dynamic obstacles. The trailer successfully avoids the obstacle while following the desired parking path, showcasing the MPC controller's robustness and adaptability in challenging scenarios.

In conclusion, this section explores the application of MPC to the TTRV system. Compared with LQR, MPC can incorporate non-linear dynamics and balance multiple objectives effectively. By solving an optimization problem at each time step, MPC adapts to changing conditions, resulting in smoother control inputs and more stable system

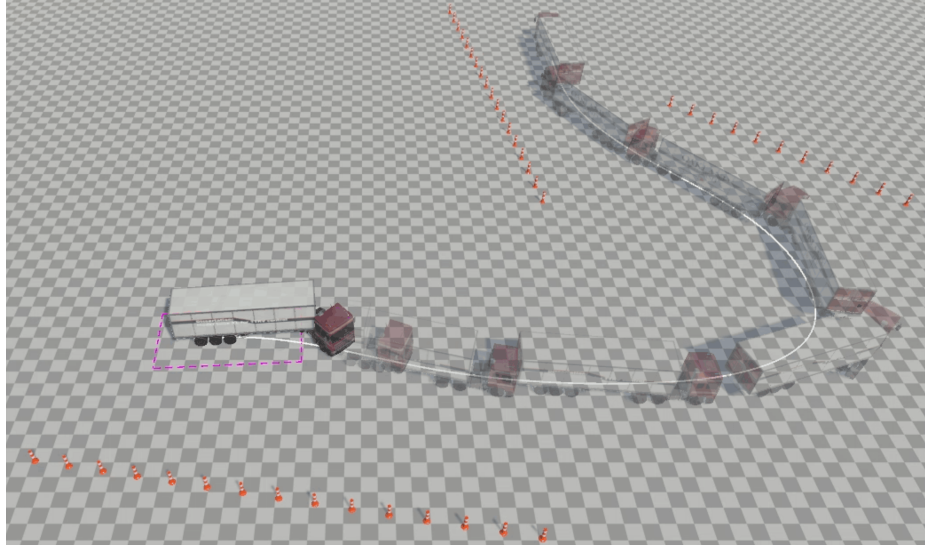


Figure 4.20: The stacked simulation frames of using RRT* and MPC controller for TTRV reverse driving control with obstacle avoidance.

behavior. These improvements help MPC to be a very popular choice for controlling the non-linear and constrained vehicle system including TTRV.

4.5 Chapter Conclusion

This chapter explored several traditional control methods for the TTRV system, focusing on vehicle state stabilization and path following. The pole placement and gain scheduling controllers, Lyapunov-based control methods, LQR, and MPC are constructed, simulated. Each method was adapted to the TTRV state space, with simulation results demonstrating their effectiveness.

Pole placement and gain scheduling provided a foundation for linear control and extension to nonlinear dynamics. Lyapunov-based control offered a systematic approach to stability. LQR demonstrated optimal control for linear systems with quadratic cost functions. MPC emerged as a powerful technique, incorporating predictive capabilities

and constraint handling. The simulation comparison shows that MPC's superior performance in complex scenarios like reverse parking and obstacle avoidance.

These traditional control methods serve as important benchmarks for the next chapter, where this research will explore advanced reinforcement learning-based approaches for TTRV control, aiming to overcome the limitations of traditional methods and achieve even better performance in complex, dynamic environments.

CHAPTER FIVE

MODEL FREE DEPP REINFORCEMENT LEARNING FOR TTRV REVERSE PARKING CONTROL

Deep Reinforcement Learning has been shown to be a powerful tool for solving complex control problems, particularly in application scenario where traditional control methods has degradation performance due to high-dimensional state spaces, nonlinear dynamics, and uncertain environments. DRL combines the strengths of Deep Learning (DL) and RL, which allows agents to learn control policies directly from raw sensory inputs.

This chapter focus on the application of Model-free Reinforcement Learning (MFRL) to the problem of TTRV reverse parking control. As analyzed in section 2.4, TTRV faces challenges during reverse maneuvering because of complex nonlinear dynamics, limited maneuverability, and physical constraints. This chapter details how DRL can address these challenges by learning robust and efficient control policies through interaction with the environment, or learn from past experience trajectory. The chapter first provides the history of Machine Learning (ML) and RL in section 5.1. In section 5.2, the design of the reward function and the state representation, which are critical components for a success learning, are discussed. Then, the PPO and TD3 algorithms are introduced in section 5.3 and section 5.4 respectively, which are the chosen DRL methods for the reverse parking control problem. The implementation details and simulation results are presented at the end of each section. Finally, the chapter concludes in section 5.5 with a summary of the findings and potential future directions.

5.1 History of Reinforcement Learning

Before diving into the details of DRL for TTRV reverse parking control, it is essential to understand the fundamental concepts of ML, DL, and Neural Network (NN). ML, a subfield of Artificial Intelligence (AI), focuses on the development of algorithms and models that enable computers to learn and improve their performance on a specific task without being explicitly programmed[52]. The early works in ML can be traced back to the 1950s, with the introduction of the perceptron algorithm by Frank Rosenblatt[53]. The perceptron is a simplified model of a biological neuron and this work build the foundation for future artificial neural network. This algorithm is designed to perform binary classification tasks and is capable of learning from examples. However, the perceptron has its limitations, as it only solve linearly separable problems. Another significant development during this period is the concept of pattern recognition, introduced by George Sebestyen[54], which aimed to develop methods for identifying patterns and regularities in the given data, which is one of fundamental tasks in ML.

In the 1980s and 1990s, ML has gained many progress, including the development of decision trees[55], support vector machines[56], and ensemble methods such as boosting[57] and bagging[58], which have become classical algorithms known for their straightforward interpretation and effectiveness, and these algorithm are still being used nowadays. These algorithms built the foundation for solving classification and regression problems and paved the way for the ML algorithms development.

Followed by the advancements in ML research, DL applicattion became popular in both research and industry field, and it focuses on learning hierarchy representation from raw data using deep neural networks[59, 60]. These networks consists multiple layers of interconnected nodes that can learn complex features from the input data by themselves. Deep learning has achieved success in many applications, including computer vision[61],

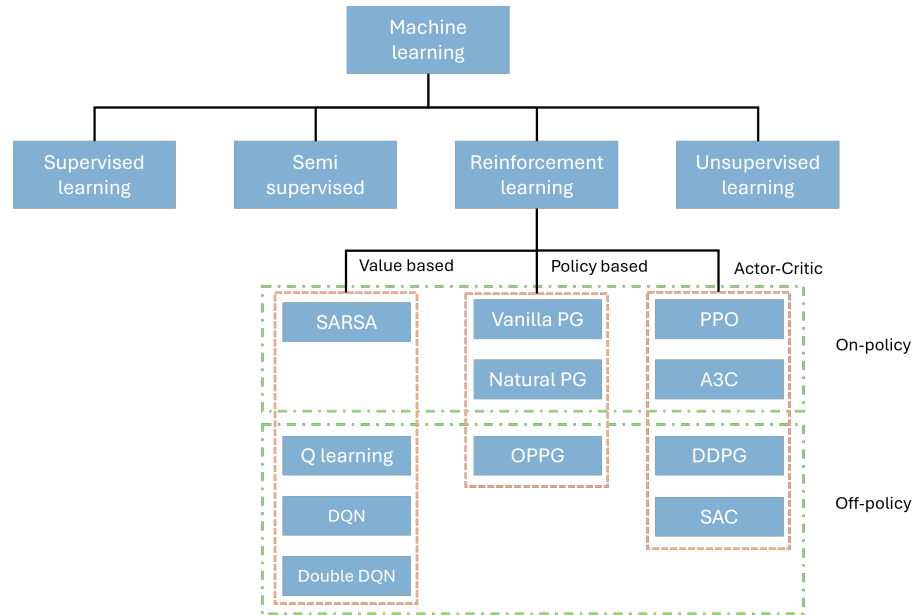


Figure 5.1: The category of machine learning algorithms and reinforcement learning algorithms.

where Convolutional Neural Network (CNN) have revolutionized image classification and object detection tasks, and natural language processing[62], where Recurrent Neural Network (RNN) and transformers have made significant progress in language modeling and machine translation. The success of DL has inspired researchers to integrate it with reinforcement learning, which further enables the algorithms to interact with raw environment states, build end-to-end control strategy and solve complex decision-making problems in high-dimensional spaces. In this structure, deep neural networks serve as the function approximators for learning value functions or policies, making it possible to tackle challenging control tasks efficiently and effectively.

Reinforcement learning is built on Markov Decision Process (MDP), as shown in Figure 5.2, MDP is a mathematical framework to model the decision-making problem where outcomes of each states are partially random[63], and it provides a mathematical

method to describe the environment where an agent interacts and learns from interacting with it. The MDP consist a set of states that represent the possible configurations of the environment, a set of actions that the agent can take in each state, a transition probability function that defines the likelihood of moving from one state to another by taking an action, and reward function that returns a value from each state-action pair. The goal of the agent is to find a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time. By interacting with the environment and observing the rewards and state transitions, the agent can learn an optimal policy which maximize the agent's long-term performance. MDP provide the theoretical foundation for RL and the framework for modeling and solving sequential decision-making problems, which also enables the development of algorithms which learn from experience and adapt to changing environments.

Early works in RL focused on tabular methods, where the value functions and policies are represented using tables or arrays such as in Figure 5.3. These methods, such as Q-learning[15] and State-Action-Reward-State-Action (SARSA)[64], is operated by storing and updating the value table for state-action pairs. The agent then use the table to choose the best action in each given state, based on the expected future rewards. This approach is highly effective in environments with a small number of discrete states and actions; however, as the state and action spaces grow large or being continuous in robotics problems, the size of the table will become pretty large, and it causes performance degradation from memory usage and computational efficiency. Moreover, the agent may need to visit each state-action pair multiple times to accurately estimate the value function, which is low efficiency or even impossible for problems with large state space.

Followed by tabular methods, the integration of deep learning with RL is considered to be the milestone for the development of RL. The Deep Q-Network (DQN)[16] proposed in 2013, demonstrated the ability of deep neural networks to learn

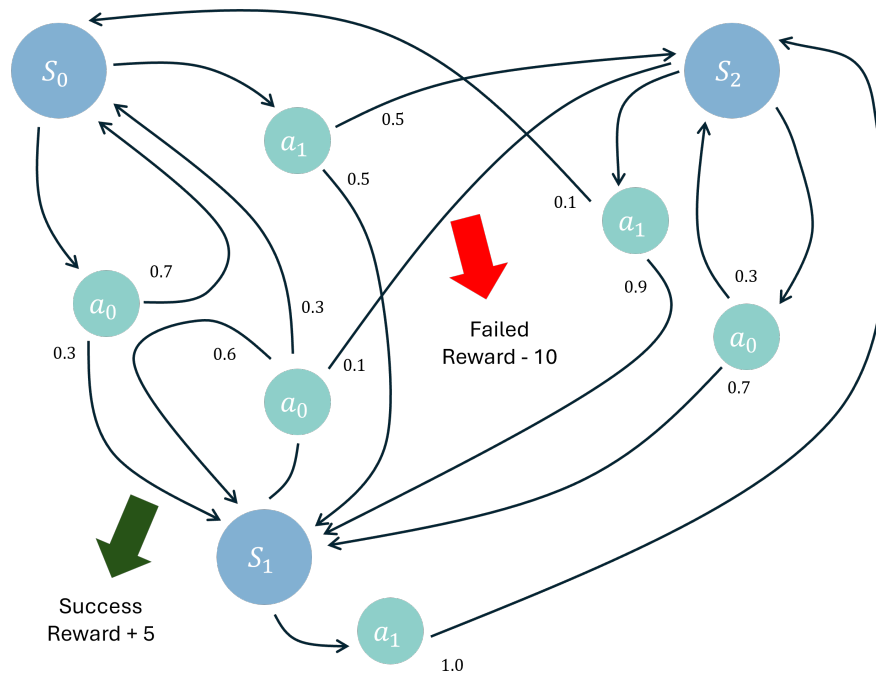


Figure 5.2: Markov Decision Process modeling decision-making in partly random situations.

Columns: attributes of observations

Player	Sepal length	Sepal width	Petal length	Petal width
Flower A	30.4	50.8	6.3	5.9
Flower B	26.5	2.6	3.6	3.5
Flower C	54.8	1.5	4.2	4.4
Flower D	19.9	6.4	13.2	1.6
Flower E	17.2	8.3	9.1	2.7
Flower F	31.4	6.5	5.4	6.6
Flower G	20.0	3.1	6.6	2.2

Rows: Observations

Figure 5.3: Tabular learning methods store and update a table of values for each state-action pair.

effective control policies directly from raw pixel inputs in Atari games, and it is a groundbreaking achievement because it showed that an AI could learn a complex task without prior knowledge of the rules but simply from screen outputs. The success of the DQN in mastering Atari games shows the potential of DRL in handling real-world tasks with high-dimensional inputs and complex dynamics. Another breakthrough of DRL was AlphaGo[65], which is proposed by DeepMind in 2016. AlphaGo used a combination of deep neural networks, RL, and Monte Carlo tree search to master the game of Go. It won four out of five games against world champion Lee Sedol, showed the ability of DRL to handle complex problems that were previously thought to be the impossible for computer to solve.

Recent advancements in MFRL have further pushed the boundaries of possible applications in robotics. Algorithms such as Deep Deterministic Policy Gradient (DDPG)[66], PPO[67], and Soft Actor-Critic (SAC)[68] have demonstrated impressive performance in continuous control tasks, enabling the learning of robust and efficient policies in high-dimensional action spaces.

In nowadays DRL research, algorithms can be categorized into model-based and model-free methods. Model-based Reinforcement Learning (MBRL) algorithms attempt to learn a model of the environment's dynamics, which is used for planning and decision-making. MBRL is usually considered to be sample-efficient, however, it may converge slower in complex, high-dimensional systems where an accurate model is normally difficult to learn. In comparison, MFRL algorithms learn control policy directly from environment interactions without explicitly modeling dynamics, and this approach can be particularly helpful in scenarios with complex, nonlinear dynamics or where the environment is subject to frequent changes, which might invalidate the learnt model. The nature of MFRL to learn effective policies without relying on an explicit model shows that

it's potential to be applied for challenging control problems in robotics and autonomous systems.

5.2 Gravitation Potential Based Reward Function

To design DRL algorithm, two critical component have significant impact on the final learning ability, which are state representation and reward function design. State representation defines how the environment and vehicle configuration are encoded for the algorithm, which affects agent's perception and decision-making abilities. The reward function provides guidance for the agent to explore the environment during learning process, providing feedback on action such as input steering and velocity. For tasks with non-linear dynamics and multiple control objectives like trailer parking, the definition of effective state representations and reward functions is even more challenging and crucial. In this section, a potential field based reward function is introduced for TTRV parking, which is inspired from robotics and designed to efficiently help the agent navigate in the environment.

The concept of potential fields has been widely used in path planning for robotic navigation since 1980s. The research done by Khatib[69] proposed the artificial potential fields for obstacle avoidance, in which the robot is considered as a particle moving under the influence of attractive and repulsive forces. The goal position generates an attractive potential, and obstacles generate repulsive potentials, which all together creates a gradient that navigate the robot towards the goal and avoid collisions. Since then, numerous variations and improvements have been proposed to address the limitations of the original potential field method, such as local minima and oscillations in narrow passages[70, 71]. Connolly et al.[72] introduced the use of harmonic potential fields to eliminate local

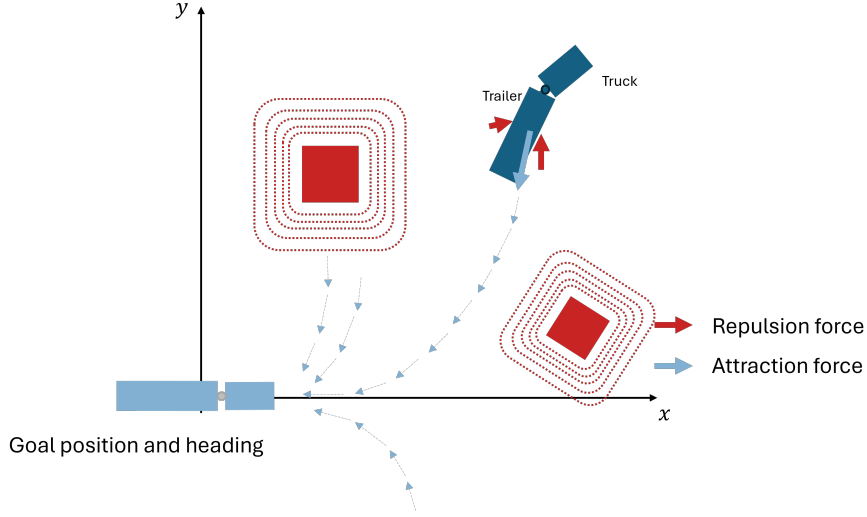


Figure 5.4: Illustration of the gravity potential reward function, which guides the trailer towards the goal while avoiding obstacles.

minima, and Rimon and Koditschek[73] proposed navigation functions which can guarantee convergence to the goal in the presence of obstacles.

In recent years, potential field methods have been combined with other path planning techniques, such as sampling-based algorithms and optimization-based approaches, to improve their performance in complex environments[74]. The integration of potential fields with RL has also gained attention, as it provides a way to incorporate obstacle avoidance and goal-reaching behaviors into the reward function, guiding the learning process towards desired navigation strategies. Building upon these research progress, this section introduces a novel reward function for TTRV parking. The proposed method is inspired by the concept of gravity potential, generates a gradient that encourages the trailer to move towards the goal while avoiding obstacles in its path as shown in Figure 5.4.

The gravity potential based reward R_g is calculated at each step based on the trailer's states $[x_2, y_2, \theta_2, \Phi]$ relative to the goal and obstacles in the environment. The

reward function consists of three main components: the attraction force, the repulsion force, and the alignment reward for truck trailer angle Φ . The attraction force is modeled using the distance to the goal d_g and the alignment between the trailer's orientation θ_2 and the tangent angle of the gravity potential curve connecting the current position to the goal. The distance reward component is calculated as:

$$R_d = \frac{1}{(1 + d_g)^4} \quad (5.1)$$

The alignment reward component is based on the cosine similarity between the trailer's orientation and the tangent angle:

$$R_\theta = \exp \left(-\frac{(\cos(\theta_2 - \theta_t) - 1)^2}{2\sigma^2} \right) \quad (5.2)$$

where θ_t is the tangent angle of the gravity potential curve at the current position, and σ is a scaling factor. To incorporate the repulsion force, a repulsion reward component R_r is defined based on the distance to the nearest obstacle d_o :

$$R_r = \begin{cases} \exp \left(-\frac{(d_o - d_{min})^2}{2\sigma_r^2} \right) & \text{if } d_o \leq d_{min} \\ 0 & \text{if } d_o > d_{min} \end{cases} \quad (5.3)$$

where d_{min} is the minimum desired distance from obstacles, and σ_r is a scaling factor.

This repulsion reward component discourages the trailer from getting too close to obstacles by assigning a smaller reward when the distance to the nearest obstacle falls below the threshold d_{min} , and the distance to obstacle is given by lidar rays from the trailer center, and each ray returns the distance to the nearest obstacle. The obstacle avoidance reward R_o is calculated based on the lidar measurements:

$$R_o = -\sum_{i=1}^N \max \left(0, \frac{d_{safe} - d_i}{d_{safe}} \right)^2 \quad (5.4)$$

where N is the number of rays, d_i is the distance to the nearest obstacle for ray i , and d_{safe} is the safe distance threshold. For rays that don't detect obstacles within the safe distance, this ray will contribute no penalty, encouraging the agent to maintain a safe distance from obstacles during navigation.

The total gravity potential reward R_g is then calculated as a weighted sum of the attraction, repulsion, and alignment components:

$$R_g = w_d R_d + w_\theta R_\theta + w_r R_r + w_o R_o \quad (5.5)$$

where w_d , w_θ , w_r , and w_o are weight factors that determine the relative importance of each component.

In summary, the proposed gravity potential reward function offers a novel effective reward design to apply DRL algorithms in trailer reverse parking and control. By combining the attraction force, repulsion force, and alignment components, the reward function creates a gradient that effectively guides the trailer towards the desired states while avoiding obstacles, which eventually shows improved navigation performance.

5.3 On-Policy Approach: Proximal Policy Optimization Based TTRV Reverse Parking Control

This section introduces an on-policy methodology for reverse parking control of TTRV systems, which combines the PPO with a the potential field based reward structure. The PPO controller is designed to solve the intricate dynamics in TTRV reverse parking scenarios. This section also provides a comprehensive overview of the implementation, including both the deep reinforcement learning architecture and the simulation environment. This proposed PPO approach is designed to navigate the complexities of high-dimensional state spaces and address the temporal challenges of TTRV systems. At

the end of this chapter, the simulation results from PPO and classical control methods are compared, showing the PPO controller's effectiveness in maneuvering safe and efficient reverse parking maneuvers for TTRV. Through this exploration, this section aims to demonstrate the potential of the hybrid approach in advancing autonomous control strategies for complex vehicular systems.

5.3.1 Introduction of Proximal Policy Optimization

Proximal Policy Optimization, introduced by OpenAI in 2017[67], is a on-policy DRL approach, based on Policy Gradient (PG) methods. PPO offers several advantages over other deep DRL techniques, including enhanced data efficiency, reduced computational demands, and a lower risk of policy divergence or catastrophic forgetting[75].

The fundamental concept of the PG method is to compute a policy gradient estimator and utilize it in a stochastic gradient ascent algorithm. The gradient estimator can be written as:

$$\hat{g} = \hat{\mathbb{E}}_t [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t] \quad (5.6)$$

where π_{θ} is a stochastic policy, $\hat{A}_t$ is the advantage estimator at timestep t , and $\hat{\mathbb{E}}_t[\dots]$ is the empirical average over a finite batch of samples. This estimator is derived by differentiating the objective function:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t [\log \pi_{\theta}(a_t | s_t) \hat{A}_t] \quad (5.7)$$

One drawback of PG methods is caused by its sensitivity to step size, which can complicate the optimization process. To solve this issue, PPO used similar approach of the Trust Region Policy Optimization (TRPO) algorithm[76], by maximize an objective

function that incorporates the ratio from the old policies $\pi_{\theta_{\text{old}}}$ to new policies π_{θ} , while penalizing large policy updates with a coefficient β :

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t - \beta \text{KL} [\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)] \right] \quad (5.8)$$

The probability ratio of old and new policies is denoted as $r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$ in following equations.

To mitigate large policy updates when maximizing the objective L^{CPI} , the PPO introduces a clipping logic which penalizes changes to the policy that move $r_t(\theta)$ away from 1:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t [\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t)] \quad (5.9)$$

The first term $r_t(\theta) \hat{A}_t$ is derived from the TRPO method, while the second term, $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t$, is the clip function that constrains the values of $r_t(\theta)$ within the range $[1 - \epsilon, 1 + \epsilon]$. The min function selects the smaller value between the clipped and unclipped objectives, which effectively create a lower bound for the objective.

In practice, glsppo often utilizes a surrogate objective that combines the clipped objective with an entropy bonus to encourage exploration:

$$L^{TOTAL}(\theta) = \hat{\mathbb{E}}_t [L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 S[\pi_{\theta}](s_t)] \quad (5.10)$$

In Equation 5.10, $L^{CLIP}(\theta)$ is the clipped objective function as defined in Equation 5.9, $L^{VF}(\theta)$ is a squared-error loss $(V_{\theta}(s_t) - V_t^{targ})^2$ for the value function, $S\pi_{\theta}$ is the entropy of the policy, and c_1, c_2 are coefficients. This formulation encourages the algorithm to balance between maximizing rewards, accurately estimating value functions,

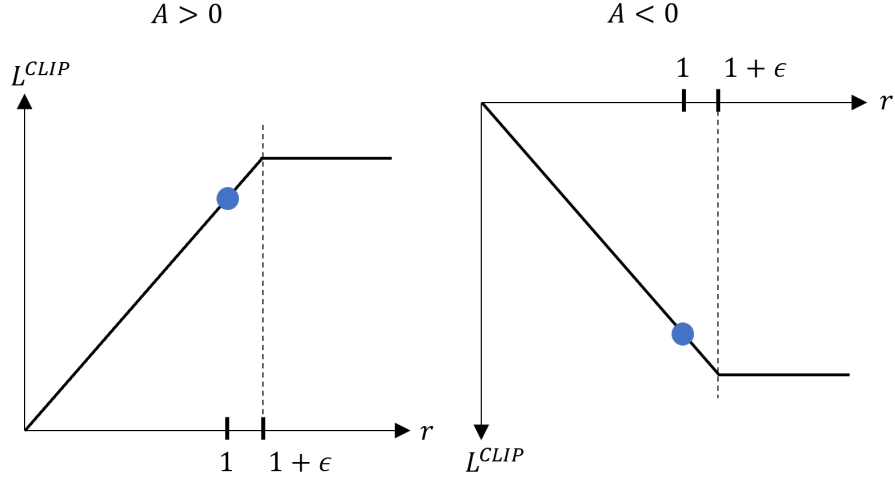


Figure 5.5: Visualization of the clip function, illustrating how the probability ratio r is clipped at $1 - \epsilon$ or $1 + \epsilon$ depending on the sign of the advantage.

and maintaining sufficient exploration through policy entropy, making it robust to hyperparameter and stable during training.

5.3.2 Implementation and Simulation

The architecture for the TTRV autonomous parking system is shown in Figure 5.6. This system has two main components: a DRL module and a simulation framework. The DRL module uses an PPO method to train the agent to park by itself and avoid obstacles while interacting with the things in the parking environment. The simulation framework includes the TTRV kinematic model, static vehicle obstacles, parking spaces, and sonar-based distance sensing, all implemented in MATLAB/Simulink and Python to interact with the training agent, and then the TTRV states can be visualized in Unity for better visual understanding and motion behavior analysis.

To smooth the vehicle control, the simulation introduces an action module that represents the truck's steering input for lateral control. The steering controller select best

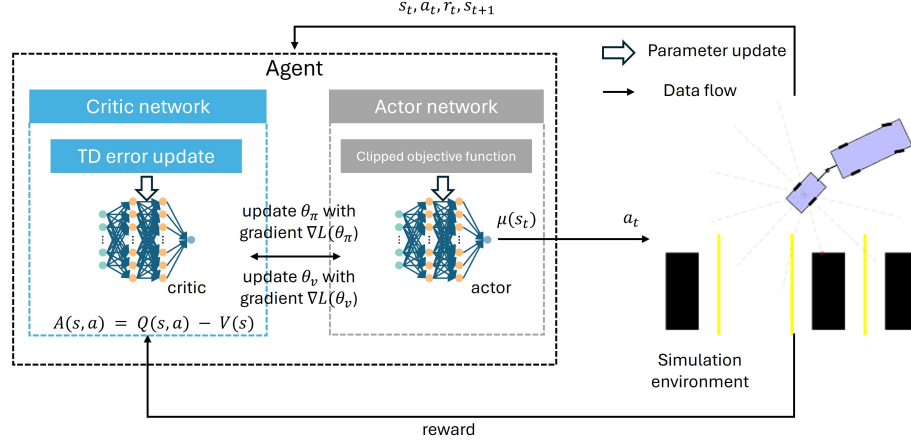


Figure 5.6: Structure of the PPO algorithm, including the DRL module and simulation framework.

action value from a continuous action space, generated by the PPO algorithm, and is defined within the range of $[-\delta_{\max}, \delta_{\max}]$, where $\delta_{\max}$ is the maximum steering angle based on the main vehicle's specifications. Also, as described in section 5.2, the reward function acts as the feedback channel for the controller, projecting the current state and action pair.

Dimension	Measurement
Wheelbase (L_1)	311.9 cm (122.8 inches)
Front Overhang	95.5 cm (37.6 inches)
Rear Overhang (L_2)	123.4 cm (48.6 inches)
Trailer Length	300 cm (118.1 inches)
Trailer's Axis to Hitch (L_2)	400 cm (157.5 inches)

Table 5.1: Ford F-150 (Regular Cab, 6.5-foot box) and trailer Dimensions.

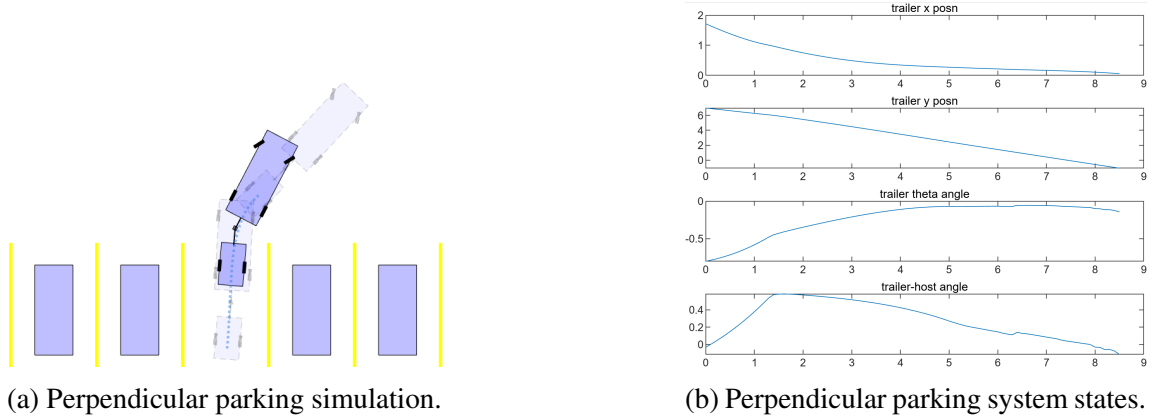


Figure 5.7: TTRV perpendicular parking.

The PPO algorithm with obstacle avoidance is first implemented in MATLAB/Simulink, and the vehicle specification is from a Ford F-150, where the vehicle specification including a box trailer is listed in Table 5.1, where the agent learns to park the trailer system autonomously. The training data is collected from the simulation interaction, and the agent's policy is updated iteratively to improve its performance.

Figure 5.7 shows the trajectory of perpendicular parking simulation results, including host velocity and steering angle, to approach the chosen parking spot during the perpendicular parking task. The starting positions are initialized with noise added, and the goal position is determined by user input. The dotted trajectory represents the trailer's path, with two faded TTRVs indicating its initial and concluding positions. In Figure 5.7b, the trailer's states during the parking task are displayed in the global coordinate system, aligning with the parking goal states including position and orientation.

In Figure 5.8, the TTRV trajectory during a 45° oblique parking simulation is depicted. Similarly, the starting positions are initialized with a certain degree of noise, and the proposed methods successfully parked the TTRV system, ensuring jackknife prevention and obstacle evasion.

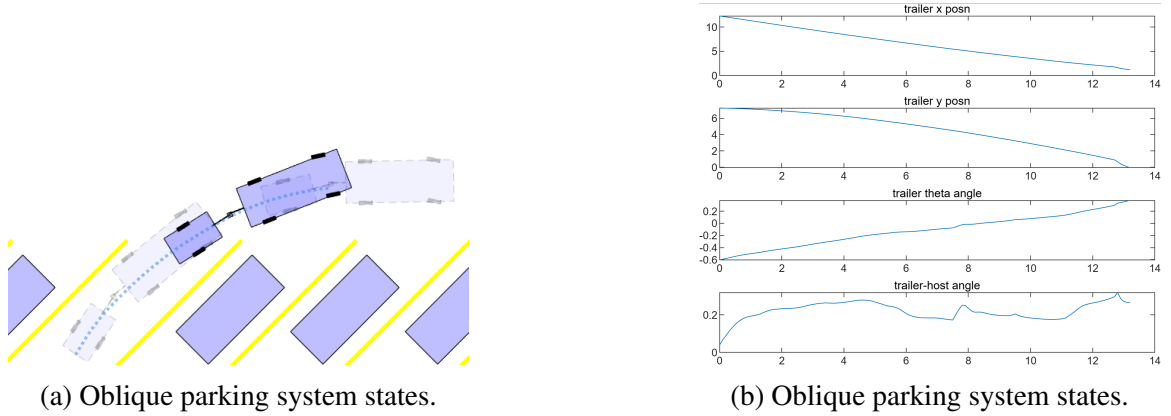


Figure 5.8: TTRV oblique parking demonstration.

Following initial prototyping, this research is migrated from MATLAB/Simulink implementation to Python environment, and established a benchmark performance for comparing various RL algorithms comparison in the rest of the research. This shift was motivated by several key factors: Python’s massive deep learning libraries and frameworks, such as TensorFlow and PyTorch, offer opensource and flexible implementations of reinforcement learning algorithms, including PPO. Furthermore, Python’s seamless integration with Unity’s simulation capabilities allows realistic and visually rich environments, which is important for analyzing complex motion behaviors and obstacle interactions.

A comprehensive test bench was developed using the OpenAI Gym framework, beginning with PPO controller without obstacle avoidance consideration. This streamlined approach enabled the isolation of main challenges in TTRV control and facilitated faster learning and convergence. By establishing this foundational baseline, the PPO controller can be set for baseline evaluation for future improvements. In following chapters, the agent will start to learn the parking strategy in the environment with obstacles, using similar sonar detection logic from the MATLAB environment. This progression allowed

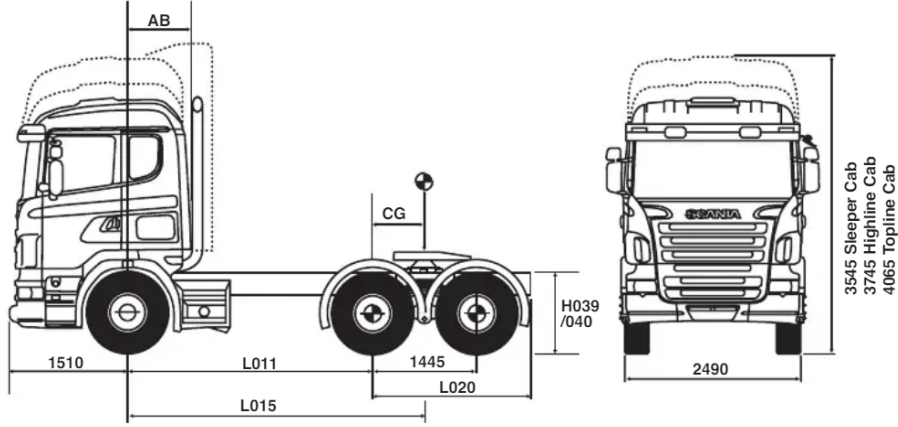


Figure 5.9: Illustration of Scania R 520 LA6x2/2MNA truck and trailer in Unity simulation environment.

Dimension	Measurement
L011 (Wheelbase)	4000 mm
L015 (Theoretical wheelbase)	3532 mm
L020 (Rear overhang)	775 mm
Front overhang	1460 mm
Width	2490 mm
Sleeper Cab CG	878 mm
Fifth Wheel position	960 mm forward of drive axle centre line
Trailer's Axis to Hitch	8000 mm

Table 5.2: Scania R 520 LA6x2/2MNA truck and trailer Dimensions.

for training the agent to navigate obstacles while parking the trailer, building upon the knowledge gained from the obstacle-free parking scenario. Also, the truck vehicle used in Python environment is changed to a cab over commercial truck, Scania R 520 LA6x2/2MNA truck, and the truck and trailer dimensions are listed in Table 5.2.

Figure 5.10 illustrates the Python-based simulation in PyGame and Unity, showing the TTRV system successfully navigating to the desired parking spot in a obstacle free

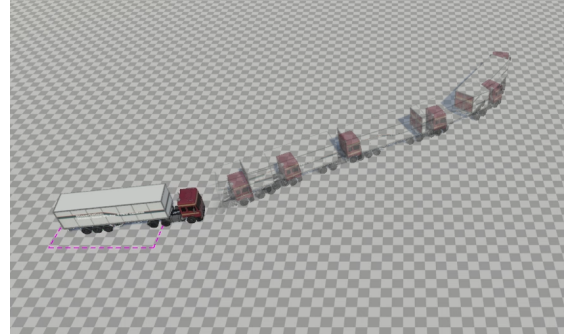
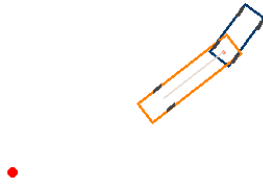


Figure 5.10: PyGame and Unity visualization environment showing trailer successfully parking.

environment. To evaluate the learning ability of the PPO algorithm, the experimental setup involves randomly initializing the truck-trailer system at distances ranging from 40 to 60 meters from the designated parking area, with randomly selected initial orientations to train the agent steer efficiently during parking, with minimal steps required and steering inputs variation. This simulation is analyzed with several evaluation criterias: success rate, mean episode length, and mean reward. These metrics provide quantitative assessments into the algorithm's learning efficiency and overall performance in the trailer parking task.

Figure 5.11 shows the agent's success rate over time. The graph illustrates both raw and smoothed data, providing insight into the learning progression. Initially, the success rate is low since the agent is still exploring and learning the proper parking maneuver, but it rapidly improves around the 1.5 million step mark, reaching and maintaining a high success rate of approximately 90 – 100% for the remainder of the training process.

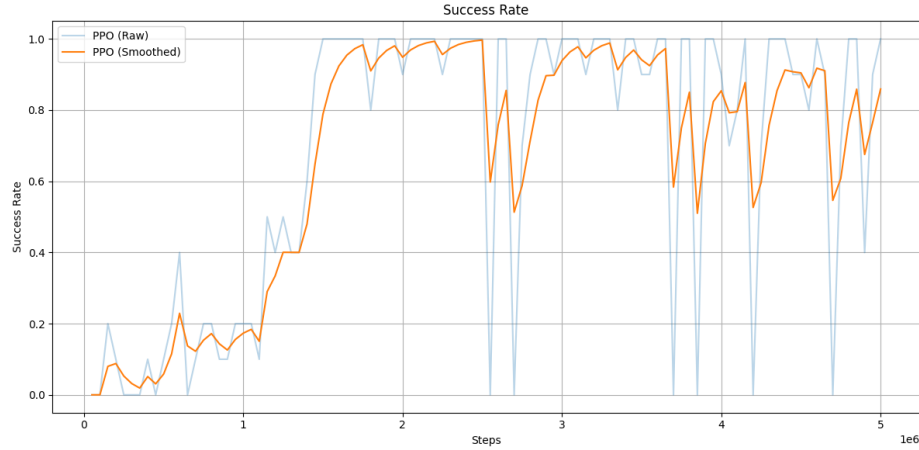


Figure 5.11: Success rate of PPO algorithm.

The mean episode length, shown in Figure 5.12, provides statistics related to the efficiency of the learned policy. The algorithm successfully reduced the episode length and eventually stabilized around 60-80 steps per episode, indicating effective learning of efficient parking strategies. The initial exploration phase, while taking relatively longer steps, allowed the algorithm to thoroughly investigate the state space, contributing to the robustness of the final policy. The observed fluctuations during the transition to efficient policies highlight the algorithm's ongoing refinement process in this complex control task.

Figure 5.13 shows the mean reward obtained over time. PPO showed an increase in rewards during the training process, demonstrating its ability to learn effective parking strategies. The fluctuations observed after 2.5 million steps could be related to the phenomenon of catastrophic forgetting, where the algorithm continues to adapt to recent experiences while potentially losing some previously learned information. This behavior underscores the complexity of the TTRV parking task and the ongoing balance between exploration and exploitation in the learning process.

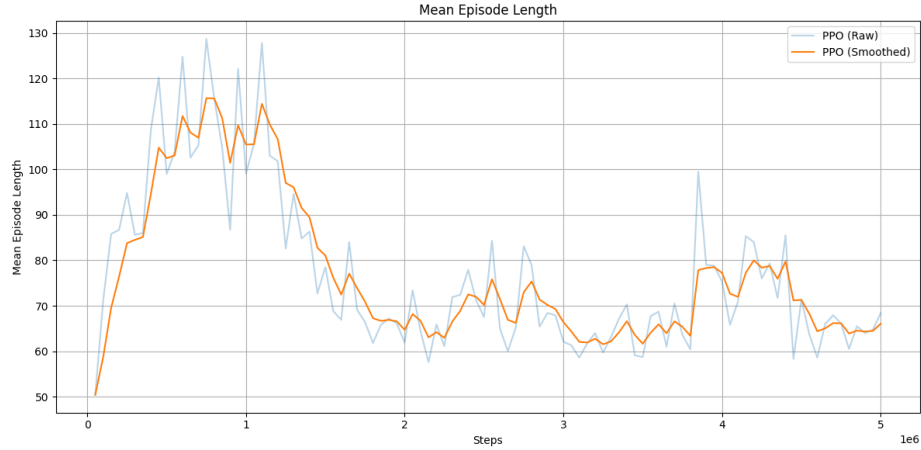


Figure 5.12: Mean episode length of PPO algorithm.

Overall, these results prove PPO’s capability in learning the challenging task of TTRV reverse parking control. While the learning process exhibits some fluctuations, the algorithm’s ability to significantly increase success rates, reduce episode lengths, and improve mean rewards over time highlights its potential for autonomous TTRV control applications. Future work would focus on further optimizing the PPO algorithm for TTRV applications, potentially incorporating transfer learning techniques to improve generalization across different trailer configurations and environmental conditions.

5.4 Off-Policy Approach: TD3 with Weight and Learning Rate Decay

This section introduces an off-policy DRL approach for TTRV autonomous reverse parking control, beginning with the original TD3 algorithm. Compared with PPO, TD3 has improved stability in continuous action spaces, and it serves as the foundation for the proposed enhancements module. The study later introduces a modification to the original TD3, which incorporates Weight Decay and Learning Rate Decay mechanisms. These additions aim to improve generalization, reduce overfitting, and optimize the

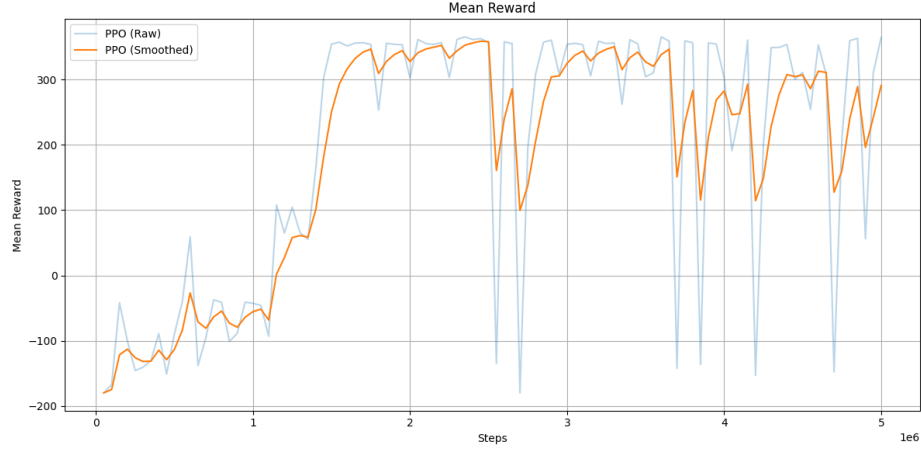


Figure 5.13: Mean reward of PPO algorithm.

learning process over time. To evaluate the modified algorithm TD3WD, the same parking simulation environment used for PPO experiments is also used to test proposed method. Additionally, a hybrid approach combining reinforcement learning with imitation learning is employed, where trajectories collected from MPC are used as expert demonstrations to guide the TD3WD algorithm's learning process. This demonstration-augmented approach allows for a comprehensive assessment of TD3WD's performance in handling the complex dynamics of TTRV reverse parking, while also exploring its potential for leveraging expert knowledge to enhance learning efficiency and performance.

5.4.1 Introduction to Twin Delayed Deep Deterministic Policy Gradient

The TD3 algorithm, introduced by Fujimoto et al. in 2018[77], shows a remarkable improvement in off-policy reinforcement learning for continuous action spaces. TD3 is built upon the DDPG algorithm[66], solving previous algorithm's limitations and enhancing overall performance and stability.

TD3 introduces three critical modifications to the DDPG framework. First, it uses a pair of critic networks to reduce overestimation bias in Q-value estimates, a common issue in actor-critic methods. By taking the minimum Q-value between the two critics as shown in Figure 5.14, TD3 provides a more conservative and robust estimate, mitigating the propagation of estimation errors through the value function. This can be written as:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi\phi'(s')) + \epsilon \quad (5.11)$$

where y is the target Q-value, r is the reward, γ is the discount factor, $Q_{\theta'_i}$ are the target critic networks, $\pi\phi'$ is the target policy network, and ϵ is the noise added for target policy smoothing.

Second, TD3 implements a delayed policy update mechanism. While the critic networks are updated at every timestep, the actor network and target networks are updated less frequently. This delay allows the critic to converge to more accurate Q-value estimates before they are used to update the policy, promoting stability and reducing the likelihood of premature convergence to suboptimal policies.

Third, TD3 introduces target policy smoothing, which is a regularization technique that adds noise to the target action. This modification helps to smooth the value estimate by incorporating information from similar state-action pairs, making the learning process more robust to inaccuracies in both the actor and the critic. The smoothed target action can be represented as:

$$a' = \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \quad (5.12)$$

where a' is the smoothed target action, $\pi_{\phi'}(s')$ is the action predicted by the target policy network, and ϵ is clipped Gaussian noise.

These modification jointly solved several challenges in deep reinforcement learning, particularly in continuous action domains. By reducing overestimation bias, TD3 mitigates the risk of the agent exploiting errors in the Q-function, which can lead to suboptimal or divergent behavior. The delayed policy updates and target policy smoothing contribute to more stable learning dynamics, allowing the algorithm to converge more reliably to high-performing policies.

The policy update in TD3 is performed less frequently than the critic updates and can be expressed as:

$$\phi = \arg \max_{\phi} \mathbb{E}_{s \sim \mathcal{D}} [Q\theta_1(s, \pi_{\phi}(s))] \quad (5.13)$$

where ϕ represents the parameters of the policy network, and $\mathcal{D}$ is the replay buffer.

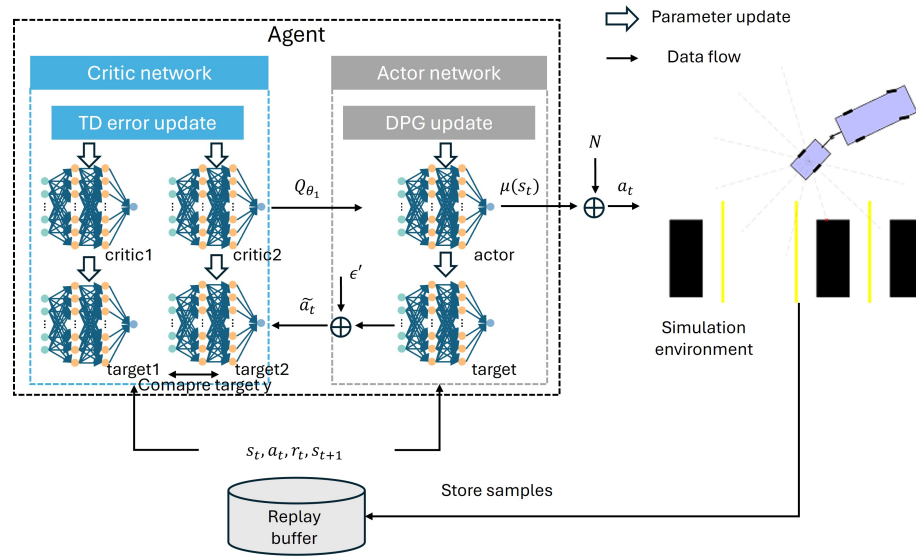


Figure 5.14: Structure of the TD3 algorithm, including the actor and critic networks.

Research experiments and practical tests have shown that TD3 performs better than DDPG and other similar algorithms in many different types of continuous control tasks[77]. TD3 is more reliable and learns more efficiently from the same amount of data. These advantages make TD3 a good choice for complicated robotic control problems. For truck-trailer reverse parking, TD3's ability to handle continuous action spaces and its robust learning dynamics make it a good start for developing effective control strategies.

5.4.2 Enhanced TD3 with Adaptive Weight and Learning Rate Decay

Weight decay and learning rate decay are fundamental techniques in machine learning optimization. Weight decay, a form of L2 regularization, penalizes large model parameters to prevent overfitting and improve the model's generalization; Learning rate decay gradually reduces the step size during training, allowing for finer adjustments as the model converges. This section introduce the integration of adaptive weight decay and learning rate decay into the TD3 framework, to enhance the algorithm's performance in complex control tasks such as truck-trailer reverse parking. By dynamically adjusting regularization and optimization parameters, these modifications seek to improve the algorithm's stability, convergence, and generalization capabilities in high-dimensional continuous action spaces.

The fundamental principle of weight decay is to augment the standard loss function L with a penalty term proportional to the squared magnitude of the model parameters:

$$L_{\text{total}} = L + \lambda \sum_i w_i^2 \quad (5.14)$$

where L_{total} is the total loss, L is the original loss function, λ is the weight decay coefficient, and w_i are the individual weights of the neural network.

In the adaptive variant, λ is not a fixed hyperparameter but rather a function which changes during training:

$$\lambda_t = f(\lambda_{t-1}, \nabla L, \text{model performance}) \quad (5.15)$$

where λ_t is the weight decay coefficient at time step t , and f is an adaptive function that considers the previous coefficient, the gradient of the loss, and current model performance metrics.

Besides the adaptive weight decay, this chapter also implement learning rate decay to fine-tune the TD3 optimization process over time. The learning rate α is adjusted according to a decay schedule:

$$\alpha_t = \alpha_0 \cdot g(t) \quad (5.16)$$

where α_0 is the initial learning rate, t is the current training step, and $g(t)$ is a decay function. Common decay functions include step decay, exponential decay, or cosine annealing:

$$g(t) = \begin{cases} \gamma^{\lfloor t/k \rfloor} & \text{step decay} \\ e^{-\beta t} & \text{exponential decay} \\ \frac{1}{2}(1 + \cos(\frac{\pi t}{T})) & \text{cosine annealing} \end{cases} \quad (5.17)$$

where γ , β , k , and T are hyperparameters controlling the decay rate and schedule.

The incorporation of adaptive weight decay and learning rate decay into the TD3 framework help the overall training process in the development of robust and efficient autonomous parking systems for TTRV. The weight decay modification makes the control policy tend to be simpler, which is good for understanding how the neural netowrk works.

At the same time, learning rate decay allows for small, careful adjustments along training process, this approach also helps choose which parts of the state information are most important, making the system more efficient. By avoiding large the weights and changing the learning rate as needed, the learning process becomes more stable. The designed algorithm adjusts both weight decay (λ) and learning rate (α) to find the balance between having a simple model, fitting the data well, and learning quickly. Lastly, changing the learning rate helps the system avoid getting stuck in suboptimal solutions early in the training process, while still be able to find good solution later on. All these improvements work together to make the TD3 method more effective for complex control tasks.

5.4.3 Implementation and Simulation

The simulation for TD3 and improved TD3WD is conducted in the same Gym environment as PPO, maintaining consistent initial conditions and training duration for both algorithms. The proposed TD3WD controller's efficacy was evaluated against the TD3 algorithm through this simulations result. Figure 5.15 illustrates the trailer parking trajectory achieved by the TD3 and TD3WD controller. In the Unity visualization, this trajectory is very similarity to the one produced by PPO algorithm, suggesting comparable performance in executing parking maneuver.

The performance of the proposed TD3WD controller was evaluated against the standard TD3 by evaluating three key metrics were analyzed: success rate, mean episode length, and mean reward. The results are presented in Figure 5.16, Figure 5.17, and Figure 5.18.

Figure 5.16 illustrates the success rate of both algorithms over 5 million training steps. The TD3WD shows better performance and stability compared to the standard TD3 and significantly improvement over the previously discussed PPO algorithm. TD3WD achieves a higher success rate more quickly and maintain the success rate consistently



Figure 5.15: Unity visualization of trailer parking in obstacle-free environment using TD3WD.

throughout the training process. This improved performance should be attributed to the adaptive weight decay modification, which enhances the model's generalization capabilities. The smoother curve of TD3WD also indicates better robustness to variations in the environment. However, both TD3 variants show some fluctuations after the 3 million step mark, which suggests overfitting or sensitivity to specific training scenarios. And also these success rate stability is better compared with the performance of PPO.

Figure 5.17 shows the mean episode length for both algorithms. After an initial spike, both TD3 and TD3WD converge to similar episode lengths, with TD3WD showing slightly more stability. The better episode length stability of TD3WD suggests that it may find more direct or efficient paths to the goal state, benefited from the adaptive learning rate.

Figure 5.18 shows the mean reward obtained by both algorithms over time. TD3WD shows a faster initial increase in mean reward and maintains a slightly higher and

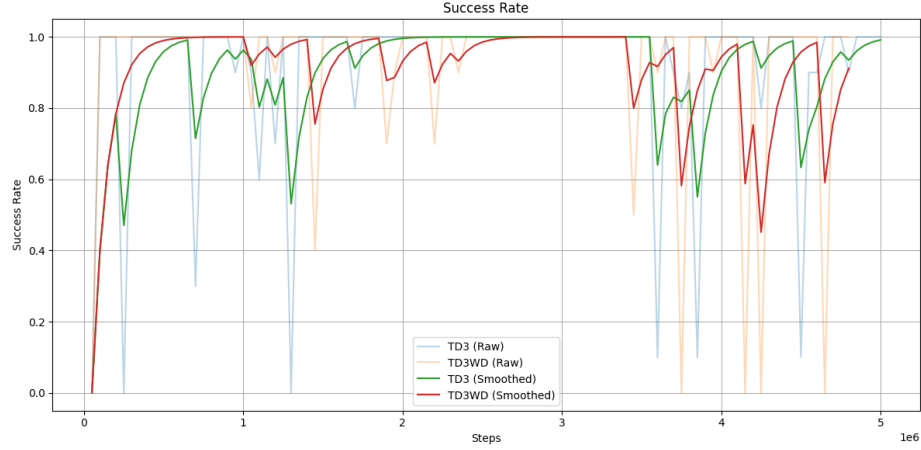


Figure 5.16: Comparison of success rates between TD3 and TD3WD.

stable reward level throughout most of the training process. This aligns with the observations from the success rate graph, indicating that TD3WD not only succeeds more often but also achieves higher-quality solutions. Compared to PPO, both TD3 variants demonstrate improved stability and consistently higher rewards, especially in the latter half of training.

The improved performance of TD3WD across all three evaluation criterias shows its robustness and learning efficiency compared to both standard TD3 and PPO. The adaptive weight and learning rate decay seems to contribute to the faster learning speed, improved stability, and better generalization. Although the observed performance drops after 3 million steps, they are less severe than the performance drop from PPO. These performance instability could be caused by overfitting, when the models begin to fit to specific scenarios rather than generalizing effectively.

Start from TD3 and TD3WD in effectively learning trailer parking policies, this research future explored the integration of imitation learning, which used pre-collected expert demonstrations dataset mixed with random exploration states for training. Unlike

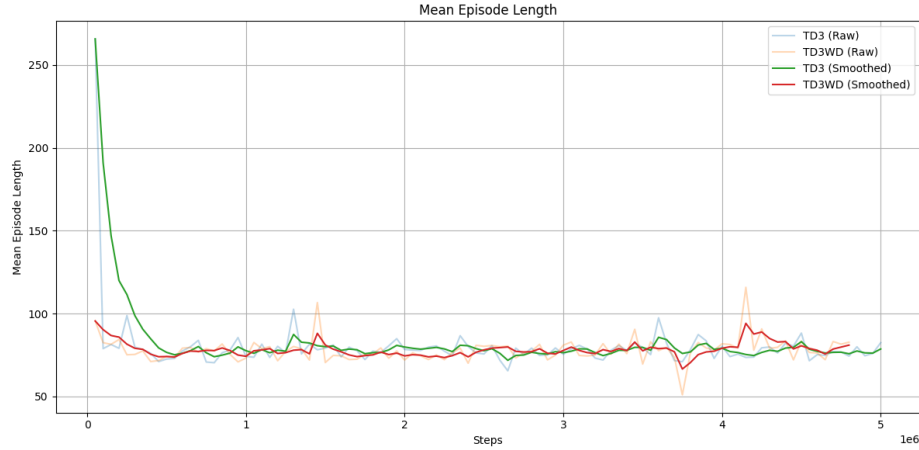


Figure 5.17: Comparison of mean episode Lengths between TD3 and TD3WD.

PPO, which is an on-policy algorithm, TD3 and TD3WD’s off-policy feature allows them to learn from experiences generated by other policies or collected from the real world. By partially initializing the experience replay buffer with expert trajectories, TD3 and TD3WD benefit from prior knowledge to accelerate learning and improved performance, as shown in Figure 5.19. This approach, known as demonstration-augmented reinforcement learning, combines the strengths of imitation learning with the adaptability of reinforcement learning. The following experiments shows how this hybrid approach can impact learning efficiency, stability, and overall performance in the challenging task of autonomous trailer parking, and outperform PPO in scenarios where expert demonstrations are available.

Figure 5.20 illustrates the TD3WD controller’s performance in the moose test scenario, showing capabilities of learnt obstacle avoidance strategy. The stacked trajectory visualization shows the TD3WD controller successfully navigating the trailer through the cones in parking trajectories, and the successful parking under the moose test simulation

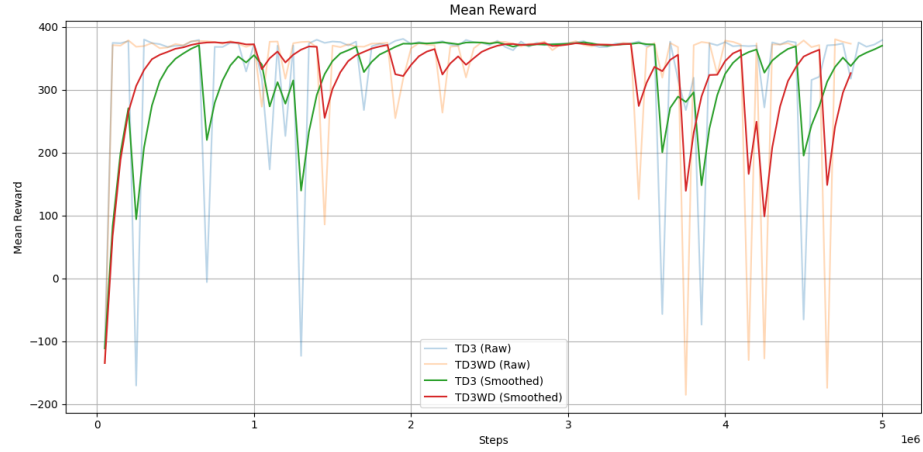


Figure 5.18: Comparison of mean rewards between TD3 and TD3WD.

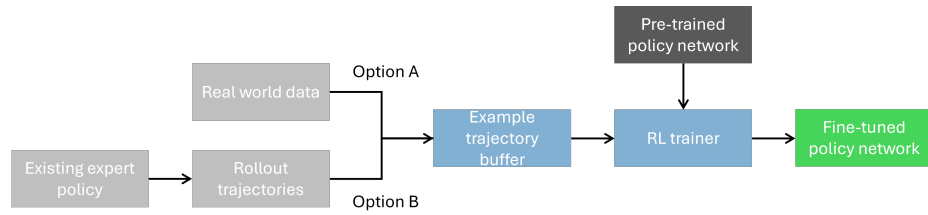


Figure 5.19: Diagram of imitation learning with expert demonstrations.

shows TD3WD’s potential for real-world applications, particularly in situations requiring both accuracy and static obstacle avoidance.

5.5 Chapter Conclusion

This chapter explores the application of model-free deep reinforcement learning to reverse parking control of TTRV, and two categories of DRL are introduced: the on-policy approach PPO and the off-policy approach TD3WD.

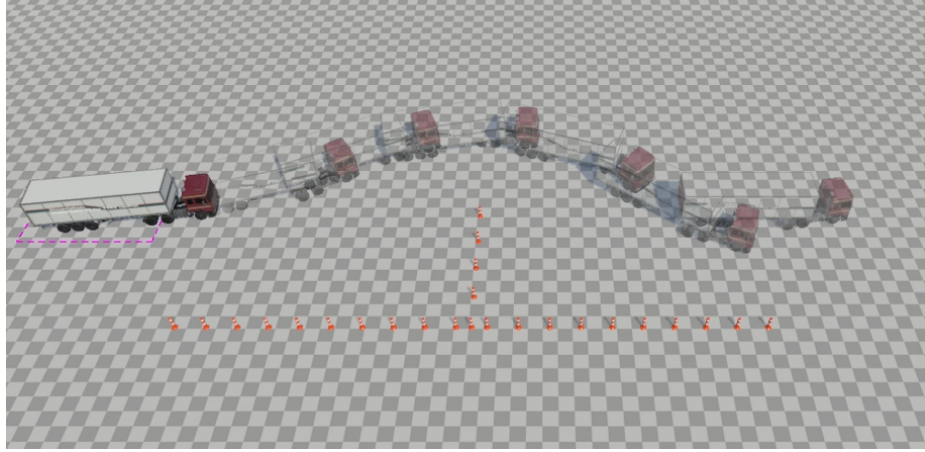


Figure 5.20: Unity visualization of trailer parking with obstacles in an evasive maneuver test using TD3WD.

The research demonstrate the effectiveness of model-free DRL in mastering TTRV reverse maneuvering capability, including high-dimensional state spaces and nonlinear dynamics. Both PPO and TD3WD successfully learns the parking strategies through directly interacting with the simulation environment. TD3WD, which is improved with adaptive weight decay and learning rate decay, showed improved performance, stability, and learning efficiency compared with traditional PPO and TD3WD algorithm. However, these approaches some limitations including potential instability and hyperparameter sensitivity, and these limitations suggest further exploration of integrating model-based approaches into the reinforcement learning process, and which will be introduced in the following chapters.

CHAPTER SIX

HYBRID MODEL-BASED REINFORCEMENT LEARNING FOR TTRV TRAJECTORY FOLLOWING AND PARKING CONTROL

This chapter explores the application of Model-based Reinforcement Learning to TTRV reverse parking control, which is designed to improve the sample efficiency and interpretability compared with model-free approaches[78]. This chapter focus on a hybrid approach which combines a known trajectory state generation model with model-free reinforcement learning techniques, this method falls under the category of MBRL with a known model, where only the global policy is learned through planning over the model.

In the proposed hybrid MBRL method, it integrate a learned trajectory state model to predict future states for trailer, and this model serve as a planning module for efficient policy optimization. This chapter also investigate strategies for integrating this trajectory state prediciton model with model-free algorithms, to combine the pros of both methdos. Additionally, this chapter explores the model uncertainty estimation and predictive model refinement over time[79].

6.1 Comparison Between Model-free And Model-based RL

Nowadays, the development of Reinforcement Learning algorithm can be roughly categorized into two: model-free and model-based approaches[80]. The major difference between these two approaches lies in their access to the MDP dynamics and the way they learn the solution[78][81].

In MFRL methods, the agent is designed to learn a policy or value function directly from the interactions with environment, without explicitly modeling the dynamics of the environment[80]. This approach treat the environment like a black box, only focus

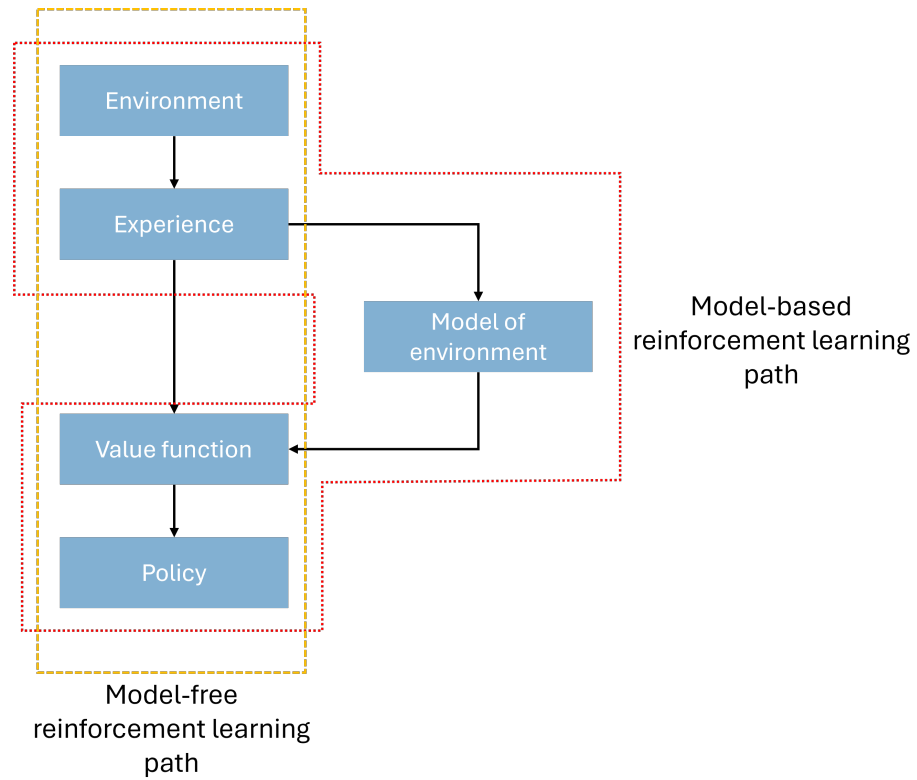


Figure 6.1: Distinction between model-based and model-free reinforcement learning based on the presence of a model and a global solution.

on the mapping relationship between states, actions, and rewards. MFRL methods typically store a local solution, which is a value function or policy for a specific state or state-action pair. This local representation allows for efficient updates and provides certain scalability to large state spaces.

One feature of MFRL methods is the irreversible access to the MDP dynamics, meaning the agent can only progress forward from the resulting next state after executing an action. This irreversibility constrains the agent's reasoning capability which help explore hypothetical scenarios or "undo" actions, but it makes MFRL methods suitable for online learning in dynamic environments. Despite the local nature of MFRL updates, MFRL methods eventually aim to train a global solution, which represents a value function or policy for the entire state space[16].

By learning action policies directly from trajectory data collected through environment interactions, an MFRL agent can adapt to complex, non-linear dynamics without precise environment model. This data-driven approach succeeded in many fields, from game playing to robotics. However, it often requires a large number of samples to achieve good performance, especially in high-dimensional state spaces or environments with sparse rewards. The sample inefficiency of MFRL methods is one of the primary limitations, particularly in scenarios where interactions with the environment can be costly or time-consuming. Although faced with these challenges, the flexibility and generality of MFRL methods continue help them to be a popular choice in modern reinforcement learning research or application.

In comparison, the Model-based Reinforcement Learning agent learns a environment model and consistently keep learning and improving optimal action policies. MBRL methods have reversible access to the MDP dynamics, allowing the agent to query the MDP at any preferred state-action pair or allowing the agent to plan ahead[80]. This reversible access to the MDP dynamics is called a "model", which can be either known

Table 6.1: RL categories comparison based on the presence of a model and a global solution.

	Model	Global solution
Planning	+	-
Reinforcement learning	+/-	+
Model-free reinforcement learning	-	+
Model-based reinforcement learning	+	+

before or learned during the training. The model typically represents the transition dynamics $P(s'|s, a)$ and sometimes the reward function $R(s, a, s')$ of the environment. The main benefit of MBRL is its potential for sample efficiency. By using the learned or known model, agent can perform "mental simulations" without interacting with the actual environment, potentially reducing the number of real-world interactions required for model learning[82]. This is particularly valuable in scenarios where real-world interactions are costly, time-consuming, or potentially dangerous.

MBRL can be further categorized into three subcategories based on how learning is integrated with planning:

- Jointly learning the model and policy: In this category, the model and policy are learned at the same time. The model is used to generate simulated trajectories, which are then used to update the policy. This approach is known as model-based policy optimization. Example algorithms such as Model-based Policy Optimization (MBPO)[83], iteratively improve both the model and the policy. The advantage of this approach is that it can adapt the model learning to focus on regions of the state space relevant to policy improvement.
- Learning the policy over a known model: In this category, the model is known, and the policy is learned through planning over the model. This approach is known as

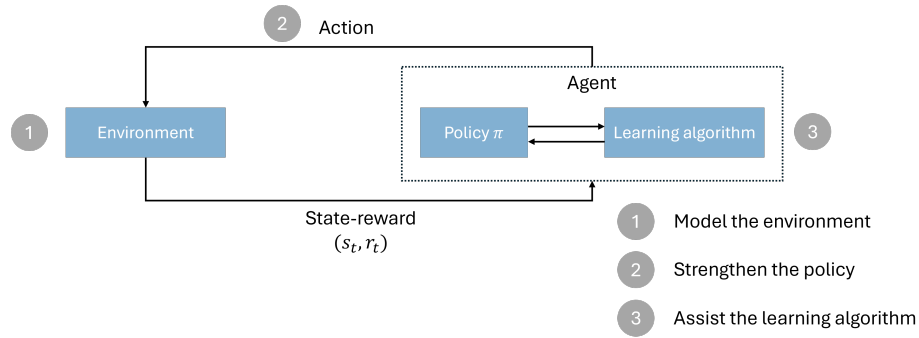


Figure 6.2: Different stages in which the learned dynamics models can be employed.

model-based planning. Techniques like Monte Carlo Tree Search (MCTS) belongs into this category. The advantage is when the model is accurate, the policy can be optimized without risking costly or dangerous real-world interactions.

- **Learning a model and performing local planning:** In this category, the model is learned, and planning is performed over the model without explicitly learning a global policy. This approach is known as model-based control. Methods like MPC use this approach, where a trajectory is planned over a short horizon using the learned model, and only the first action is executed before replanning.

As described in Figure 6.2, the model can be used at different stages during the agent learning process of the MBRL. These stages represent different ways of leveraging the model:

- **Model-based policy optimization:** The model is used to generate simulated trajectories, which are then used to update the policy. This approach allows for off-policy learning and can significantly reduce the number of real-world interactions needed. However, it relies heavily on the accuracy of the learned model, and model errors can compound over long horizons.

- Model-based planning: The model is used to plan ahead and generate a policy, which is then executed in the environment. This approach can produce more accurate policies if the model is reliable, as it can consider long-term consequences of actions. However, it can be computationally expensive, especially for large state spaces.
- Model-based control: The model is used to generate a policy, which is then executed in the environment. This approach, often implemented as Model Predictive Control, combines the benefits of planning with the robustness of short-horizon decision making. It can adapt quickly to unexpected changes in the environment but may struggle with long-term strategic planning.

One of the key challenges in MBRL is managing model uncertainty. Since the learned model is an approximation of the true environment dynamics, it will inevitably have errors. These errors can grow along long trajectories, leading to a phenomenon known as "model bias"[84]. Many techniques have been developed to mitigate this challenge. Ensemble methods, which involve training multiple models and aggregating their predictions, help reduce individual model errors. Probabilistic models, such as Gaussian Processes or Bayesian Neural Networks, represent model uncertainty explicitly. Additionally, model-based uncertainty-aware planning incorporates model uncertainty into the planning process to avoid overly optimistic predictions. These approaches collectively aim to mitigate the impact of model uncertainty and improve the robustness of model-based reinforcement learning systems.

6.2 Trajectory Planner Based on Hybrid A* algorithm

As described in section 3.4, the Hybrid A* algorithm is suitable for generating driving paths for the TTRV parking task. By combining the benefits of grid-based search

and continuous optimization, Hybrid A* efficiently explore the environment state space and find feasible paths which satisfy TTRV kinematic constraints. In this section, the Hybrid A* algorithm is used to generate a trajectory state model for the TTRV parking control task.

Building upon the Hybrid A* algorithm introduced earlier, this chapter now present a modified version tailored specifically for TTRV reverse driving. This adaptation incorporates the unique kinematic constraints of the TTRV system and accounts for the challenges of reverse maneuvering and obstacle avoidance.

The kinematic equations which defines the TTRV's motion during reverse driving are introduced in Equations (2.9) to (2.11) and (2.13) to (2.15), and here the simplified state space with only $[x_2, y_2, \theta_2, \phi]$ is used:

$$\begin{aligned} \dot{x}_2 &= -v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \cos \theta_2 \\ \dot{y}_2 &= -v \cos \phi \left(1 - \frac{L_2}{L_1} \tan \phi \tan \delta\right) \sin \theta_2 \\ \dot{\theta}_2 &= v \left(\frac{\sin \phi}{L_3} + \frac{L_2}{L_1 L_3} \cos \phi \tan \delta\right) \\ \dot{\phi} &= \frac{v}{L_3} \sin \phi + \frac{v}{L_1} \left(1 + \frac{L_2 \cos \phi}{L_3}\right) \tan \delta \end{aligned} \tag{6.1}$$

The cost function for this modified Hybrid A* incorporates additional terms to account for reverse driving challenges:

$$f(n) = g(n) + h(n) + w_1 \cdot s(n) + w_2 \cdot r(n) + w_3 \cdot o(n) \tag{6.2}$$

where $g(n)$ is the cost to reach node n , $h(n)$ is the heuristic estimate, $s(n)$ is a smoothness penalty, $r(n)$ is a reverse driving which encourage the vehicle try to drive forward, and $o(n)$ is an obstacle proximity penalty. The weights w_1 , w_2 , and w_3 balance these different factors.

To ensure the generated trajectory and satisfy the TTRV's motion constraints and avoids obstacles, the following constraints are considered:

- Jack-knife prevention: Limit the angle ϕ between the truck and trailer to prevent jack-knifing, if the ϕ is beyond certain value, then the TTRV is considered to be near jack-knife constrain, and a penalty is given with episode failure.
- Obstacle avoidance: Use a configuration space representation that accounts for the safe distance of the TTRV during reverse motion.
- Curvature constraints: Limit the steering angle δ to satisfy the vehicle's minimum turning radius.

The resulting trajectory state model provides feasible parking trajectory states and actions which can guide the TTRV from its initial state to the desired parking state and take obstacles avoidance and kinematic constraints into consideration. This model is used as a foundation for the proposed model-based reinforcement learning approach, providing structured exploration for path generation and expert demonstrations for the agent's learning.

To improve the training efficiency of model-based approach, this chapter combines PPO with the trajectory following state model. The trajectory state model serves as a foundation model for the neural network, parameterized as $f_{\theta}(s_t, a_t)$. The observation states s_t , as introduced above, is the tracking error between the trailer reference point and trajectory reference points, while the action input of the foundation model is the truck steering angle Δx . The trajectory state model takes input of the current state s_t and action a_t . If the system states deviation is beyond certain threshold, the trajectory from the previous cycle is optimized considering the current environment constraints, and newly predicted states $\hat{s}_{t+1}$ are generated, providing query access to MDP dynamics.

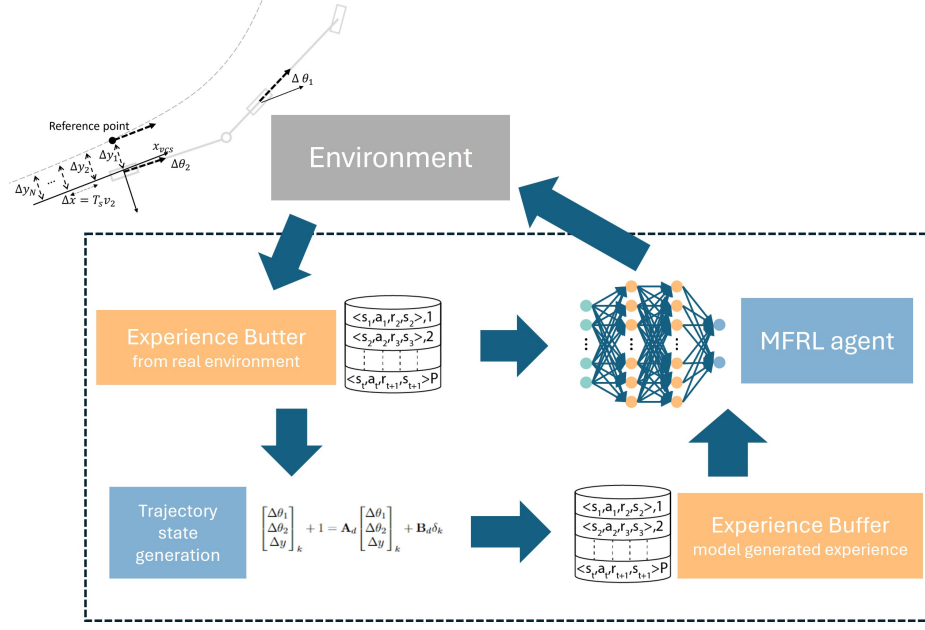


Figure 6.3: Integration of trajectory state model-based reinforcement learning with PPO.

Figure 6.3 shows the integration of trajectory state model-based reinforcement learning with PPO. The hybrid A* trajectory state planning model acts as a planning module, generating predicted next states based on current states and actions. Meanwhile, PPO serves as the backbone, learning the optimal policy for the trajectory following task. This hybrid approach leverages the strengths of both model-based and model-free methods, enabling efficient and effective learning for the TTRV parking control problem.

6.3 Integration of Model-based and Model-free Approaches

In the previous chapter, the TTRV kinematics planner generates system states, and the state variables are used in predefined hybrid A* reference path within trajectory planning module to calculate deviation states. Equation 6.3 presents a simplified TTRV's

lateral dynamics model, which is used in the section 4.3 and offers a efficient representation of vehicle's kinematic behavior.

$$\begin{bmatrix} \Delta \dot{\theta}_1 \\ \Delta \dot{\theta}_2 \\ \Delta \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{v}{L_3} & -\frac{v}{L_3} & 0 \\ 0 & v & 0 \end{bmatrix} \begin{bmatrix} \Delta \theta_1 \\ \Delta \theta_2 \\ \Delta y \end{bmatrix} + \begin{bmatrix} \frac{v}{L_1} \\ -\frac{L_2}{L_1 L_3} v \\ 0 \end{bmatrix} \delta \quad (6.3)$$

Because RL has the flexibility to train the controller based on different observation space, this research augmented the original observation space from LQR with the ϕ value to encourage the truck using smaller truck-trailer angle during reverse parking. as shown in Figure 6.4, the reference states error, which includes $\Delta \theta_1$, $\Delta \theta_2$, Δy , and Φ , which is different from the single reference point and deviation state used in LQR controller shown in Figure 4.10.. These represent heading error between truck's heading angle and reference point heading angle, heading error between trailer heading angle and reference point heading angle, lateral distance in trailer coordinate between trailer position and reference point, and the angle between the truck and trailer direction, respectively.

In controller applications, the preview distance is considered into system for improved driving consistency. Figure 6.4 shows the first N preview reference points and deviation value plotted within trailer coordinate. The reference error value for each preview reference sample point is calculated and denoted by $[\theta_{1_k}, \theta_{2_k}, y_k]$. Δx is sampling interval, and T_s refers to simulation step time. Moreover, Equation 6.3 requires discretization in simulation.

$$\begin{aligned} \mathbf{A}_d &= e^{\mathbf{A}T} \\ \mathbf{B}_d &= \mathbf{A}^{-1} \left(e^{\mathbf{A}T} - \mathbf{I} \right) \mathbf{B} \end{aligned} \quad (6.4)$$

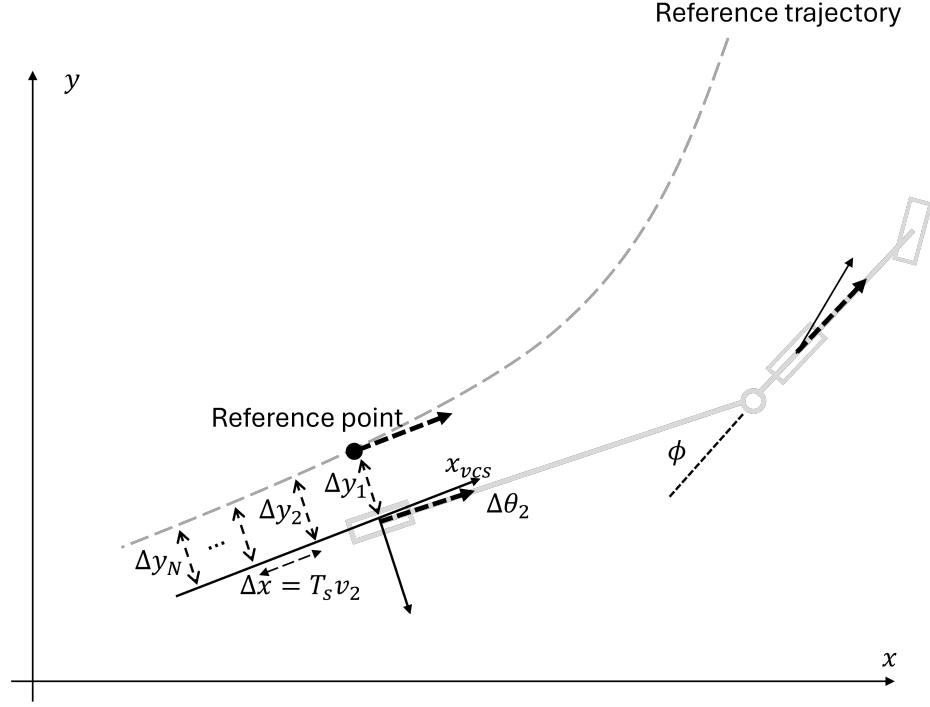


Figure 6.4: Vehicle trajectory state model considering preview reference points and deviation states.

Therefore, the N preview sample points can be computed by the equations, where A_r and B_r are defined.

$$\begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \Delta y \end{bmatrix}_k + 1 = \mathbf{A}_d \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \Delta y \end{bmatrix}_k + \mathbf{B}_d \delta_k \quad (6.5)$$

where

$$A_r = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}_{(N \times N)} \quad (6.6)$$

$$B_r = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}_{(N \times 1)} \quad (6.7)$$

6.4 Reward Design for Truck-Trailer Path Following

In reinforcement learning for autonomous vehicle control, the design of an appropriate reward function is crucial for effective learning and performance. For the TTRV model-based planning and trajectory following, a comprehensive reward function has been developed that encourages the system to maintain alignment with the desired trajectory while considering multiple factors. This reward function is inspired by the principles of optimal control and the specific challenges of articulated vehicle navigation[85].

The total reward R_{total} is computed as a weighted sum of three main components, with additional bonuses or penalties for episode outcomes:

$$R_{\text{total}} = w_{\phi} R_{\phi} + w_{\delta} R_{\delta} + w_d R_d + R_{\text{outcome}} \quad (6.8)$$

In the above reward expression, the following terms are defined:

- R_{ϕ} is the articulation angle alignment reward
- R_{δ} is the steering angle reward
- R_d is the trajectory deviation reward
- R_{outcome} is the episode outcome reward

- w_ϕ, w_δ, w_d are the respective weights

The articulation angle alignment reward R_ϕ penalizes misalignment between the truck and trailer, which encourage the system to use smaller truck-trailer angle during reverse parking:

$$R_\phi = -\sin^2(\phi) \quad (6.9)$$

where ϕ is the angle between the truck and trailer. This quadratic penalty on the sine of the articulation angle is designed to keep the truck and trailer aligned, while avoiding close to the jackknife situations.

The steering angle reward R_δ discourages excessive steering:

$$R_\delta = -\left(\frac{\delta}{\delta_{\max}}\right)^2 \quad (6.10)$$

where δ is the current steering angle and $\delta_{\max}$ is the maximum allowable steering angle. This quadratic penalty on normalized steering angle encourages smoother control actions and helps to prevent frequent steering change, which is particularly important for the stability of TTRV[38].

The trajectory deviation reward R_d encourages the system to stay close to the desired path:

$$R_d = \exp(-\mathbf{e}^T \mathbf{W} \mathbf{e}) \quad (6.11)$$

where $\mathbf{e}$ is the lateral error vector and $\mathbf{W}$ is a weight matrix for different error components. This exponential form provides a continuous, differentiable signal that becomes increasingly negative as the system deviates from the desired path. The use of a weighted

quadratic form inside an exponential function is inspired by optimal control techniques and allows for fine-tuning the sensitivity to different types of path deviations[37].

The episode reward R_{outcome} provides a significant bonus for successful completion or a penalty for failure:

$$R_{\text{outcome}} = \begin{cases} 300 & \text{if episode successful} \\ -200 & \text{if episode failed} \\ 0 & \text{otherwise} \end{cases} \quad (6.12)$$

This reward component helps to shape the long-term behavior of the agent, encouraging it to complete the path following task successfully. The asymmetry between the success bonus (300) and failure penalty (-200) is designed to provide a slightly optimized bias, especially helpful for the exploration during the early stages of learning[80].

An important feature of this reward function is the termination condition based on the TTRV observation space. If the lateral error $\mathbf{e}$ is outside of the defined observation space, current episode immediately terminates with the proposed failure penalty, because such state is always from an inefficient trajectory. This helps to constrain the learning process to feasible regions of the state space and encourages the agent to maintain control within reasonable bounds, and the failure episode can be detected and terminated in early stage.

The combination of dense rewards (articulation angle, steering angle, and trajectory deviation) with sparse rewards (episode outcomes) is a technique known to be effective in reinforcement learning for robotics and control tasks. The dense rewards provide a continuous learning signal that guides the agent towards better performance, while the sparse rewards help to align the agent's behavior with the overall task objectives.

This reward function is designed to guide the reinforcement learning agent towards smooth, accurate trajectory following while avoiding jack-knife situations and excessive steering. By incorporating multiple factors relevant to truck-trailer control, it provides a rich learning signal that captures the complexities of the task.

6.5 Implementation and Simulation

To evaluate the performance of the trajectory state model-based reinforcement learning approach for TTRV parking control, the agent was trained for 5 million steps with data collected on several performance metrics throughout the training process. Because of this fast progress, only the data from these first 3 million steps is shown here, giving a clearer picture of the key learning period. The results of these simulations are presented in Figure 6.8, Figure 6.5, Figure 6.6, Figure 6.7, and Figure 6.9, which illustrate the training loss, mean episode length, mean reward, success rate, explained variance and entropy loss during training respectively. These figures together give a full picture of how the agent learns and improves over simulation time. Also the agent is transfer to a new environment with different obstacles to test the generalization capability of the model.

Figure 6.5 shows the mean episode length over time. Interestingly, the episode length quickly increases from an initial value of about 45 which suggests a quick failure to around 80 which is roughly the successful trajectory length within the first 100,000 steps. This rapid increase suggests that the agent learns how to avoid early terminations and maintain control for longer periods. After this initial jump, the mean episode length remains relatively stable between 75 and 85 steps, with some minor fluctuations. This stability indicates that the agent has learned to consistently complete the parking task within a similar number of steps, regardless of the starting conditions.

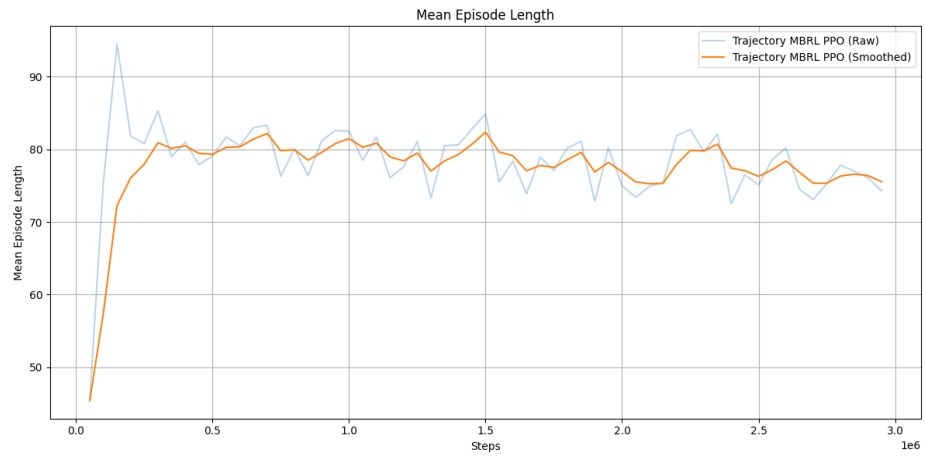


Figure 6.5: Mean episode length over training steps.

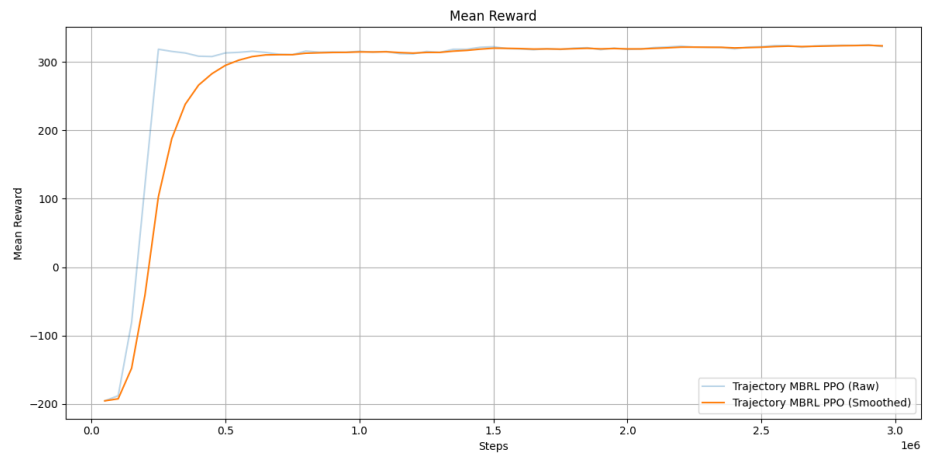


Figure 6.6: Mean reward per episode over training steps.

Figure 6.6 presents the mean reward per episode over the training period. A early improvement in the mean reward is observed during the first 500,000 steps, with the reward increasing from an initial value of about -200 to over 300. This rapid increase aligns with the sharp decrease in training loss observed in Figure 6.8, further conforms the agent’s quick learning during the initial stage. After reaching a reward of around 300, the improvement rate slows down, with the reward gradually increasing to around 320 untile the end of training process. This pattern suggests that the agent quickly learns to complete the parking task successfully and then focuses on fine-tuning its performance to maximize the reward.

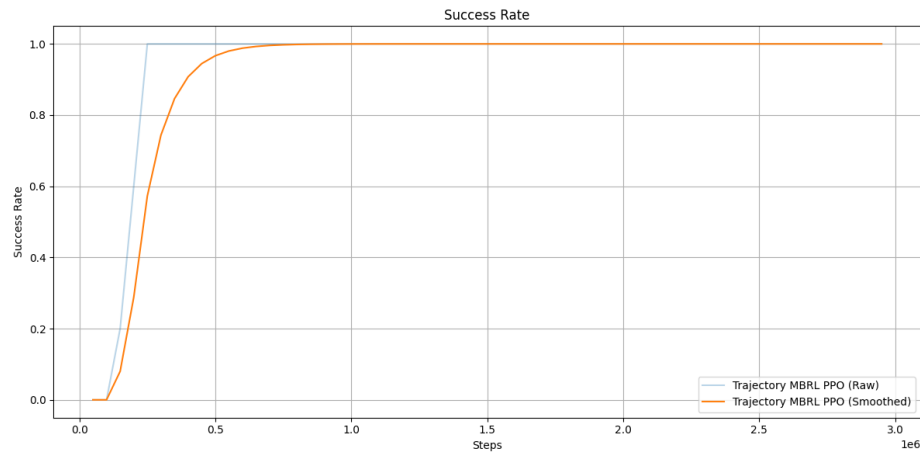


Figure 6.7: Success rate over training steps.

Figure 6.7 shows the success rate of the agent over time. The success rate shows a sharp increase from 0 to nearly 1.0 within the first 500,000 steps. This rapid improvement in success rate corresponds well with the trends observed in the mean reward and training loss graphs. After reaching a success rate of about 0.95, the agent continues to improve

slightly, eventually achieving a near-perfect success rate of 0.99 by the end of training. This high success rate demonstrates the effectiveness of the trajectory state model-based reinforcement learning approach in solving the TTRV parking control problem.

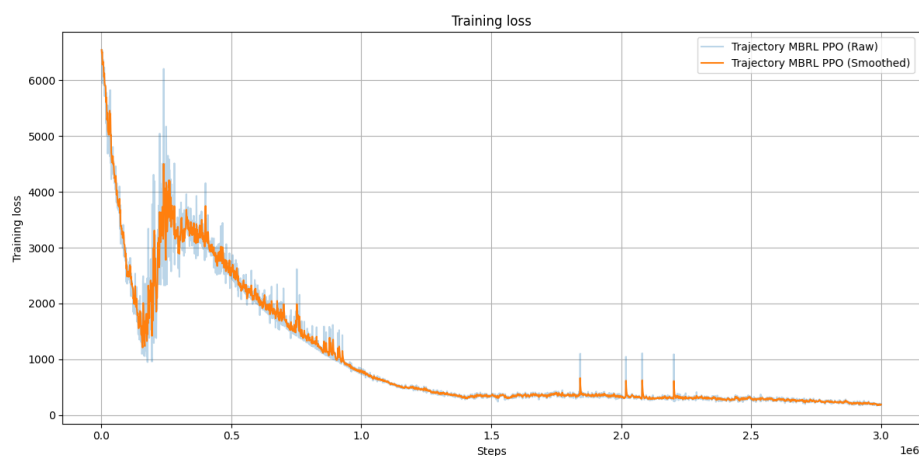


Figure 6.8: Training loss over 3 million steps.

Figure 6.8 and Figure 6.9 show the training loss and explained variance over 3 million steps respectively. Both metrics display rapid initial improvement: the loss drops sharply from 6000, while explained variance rises quickly from 0 to 0.6. After this initial phase, both metrics show more gradual changes. The loss stabilizes around 500 after 1.5 million steps, while explained variance reaches 0.8 by 3 million steps, eventually stabilizing near 0.9. These patterns indicate that the agent quickly learns the basics, then refines its performance more slowly during further successful training.

Figure 6.10 shows the trajectory state model-based reinforcement learning agent maneuvering a "3" shaped path with stacked screenshots. The agent proves its capability to handle non-linear paths, as well as making smooth transitions between different

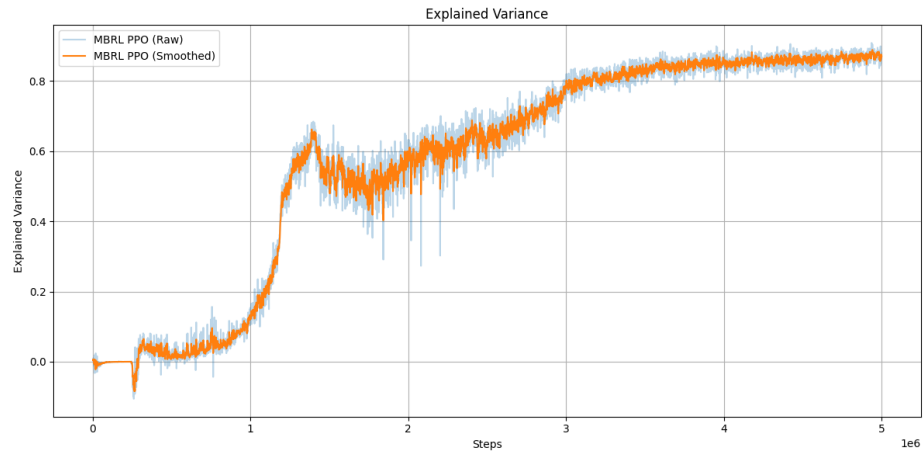


Figure 6.9: Explained variance over training steps.

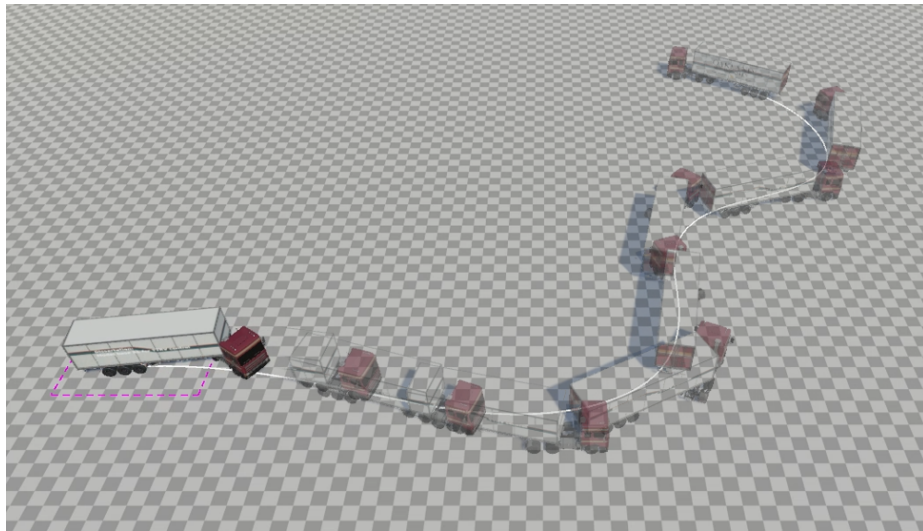


Figure 6.10: Trajectory state model-based reinforcement learning agent maneuvering a "3" shaped path.

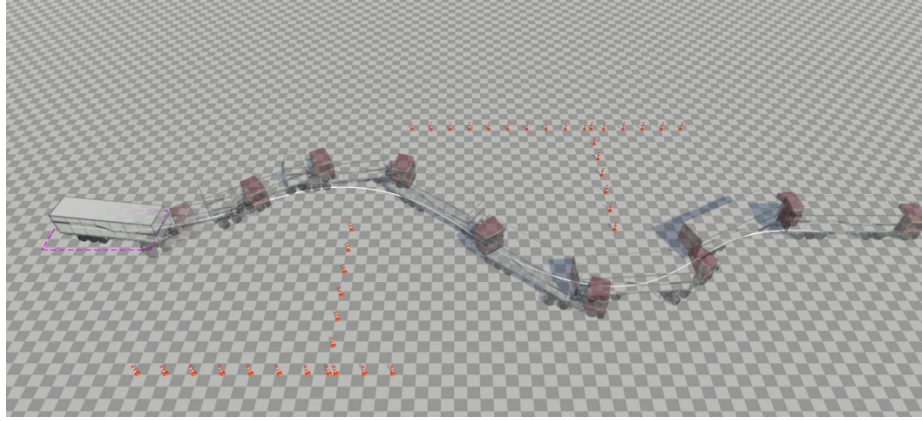


Figure 6.11: Trajectory state model-based reinforcement learning agent maneuvering a double moose path.

curvatures. This complex maneuver showed that the agent's capability to plan and execute accurate maneuver, using the trajectory state model to predict and optimize its path through challenging configurations.

Figure 6.11 shows the agent navigating a double moose test scenario, in which traffic cones are placed in an S shape to simulate the obstacle, road, and road edges. The visualization demonstrates the agent's ability to perform rapid evasive maneuvers. The double moose test is a more challenging test scenario which requires two consecutive and opposite maneuvering. The smooth trajectory suggests effective use of the model's predictive capabilities to anticipate and respond to the sharp directional changes required in this advanced obstacle avoidance scenario.

Figure 6.12 shows the trajectory state model-based reinforcement learning agent executing a narrow U-turn maneuver. The stacked screenshots show that it navigates a tight 180-degree turn within a bounded space, and also the TTRV needs to have both forward and backward driving adjustment. This challenging scenario tests the agent's ability to plan complex trajectories and maneuver itself in limited parking area, which

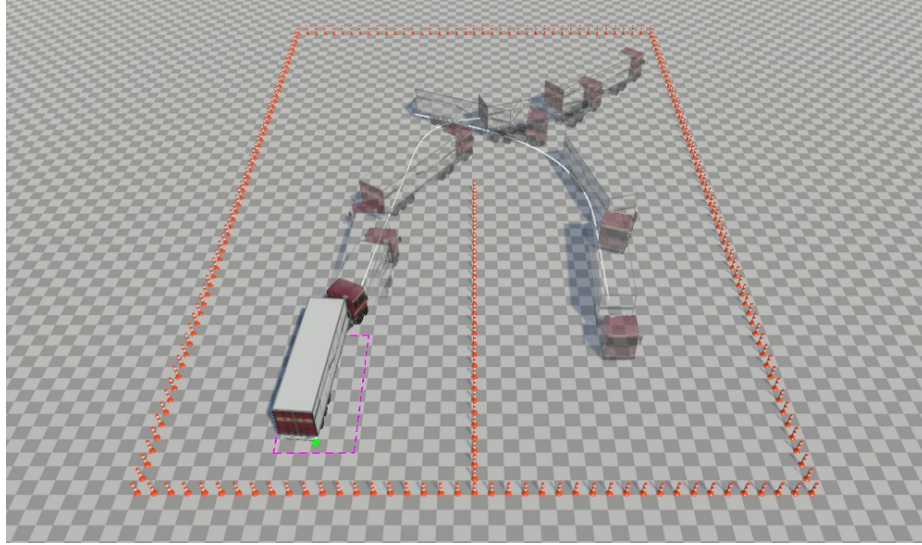


Figure 6.12: Trajectory state model-based reinforcement learning agent maneuvering a narrow U-turn path with direction change.

requires a precise control of both the truck and trailer. The successful completion of this maneuver proves the agent's ability in utilizing the trajectory state model to predict the system's behavior and its potential for real-world applications in constrained environments.

The simulation results demonstrate the effectiveness of the proposed trajectory state model-based reinforcement learning approach for TTRV parking control. Several important findings has been introduced, including rapid initial learning, consistent policy refinement, and high success rates across various starting conditions. The evaluation cretirias shows that the integration of trajectory state model with reinforcement learning enables the effective planning and learnign for TTRV reverse parking control. The stability in episode length and consistent high performance indicates the robustness of learned policy in different scenarios.

6.6 Chapter Conclusion

This chapter introduced a hybrid model-based reinforcement learning approach for TTRV trajectory following and parking control. It begins by comparing model-free and model-based reinforcement learning methods, highlighting the advantages of the using model based approach for TTRV controller design. A trajectory planner based on the Hybrid A* algorithm is introduced, followed by the introduction of how to integrate this trajectory state model with the model-free approaches PPO. Also, the reward for truck-trailer path following is introduced, which is important because the well designed reward mechanism help to facilitate the learning process.

Simulation results demonstrate the effectiveness of the proposed method, showing rapid learning and high performance with a near-perfect success rate of 0.99 within 3 million training steps. The approach shows improved sample efficiency compared to model-free methods and demonstrates robustness in complex scenarios such as S-curves, narrow U-turns, and double moose tests. In the next chapter, another MBRL approach will be introduced, where the agent need to learn the trajectory state by itself, and the imitation learning will be introduced to guide the agent to learn the optimal strategies more efficiently.

CHAPTER SEVEN

IMITATION INSPIRED PROBABILISTIC ENSEMBLES WITH TRAJECTORY SAMPLING FOR TTRV PARKING CONTROL

Model-based reinforcement learning algorithms such as Probabilistic Ensembles with Trajectory Sampling have shown good results in solving dynamic control problems. Previous chapter talked about the MBRL approach with a known environment model, Hybrid A* path planning, and the model-free methods can be used on the planner model to reduce trajectory state deviation. However, the model-free approach might not be flexible enough to handle the complex nature of all possible parking maneuvers. In this chapter, a MBRL approach with an unknown model is introduced, which is based on the PETS framework.

PETS learns dynamics model of the environment and uses this model to sampling trajectory states distributions, which will be used for predicting the control actions' performance. However, using PETS directly on TTRV parking can be challenging because there is not enough useful initial data, and parking maneuvers can be pretty complex because the parking tasks usually take a long episode length which requires the trajectory states sampling for long future, and at the same time, many possible states and parking trajectories will also bring extensive data to learn.

This chapter introduced the integration of imitation learning with PETS algorithm, and IIPETS is specifically suitable for ADAS task, because other than training with completely random collected dataset, IIPETS uses trajectories datasets collected from real environment, or an expert demonstration, and this gives a good starting point for learning the dynamics model and making the policy better.

7.1 Introduction to Trajectory Sampling Methods

In MBRL, trajectory sampling methods have attracted many researcher’s attention because of their ability to leverage the learned dynamics models to generate trajectory data for policy optimization. These model-based methods try to mitigate the sample inefficiency challenge in the traditional reinforcement learning algorithms by integrating real environment interactions with simulated data from the learned model. Through interleaving real and simulated transitions, trajectory sampling methods can accelerate policy learning and improve the algorithm sample efficiency.

One of the earliest model-based reinforcement learning approaches is the Dyna algorithm proposed by Sutton[86]. As shown in Figure 7.1, Dyna mix the real environment interactions with simulated steps from a learned dynamics model. The policy is updated using both real and simulated transitions, which accelerates learning. Formally, the Dyna update rule for the action-value function $Q(s, a)$ is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \quad (7.1)$$

where α is the learning rate, γ is the discount factor, r is the reward, and s' is the next state obtained either from the real environment or the learned model.

Although Dyna itself is not a trajectory sampling approach, it built the foundation for combining model-based planning with model-free updates. Recent research and industrial products have explored integrating trajectory sampling with modern model-free algorithms like Q-learning[87] and policy gradients[88]. For instance, the Stochastic Trajectory Estimation with Value Expansion (STEVE) algorithm[89] used trajectory sampling with deep ensembles to capture model uncertainty for risk-sensitive exploration.

Compared with Dyna, model-based trajectory sampling approaches explicitly generate and utilize complete trajectories from learned dynamics models. These methods

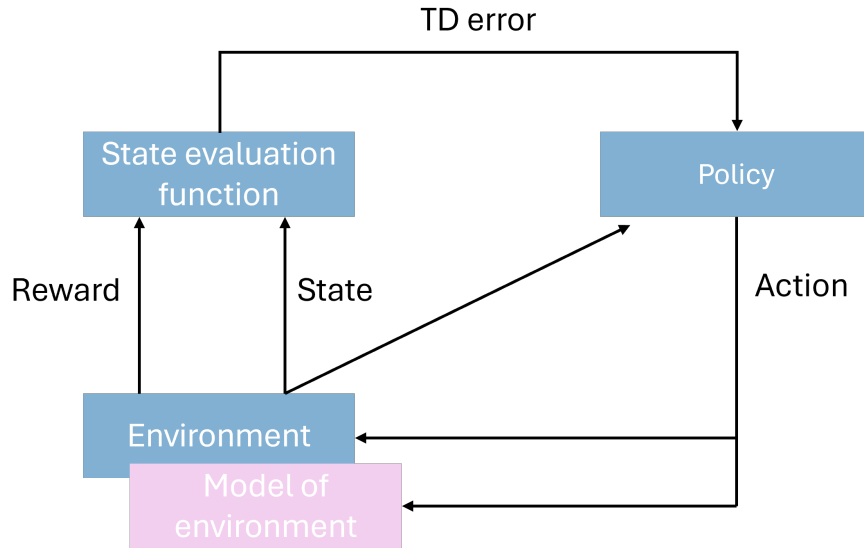


Figure 7.1: Dyna algorithm architecture.

sample entire sequences of states and actions, allowing for comprehensive policy updates that often integrate model-based planning with model-free reinforcement learning techniques. Trajectory sampling approaches aim to improve algorithm performance by reducing model bias through ensemble techniques or probabilistic models, while managing uncertainty in the dynamics model. By generating synthetic experiences, these methods enhance learning exploration capabilities and increase robustness, particularly in complex environments with high variability or limited real-world data. Recent advances in this research field, such as MBPO[83], continue to push the boundaries of what can be achieved with trajectory sampling and model-based methods in reinforcement learning, showing improvements in sample efficiency and performance across a wide range of tasks.

A key challenge in trajectory sampling methods is learning an accurate dynamics model that can generate high-quality synthetic data. Probabilistic ensembles of neural networks could be a promising solution for learning probabilistic dynamics models. Given

a state-action pair (s, a) , a probabilistic ensemble dynamics model outputs a distribution over the next state s' as[79, 88]:

$$p(s'|s, a) = \frac{1}{N} \sum_{i=1}^N \mathcal{N}(f_i(s, a), \sigma_i^2) \quad (7.2)$$

where f_i is the output of the i -th neural network in the ensemble, σ_i^2 is the corresponding variance, and N is the total number of networks in the ensemble. Each f_i is trained independently to predict the next state s' from the current state s and action a . This creates a Gaussian mixture model, where each network's output $f_i(s, a)$ output the mean, and σ_i^2 represents the associated variance. The ensemble model captures both epistemic uncertainty (due to limited data) and aleatoric uncertainty (inherent randomness in the environment), allowing different networks to specialize in different regions of the state-action space.

During trajectory sampling, future states are generated from this predictive distribution by repeatedly sampling $s' \sim p(s'|s, a)$ and then applying the selected action. The ensemble approach provides a principled way to represent predictive uncertainty, enabling the generation of diverse, probabilistic trajectories that help the agent explore effectively and make robust decisions.

One influential trajectory sampling algorithm that used probabilistic ensembles is PETS[79]. PETS is a model-based reinforcement learning algorithm that optimizes a control policy π to maximize the expected return while considering model uncertainty. The algorithm uses an ensemble of dynamics models to capture uncertainty and uses MPC for action selection:

$$\max_{\pi} \mathbb{E} \tau \sim p(\tau|\pi) \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] \quad (7.3)$$

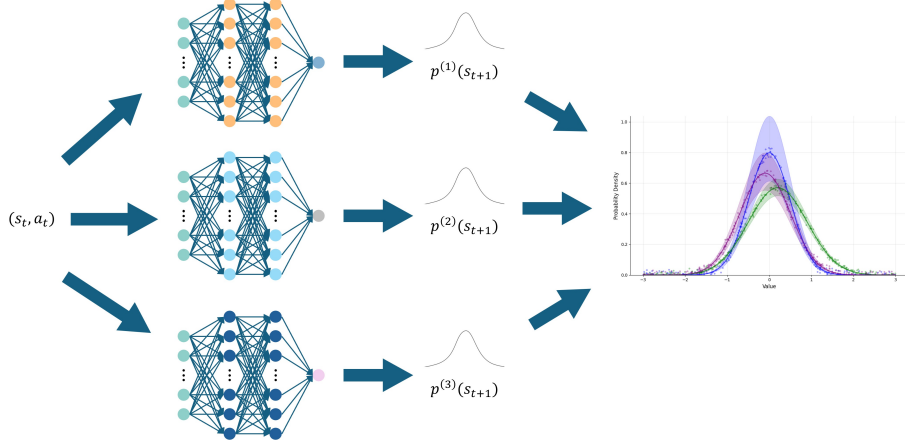


Figure 7.2: Probabilistic ensemble dynamics model for trajectory sampling.

Here, $p(\tau|\pi)$ is the trajectory distribution induced by the policy π and the learned dynamics model. γ is the discount factor, and $r(s_t, a_t)$ is the reward function.

The policy π is implicitly defined through an optimization process using sampled trajectories from the ensemble dynamics model. In each iteration, PETS samples a set of trajectories $\tau_{i=1}^M$ from $p(\tau|\pi)$ by first sampling an initial state s_0 from the environment, and then repeatedly sampling actions and next states.

The action selection process in PETS involves sampling K candidate action sequences $A_k, k = 1^K$, where $A_k = (a_0^k, \dots, a_{T-1}^k)$. For each action sequence, the state is propagated through the ensemble of dynamics models to obtain a distribution of trajectories. The algorithm then selects the action sequence that maximizes the expected return across the ensemble:

$$A^* = \arg \max_{A_k} \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T-1} r(s_t^{n,k}, a_t^k) \quad (7.4)$$

where N is the number of dynamics models in the ensemble, and $s_t^{n,k}$ is the state at time t predicted by the n -th model for the k -th action sequence. PETS then executes the first

action a_0 from the optimal action sequence A . This process is repeated at each timestep, implementing a form of model predictive control.

The dynamics models in the ensemble are trained to minimize the negative log-likelihood of the observed transitions:

$$\min_{\theta} - \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \log p_{\theta}(s_{t+1} | s_t, a_t) \quad (7.5)$$

where $\mathcal{D}$ is the dataset of observed transitions, and θ are the parameters of the dynamics model. By using an ensemble of models and considering their predictions during trajectory optimization, PETS implicitly accounts for model uncertainty without explicitly minimizing the variance of returns as originally stated.

Extensions of PETS have explored alternative risk measures[90], and hierarchical policies[91]. Various policy optimization algorithms such as TRPO, PPO, or SAC can be integrated within the PETS framework to potentially improve performance.

Another related approach is the Model-based Bellman Policy Optimization (MB-BPO) algorithm[88], which optimizes a trust-region constrained objective using trajectories sampled from an ensemble dynamics model:

$$\max_{\pi'} \mathbb{E}_{\tau \sim p(\tau | \pi')} \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] \quad \text{s.t.} \quad \max_{s, a} D_{\text{KL}}(\pi'(\cdot | s) || \pi(\cdot | s)) \leq \varepsilon \quad (7.6)$$

where ε is a trust-region constraint, and D_{KL} is the Kullback-Leibler divergence between the new policy π' and the old policy π .

7.2 Learning from Imitation Inspiration

One of the challenge in applying PETS directly to the TTRV parking task is the lack of informative initial data to accurately learn the dynamics model, especially given

the high-dimensional observation space. The simulation results showed that the purely random trajectories often fail to provide sufficient information about feasible parking maneuvers, which significantly slow down policy optimization.

Recent advancements in vehicle communication technology offer a potential solution to this data shortage and suboptimal problem. Modern vehicles now collect real-time environmental data and upload drivers' driving data to cloud servers. This technology provides an opportunity to integrate the real driving data into the training dataset, which provide data natural and informative than randomly generated trajectories. For example, Tesla's Autopilot system has successfully used vast amounts of driver-collected data to improve autonomous driving performance[92]. A similar approach can be applied to the TTRV parking task in this research, providing relevant data for more effective learning.

To address this challenge, this chapter propose integration of imitation learning to initialize both the dynamics model and the policy with a set of demonstrated parking trajectories, generated by a MPC controller designed for reverse parking. Let $\mathcal{D}_{\text{rand}}$ be the set of randomly collected trajectories, and $\mathcal{D}_{\text{real}}$ be the set of trajectories from real drivers. The initial dataset for learning the dynamics model and policy is then:

$$\mathcal{D}_{\text{init}} = \mathcal{D}_{\text{rand}} \cup \mathcal{D}_{\text{real}} \quad (7.7)$$

This imitation learning-inspired initialization provides a more informative starting point for the PETS algorithm by exposing the model to relevant parking behaviors early on. The subsequent trajectory sampling and policy optimization steps can refine and improve upon this initialization, leveraging the full power of the PETS framework:

$$\max_{\pi} \mathbb{E} \tau \sim p(\tau|\pi) \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] \quad (7.8)$$

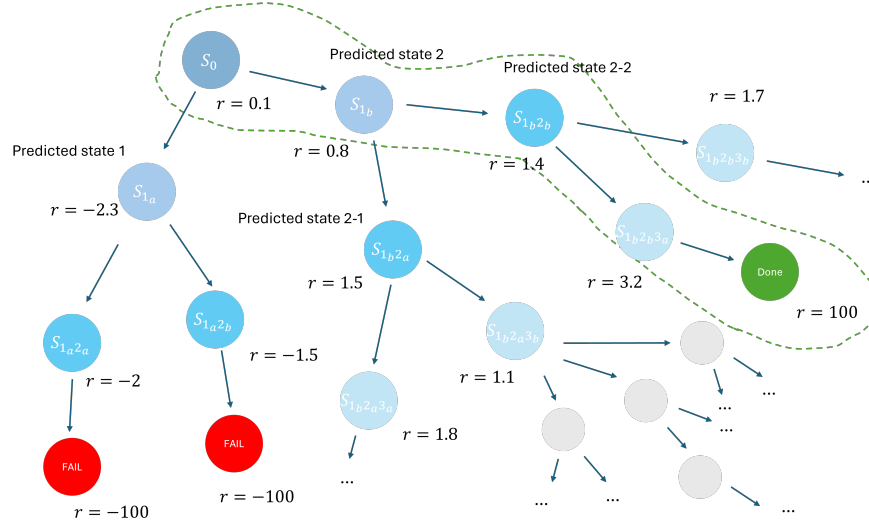


Figure 7.3: Illustration of the IIPETS trajectory prediction process.

where $p(\tau|\pi)$ is the trajectory distribution induced by the policy π and the learned dynamics model.

During trajectory sampling, PETS generates a set of trajectories $\{\tau_i\}_{i=1}^M$ from the current policy π and the learned dynamics model $p(\cdot|s, a)$:

$$\tau_i = \{s_0, a_0, s_1, a_1, \dots, s_T\}, \quad a_t \sim \pi(\cdot|s_t), \quad s_{t+1} \sim p(\cdot|s_t, a_t) \quad (7.9)$$

These sampled trajectories, combined with the initial dataset $\mathcal{D}_{\text{init}}$, are used to optimize the policy π by balancing expected return and risk:

$$\max_{\pi} \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] - \lambda \text{Var}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] \quad (7.10)$$

After each policy optimization step, the dynamics model is further refined using the updated policy and the expanded dataset $\mathcal{D}_{\text{combined}} = \mathcal{D}_{\text{init}} \cup \{\tau_i\}_{i=1}^M$:

$$\min_{\theta} \sum_{(s,a,s') \in \mathcal{D}_{\text{combined}}} \|s' - p(s'|s, a; \theta)\|_2^2 \quad (7.11)$$

where θ represents the parameters of the dynamics model.

This iterative process of trajectory sampling, policy optimization, and dynamics model refinement benefits from the inclusion of real driver data in the initial dataset. This data helps the dynamics model capture the nature of human driving behavior, which may not be fully represented in randomly collected trajectories or those generated by simple controllers. Moreover, the experience from multiple human drivers allows the model and policy to generalize better across a range of parking scenarios and environmental conditions.

The ratio of random trajectories to real driver data in $\mathcal{D}_{\text{init}}$ can be tuned as a hyperparameter, providing flexibility to balance between exploration through random data and guidance from human demonstrations. This is particularly useful when the real driver data may be biased or incomplete, necessitating further random exploration.

In summary, this imitation learning-inspired initialization offers a more relevant starting point for the PETS algorithm, improving learning efficiency and potentially enhancing performance in TTRV parking tasks. By combining the strengths of imitation learning and model-based reinforcement learning, this chapter aims to address the unique challenges posed by complex vehicle control tasks, pushing the boundaries of autonomous parking systems.

7.3 Implementation and Experimental Setup

The implementation of IIPETS involves an integration of imitation learning with the PETS framework, designed to address the non-holonomic challenges of TTRV reverse

drivnig. The approach begins with a comprehensive data collection process to initialize the agent’s knowledge database, followed by the learning procedure.

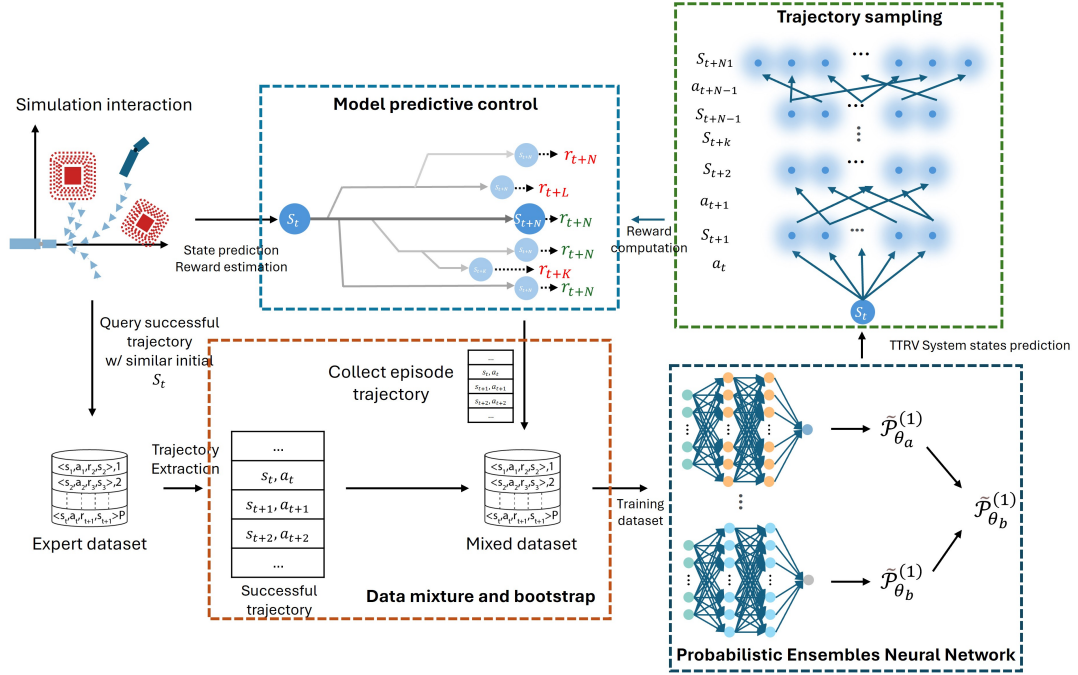


Figure 7.4: Overview of the IIPETS algorithm workflow.

To facilitate the learning speed, a modified potential field reward is implemented. Rather than using the euclidean distance, the distance to goal is computed by adding the distance from trailer to the closest point on the Kinodynamic RRT* generated path, and the curvy distance from the point on the trajectory to goal along the trajectory. Also, to achieve obstacle avoidance, a sigmoid function-based reward system to maintain a safe distance between the trailer and obstacles. The reward function is designed as follows:

$$R(d) = \begin{cases} 0 & \text{if } d \geq d_{\text{safe}} \\ -\frac{1}{1+e^{k(d-d_{\text{threshold}})}} & \text{if } d < d_{\text{safe}} \end{cases} \quad (7.12)$$

where d is the distance to the nearest obstacle, d_{safe} is the safe distance threshold, $d_{\text{threshold}}$ is the distance at which the penalty starts to increase rapidly, and k is a parameter controlling the steepness of the sigmoid function.

This reward structure encourages the agent to maintain a safe distance from obstacles while allowing for necessary close maneuvers in tight parking scenarios. The Figure 7.5 shows how the reward/penalty varies with the distance to the nearest obstacle. As the distance increases, the penalty decreases sigmoidally, approaching zero for distances beyond 6 units. The steep decline in penalty between distances 1 and 4 provides a strong gradient for learning, while the flattening beyond 6 units indicates a safe zone with minimal penalty.

Building from the imitation learning inspiration discussed earlier, the proposed IIPETS learning process bridges the gap between the theoretical foundations of imitation-inspired learning and the practical implementation in the context of TTRV parking. By using the mocked diverse dataset, this chapter aims to help the agent learn a robust initial understanding of the task dynamics.

The dataset needs to balance between different trajectory sources: 50% of the data comes from Kinodynamic RRT* planned paths, providing optimal paths that respect the vehicle's kinodynamic constraints, and the MPC controller is used to generate trajectory states. The remaining 50% consists of random exploration trajectories from different initial states, which help to cover a broader range of possible states and actions. This mix of optimal, human-like, and random trajectories provides a balanced initial dataset that captures various aspects of the parking task.

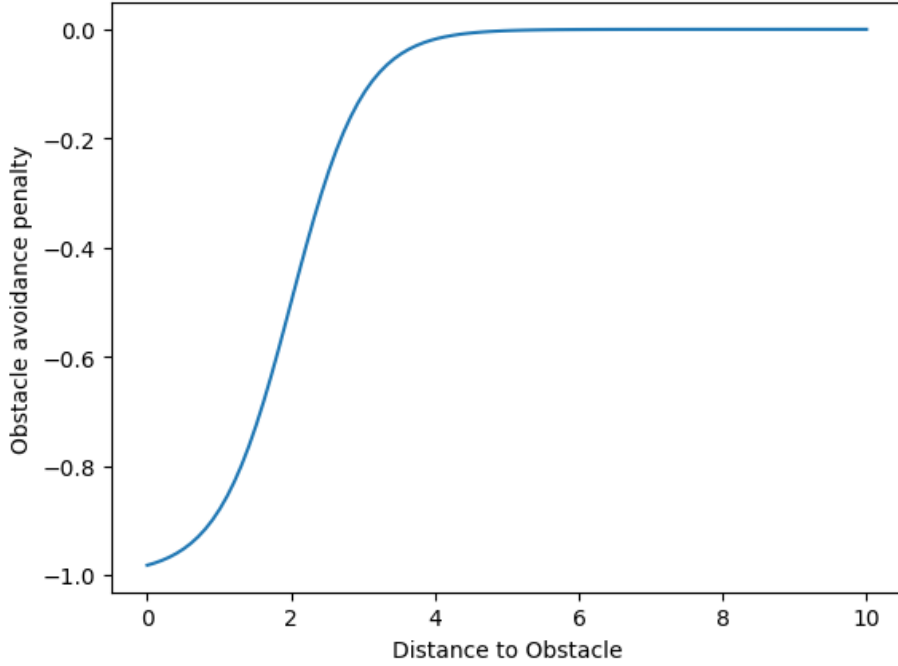


Figure 7.5: Obstacle avoidance penalty function.

All collected trajectories are stored in a replay buffer $\mathcal{D} = \{(s_t, a_t, s_{t+1}, r_t)\}_{t=1}^T$, where r_t is the reward computed based on the state transition. This diverse composition ensures that the agent is exposed to a wide variety of parking scenarios and strategies, from theoretically optimal to more exploratory approaches, thereby fostering a more robust and adaptable learning process.

The IIPETS agent undergoes an initial learning phase using the replay buffer. During this phase, the agent loads the trajectory states and actions, predicts the next states and associated rewards, and adds these predictions to the replay buffer. This process allows the agent to learn from demonstrated behaviors before interacting with the environment directly.

The learning process consists of two main components: dynamics model learning and policy optimization. For the dynamics model, this research employs an ensemble of K

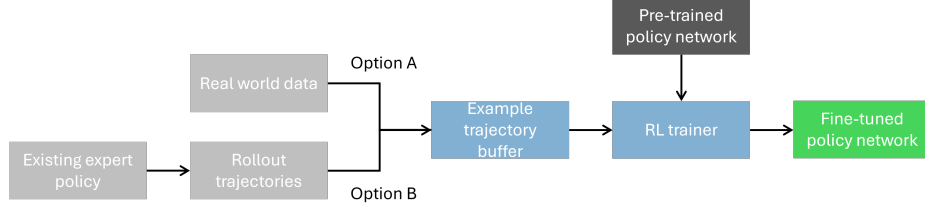


Figure 7.6: Overview of the IIPETS algorithm workflow.

neural networks $\{f_{\theta_k}\}_{k=1}^K$, each parameterized by θ_k . The dynamics model is trained to minimize the mean squared error:

$$\mathcal{L}_{\text{dynamics}} = \frac{1}{|\mathcal{D}|} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \frac{1}{K} \sum_{k=1}^K \|f_{\theta_k}(s_t, a_t) - s_{t+1}\|_2^2 \quad (7.13)$$

The policy π_ϕ , parameterized by ϕ , is optimized using the PETS objective:

$$\mathcal{J}(\phi) = \mathbb{E}_{\tau \sim p(\tau|\pi_\phi)} \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] - \lambda \text{Var}_{\tau \sim p(\tau|\pi_\phi)} \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right] \quad (7.14)$$

where γ is the discount factor and λ is the risk sensitivity parameter.

7.4 Results and Discussion

The performance of the IIPETS algorithm for TTRV reverse parking is evaluated through a series of experiments in environment with and without obstacles. First stage of the simulation is conducted in the environment only bounded with boundaries, and the TTRV can navigate freely to the goal as long as it is within the boundary. Figure 7.8, Figure 7.9, Figure 7.10, and Figure 7.11 shows key evaluation metrics of the algorithm's performance over the training.



Figure 7.7: Stacked screenshots of the IIPETS algorithm in an obstacle-free environment.

As shown in Figure 7.7, the IIPETS algorithm successfully learns to navigate the truck and trailer to the parking goal in an obstacle-free environment. The agent maneuvers in a smooth and efficient parking strategy, proving its capability to learn complex vehicle dynamics and achieve precise vehicle statecontrol. The algorithm's performance is further analyzed through key evaluation metrics, providing insights into its learning progress and adaptability.

Figure 7.8 shows the evolution of rewards per trial. The original rewards (blue line) exhibit high volatility, with sudden drops indicating unsuccessful parking attempts. In contrast, the smoothed rewards (red line) demonstrate a clear upward trend, suggesting that the algorithm learns to consistently achieve higher rewards over time. This improvement indicates that IIPETS effectively learns to perform the parking task more reliably as training progresses.

The validation scores, depicted in Figure 7.9, show periodic spikes followed by rapid declines. This pattern suggests that the algorithm occasionally explores new



Figure 7.8: Trial rewards: original vs smoothed.

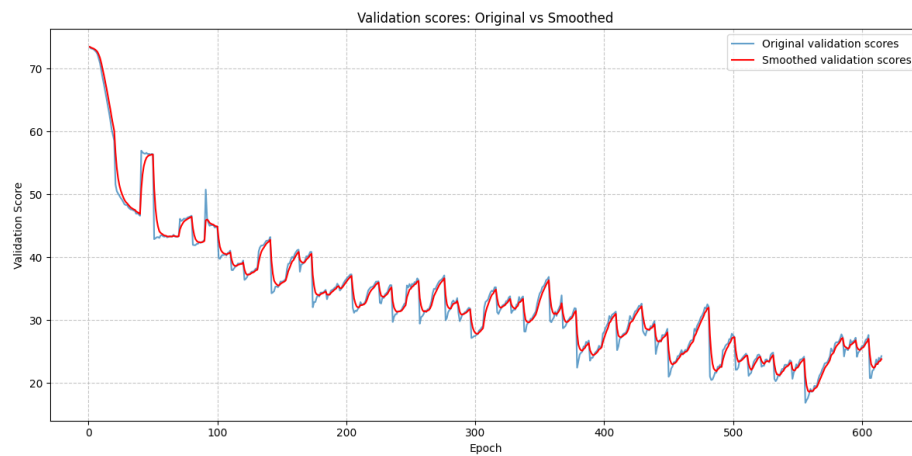


Figure 7.9: Validation scores: original vs smoothed.

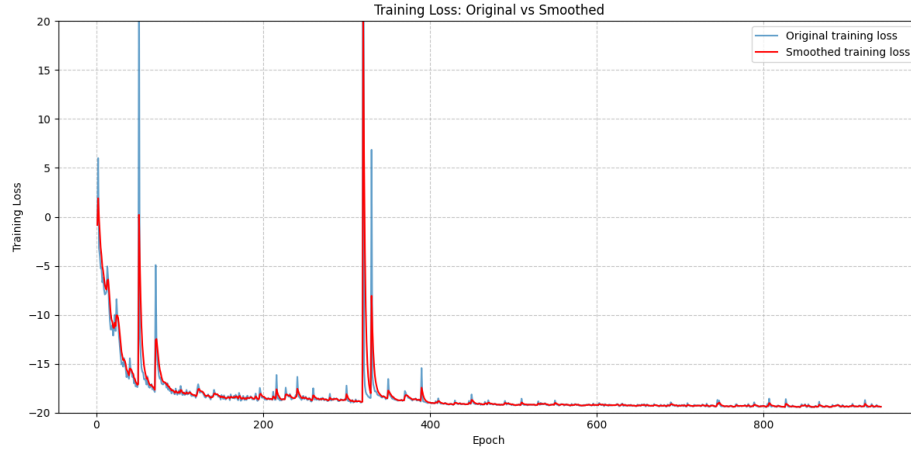


Figure 7.10: Training loss: original vs smoothed.

strategies, leading to temporary increases in validation error, before quickly adapting and improving its performance.

Figure 7.10 illustrates the training loss over epochs. The plot shows an initial period of high volatility, followed by a gradual stabilization. This behavior is typical of deep learning models and indicates that the neural networks within IIPETS are successfully learning to model the dynamics of the parking task.

The success rate of parking episodes, shown in Figure 7.11, provides perhaps the clearest indication of IIPETS's performance. The smoothed success rate (red line) shows a rapid initial improvement, followed by consistent high performance with occasional dips. By the end of training, the algorithm achieves near-perfect success rates, demonstrating its ability to reliably park the truck and trailer.

Overall, these results indicate that IIPETS successfully learns to perform truck trailer reverse parking in an obstacle-free environment. The algorithm shows clear improvements in reward, reduction in training loss, and high success rates, suggesting effective learning of the task dynamics and policy optimization. To further evaluate the

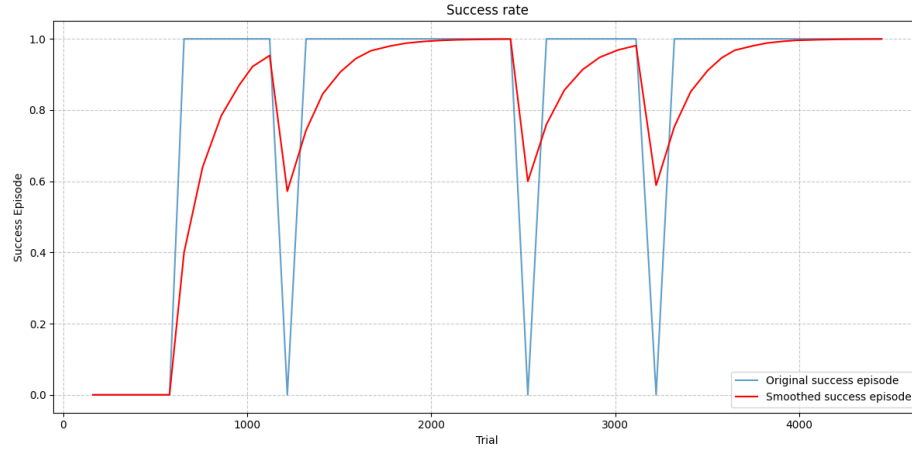


Figure 7.11: Success episode: original vs smoothed.

robustness and adaptability of IIPETS, this research extended the experiments to include scenarios with obstacles, simulating a vehicle moose test-style environment.

The proposed IIPETS algorithm is also tested in a moose test-style environment with obstacles. As shown in Figure 7.12, the agent successfully navigates through the obstacles to reach the parking goal, which demonstrates the ability to adapt to complex and dynamic environments. Figure 7.13 and Figure 7.14 illustrate the performance of IIPETS in this more complex setting. As expected, the presence of obstacles introduces additional challenges, resulting in a slower and more volatile learning process.

Figure 7.13 shows the evolution of rewards per trial in the obstacle-rich environment. Compared to the obstacle-free scenario, more pronounced fluctuations in both the original and smoothed reward curves are observed. The learning process exhibits a less steep upward trend, indicating the increased difficulty of the task. Despite these challenges, the smoothed rewards (red line) still demonstrate a gradual improvement over time, suggesting that IIPETS can adapt to the more complex environment, albeit at a slower pace.

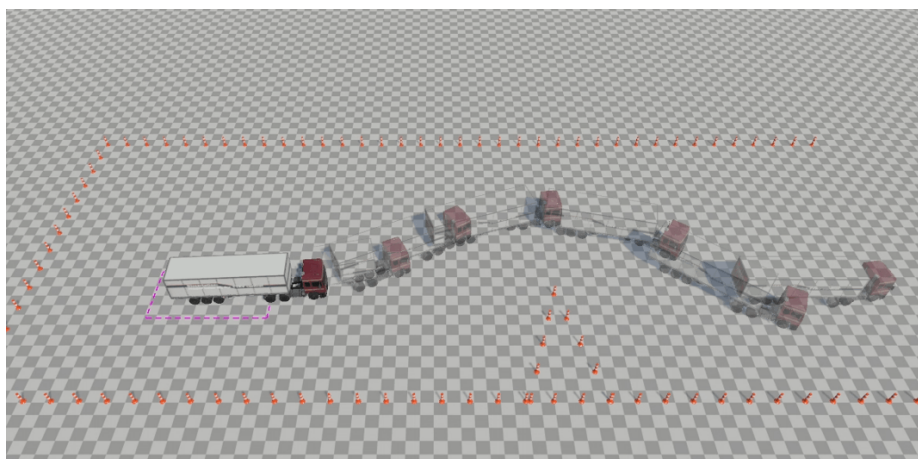


Figure 7.12: Stacked screenshots of the IIPETS algorithm in an moost test environment.

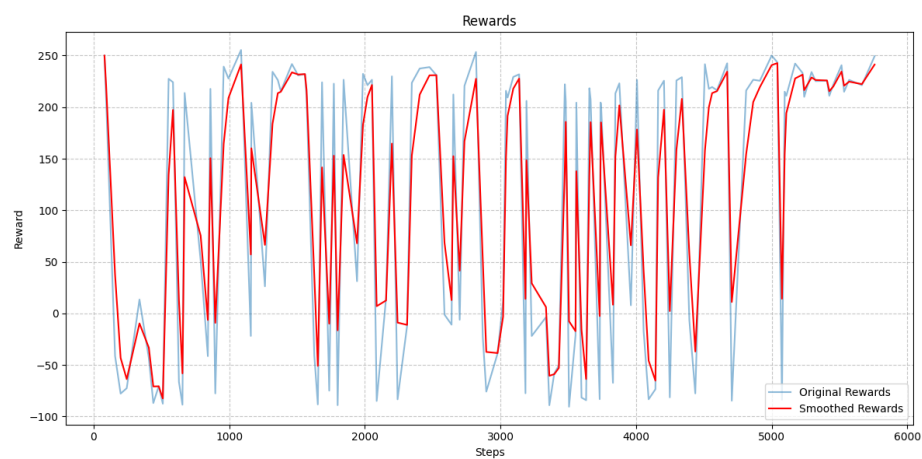


Figure 7.13: Trial rewards with obstacles: original vs smoothed.

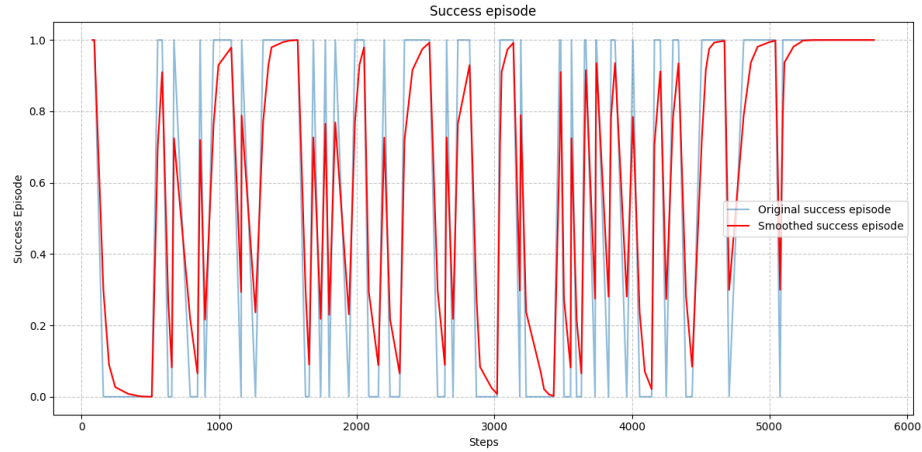


Figure 7.14: Success episode with obstacles: original vs smoothed.

Figure 7.14 depicts the success rate of parking episodes in the obstacle-avoidance scenario. The graph reveals a more erratic learning pattern compared to the obstacle-free environment. The original success rate (blue line) shows frequent fluctuations between complete success and failure, highlighting the challenging nature of the task. The smoothed success rate (red line) provides a clearer picture of the overall trend, showing a gradual improvement over time. However, the learning curve is less steep and takes longer to reach high success rates, further emphasizing the increased complexity of the obstacle-avoidance task.

These results demonstrate that while IIPETS can adapt to more complex environments with obstacles, the learning process becomes more challenging and time-consuming. The algorithm's ability to improve performance in this scenario underscores its potential for real-world applications where obstacle avoidance is crucial. Future work could focus on enhancing IIPETS's efficiency in complex environments and exploring strategies to accelerate learning in obstacle-rich scenarios.

7.5 Chapter Conclusion

This chapter introduced the IIPETS algorithm for TTRV reverse parking control, combining imitation learning with the PETS framework. IIPETS demonstrated improved sample efficiency and simulation performance compared to model-free RL approaches such as PPO and TD3, and comparable performance as the benchmark methods like MPC and LQR. The algorithm successfully learned parking strategy in both obstacle-free and obstacle-rich environments, showing consistent improvement in rewards, success rates, and training stability. These results highlight IIPETS's potential for real-world autonomous parking applications, especially in complex obstacle avoidance tasks and with the dataset collected from real drivers. Future research should focus on improving efficiency in complex environments, initializing the Gaussian model parameters (mean and variance) based on real driver data or physical constraints to ensure realistic state distributions, and comparing its simulation outputs with state-of-the-art control methods.

CHAPTER EIGHT

CONCLUSION AND FUTURE WORK

This chapter concludes the dissertation by summarizing the key findings and contributions in the field of autonomous control for TTRV systems, with a focus on reverse parking. It summarizes the progression from classical control methods to advanced reinforcement learning techniques, highlighting the novel approaches developed throughout this research. The chapter reflects on the significance of the trajectory state model-based RL controller and the IIPETS algorithm, which allow fast and robust control for TTRV. Additionally, it outlines potential future research directions, emphasizing the current challenges and research opportunities for autonomous vehicle technology development.

8.1 Conclusions

This dissertation has explored the development history and state of art application of autonomous control for TTRV, particularly focusing on reverse parking. This research began with a comprehensive analysis of TTRV kinematics and physical constraints, which provides the foundation for understanding the complexities of truck trailer control problem. The study then explores the current popular path planning and control algorithm, from classical approaches which has been widely used for modern vehicles, to the emerging technology such as reinforcement learning algorithms.

The classical methods, including Dubins paths, Reeds-Shepp curves, Hybrid A*, and Kinodynamic RRT*, are tested in several simultion parking scnerios and demonstrated their strengths in such well-defined environment, but these methods showed

limitations in adapting to complex, dynamic scenarios from real world application from the industry feedback. The exploration of traditional controllers such as pole placement, Lyapunov-based methods, LQR, and MPC proved their efficiency in controlling the vehicle trajectory following of planned path. These algorithms provide a comparison benchmark for the intelligent control methods mentioned in the rest of this dissertation.

This dissertation then switches to intelligent control algorithms, especially reinforcement learning, which has shown great potential in solving complex control problems in gaming and robotics. The model-free approaches, particularly the implementations of PPO and modified TD3 algorithms, demonstrated remarkable performance and adaptability in diverse parking scenarios. These methods effectively learned control policies directly from experience, overcoming many limitations of classical approaches.

In the last part of this dissertation, model-based reinforcement learning approaches are introduced and implemented. The first model is a hybrid method that combines trajectory planning module with model-free reinforcement learning methods, which showed a significant improvement in learning efficiency and performance. The second model is the IIPETS algorithm, which integrates imitation learning principles and probabilistic ensembles to achieve high sample efficiency in complex parking tasks.

Throughout the study, there was a clear improvement in the ability to handle different environments, and perform well in various parking scenarios, in both obstacle free and obstacle rich situations. The trajectory model based approach, as well as the IIPETS algorithm, in particular, represents a big step forward in controlling TTRVs maneuvering reversely, and it provides a promising and efficient solution to reverse parking problem.

8.2 Contributions

Several contributions has been made to ADAS system, specifically for truck-trailer systems. The work on TTRV kinematic modeling and its system constrain analysis provides a comprehensive framework for understanding the nature of this system, which is a valuable resource for future research and development for truck controller.

The analysis of classical trajectory planning and control methods as well as the comparison between different algorithms offer insights to the the pros and cons of these traditional approaches. This simulation evaluation showed the capability of these classical methods, and provided a comaprison baselien for future intelligent control algorithms.

The application of model-free deep reinforcement learning algorithms to TTRV control started the discussion of modern intelligent controller in the robotics. The implementations of PPO and the modified TD3 algorithms, which is modified specifically for the reverse parking task, demonstrate the potential of these approaches in handling complex, non-linear control problems. Also, the simulation environment and the 3D visualization and the evaluation metrics developed in this study provide a standardized platform testing and comparing different control algorithms for future researchers in this field.

The biggest contribution of this work is the development of hybrid and imitation-inspired model-based approaches. The hybrid model-based method successfully integrates classical planning techniques with modern learning algorithms, offering a balanced approach that leverages the strengths of both paradigms. This hybrid method shows how traditional robotics knowledge can be effectively combined with data-driven learning to create more robust and efficient control systems. The IIPETS algorithm also contribute significantly to the field of autonomous vehicle control for its sample efficiency and generalization capability by using imitation learning and probabilistic ensembles,

which is a promising research direction for the future, as computation resources and data collection is becoming popular and more accessible.

These contributions all together advance the state-of-the-art in TTRV intelligent control, offering new methodologies and insights that bridge the gap between theoretical models and practical implementations. The opensource code and simulation environment developed in this study not only addresses the specific challenges of TTRV reverse parking but also provides a foundation for solving similar complex control tasks in robotics and autonomous systems.

8.3 Future Work

While this dissertation has made significant strides in TTRV control, there are also several exciting open topics for future research. One direction is the improvement of IIPETS algorithm to more complex environments, including dynamic obstacles. Due to the computation power and data collection limitation, the current implementation of IIPETS is limited to certain defined simulation environment, and the algorithm could be further developed to handle more complex scenarios with vehicle dynamic model considered and real-world data collected.

Integrating advanced sensor technologies for improved state estimation and obstacle detection presents another potential improvement work. Nowadays, companies like Tesla and Xiaomi are developing pure vision based autonomous driving systems, which reduced the cost and complexity of the sensor system. However, the current simulation environment in this study are based on rasterized detection, different from the vision based system's detection logic. As a result, how to integrate the vision based detection into the current simulation environment and the control algorithm is a promising direction for both academic and industry researchers.

Exploring transfer learning techniques could significantly enhance the adaptability of the controllers. As data collection has become increasingly popular and the amount of data is growing, benefiting from the development of modern sensors and V2X communication, future research could focus on developing methods to efficiently adapt pre-trained algorithms to different vehicle configurations or parking scenarios without extensive retraining. Such model tailoring could happen on each customer's vehicle, allowing the ADAS system to be more personalized and efficient. This approach would greatly increase the practical applicability of these systems across various vehicle types and environments.

Through continued research in these directions, the development of ADAS will drive towards safer, more efficient, and truly autonomous transportation systems. The journey from theoretical models to practical, widely deployed autonomous vehicles is complex, but the potential benefits to society in terms of safety, efficiency, and accessibility make it a promising and exciting field of study.

REFERENCES

- [1] O. Ljungqvist, *Motion planning and feedback control techniques with applications to long tractor-trailer vehicles*, vol. 2070. Linköping University Electronic Press, 2020.
- [2] B. Jujnovich and D. Cebon, “Comparative performance of semi-trailer steering systems,” in *Proceedings of the 7th International Symposium on Heavy Vehicle Weights and Dimensions*, pp. 195–214, Delft, 2002.
- [3] D. Sperry, A. M. Schuster, S. R. Cotten, S. Agrawal, E. Mack, N. Britt, and J. Liberman, “Trucking in the era of covid-19,” *American Behavioral Scientist*, p. 00027642211066039, 2022.
- [4] M. Galvani, “History and future of driver assistance,” *IEEE Instrumentation & Measurement Magazine*, vol. 22, no. 1, pp. 11–16, 2019.
- [5] J. B. Cicchino, “Effectiveness of forward collision warning and autonomous emergency braking systems in reducing front-to-rear crash rates,” *Accident Analysis & Prevention*, vol. 99, pp. 142–152, 2017.
- [6] S. Cafiso and A. Di Graziano, “Evaluation of the effectiveness of adas in reducing multi-vehicle collisions,” *International journal of heavy vehicle systems*, vol. 19, no. 2, pp. 188–206, 2012.
- [7] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.
- [8] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.
- [9] J. Staddon, “The dynamics of behavior: Review of sutton and barto: Reinforcement learning: An introduction .,” *Journal of the Experimental Analysis of Behavior*, vol. 113, no. 2, 2020.
- [10] J. Peters, S. Vijayakumar, and S. Schaal, “Reinforcement learning for humanoid robotics,” in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp. 1–20, 2003.
- [11] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz, *et al.*, “Self-driving cars: A survey,” *Expert systems with applications*, vol. 165, p. 113816, 2021.

- [12] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.
- [13] E. Thorndike, *Animal intelligence: Experimental studies*. Routledge, 2017.
- [14] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine learning*, vol. 3, pp. 9–44, 1988.
- [15] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.
- [16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [17] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.
- [18] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, “Kinematic and dynamic vehicle models for autonomous driving control design,” in *2015 IEEE intelligent vehicles symposium (IV)*, pp. 1094–1099, IEEE, 2015.
- [19] P. Ta, Y. Lin, and C. Ta, “Implementation of trajectory tracking control for wheeled mobile robots based on fuzzy controller,” *Journal of China University of Science and Technology*, vol. 41, pp. 45–65, 2009.
- [20] A. M. Bloch and A. M. Bloch, *Nonholonomic mechanics*. Springer, 2015.
- [21] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.
- [22] J.-P. Laumond *et al.*, *Robot motion planning and control*, vol. 229. Springer, 1998.
- [23] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.
- [24] N. A. Fufaev, *Dynamics of nonholonomic systems*. American Mathematical Society, 1972.
- [25] F. Bullo and A. D. Lewis, *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*, vol. 49. Springer, 2019.
- [26] F. Sorge, “Motion and force analysis of slow-speed multi-trailer systems,” *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 233, no. 8, pp. 2615–2635, 2019.

- [27] X. Liu, A. K. Madhusudhanan, and D. Cebon, “Minimum swept-path control for autonomous reversing of a tractor semi-trailer,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4367–4376, 2019.
- [28] G. Lei and Y. Zheng, “Research on cooperative trajectory planning algorithm based on tractor-trailer wheeled robot,” *IEEE Access*, vol. 10, pp. 64209–64221, 2021.
- [29] A. Jameson, “Solution of the equation $ax+xb=c$ by inversion of an $m \times m$ or $n \times n$ matrix,” *SIAM Journal on Applied Mathematics*, vol. 16, no. 5, pp. 1020–1023, 1968.
- [30] P. Polack, F. Althé, B. d’Andréa Novel, and A. de La Fortelle, “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?,” in *2017 IEEE intelligent vehicles symposium (IV)*, pp. 812–818, IEEE, 2017.
- [31] S. Abraham and J. Nilsson, “Trailer parking assist,” *Semantic Scholar*, 2013.
- [32] R. W. Brockett, *Finite dimensional linear systems*. SIAM, 2015.
- [33] S. L. Brunton and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.
- [34] C. Altafini, A. Speranzon, and B. Wahlberg, “A feedback control scheme for reversing a truck and trailer vehicle,” *IEEE Transactions on robotics and automation*, vol. 17, no. 6, pp. 915–922, 2001.
- [35] C. Altafini, “Controllability and singularities in the n -trailer system with kingpin hitching,” *IFAC Proceedings Volumes*, vol. 32, no. 2, pp. 2169–2174, 1999.
- [36] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
- [37] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.
- [38] 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, 2015.
- [39] W. Schwarting, J. Alonso-Mora, and D. Rus, “Planning and decision-making for autonomous vehicles,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 187–210, 2018.
- [40] M. W. Otte, “A survey of machine learning approaches to robotic path-planning,” *University of Colorado at Boulder*, 2015.

- [41] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.
- [42] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.
- [43] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.
- [44] D. J. Webb and J. Van Den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE international conference on robotics and automation*, pp. 5054–5061, IEEE, 2013.
- [45] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
- [46] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," *Advances in neural information processing systems*, vol. 28, 2015.
- [47] G. F. Franklin, J. D. Powell, A. Emami-Naeini, and J. D. Powell, *Feedback control of dynamic systems*, vol. 4. Prentice hall Upper Saddle River, 2002.
- [48] H. K. Khalil, "Lyapunov stability," *Control systems, robotics and automation*, vol. 12, p. 115, 2009.
- [49] H. Khalil, *Nonlinear Systems*. Pearson Education, Prentice Hall, 2002.
- [50] N. Lehtomaki, N. Sandell, and M. Athans, "Robustness results in linear-quadratic gaussian based multivariable control designs," *IEEE Transactions on Automatic Control*, vol. 26, no. 1, pp. 75–93, 1981.
- [51] H. Yan, F. Yang, and Z. Wang, "Model predictive control for unmanned tracked vehicle path following," in *2018 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC)*, pp. 101–106, IEEE, 2018.
- [52] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.
- [53] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [54] G. S. Sebestyen, *Decision-making processes in pattern recognition (ACM monograph series)*. Macmillan Publishing Co., Inc., 1962.

- [55] J. R. Quinlan, “Induction of decision trees,” *Machine learning*, vol. 1, pp. 81–106, 1986.
- [56] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [57] R. E. Schapire, “The strength of weak learnability,” *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990.
- [58] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.
- [59] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
- [62] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
- [63] R. Bellman, “A markovian decision process,” *Journal of mathematics and mechanics*, pp. 679–684, 1957.
- [64] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.
- [65] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [66] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [67] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [68] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

- [69] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE international conference on robotics and automation*, vol. 2, pp. 500–505, IEEE, 1985.
- [70] S. S. Ge and Y. J. Cui, “New potential functions for mobile robot path planning,” *IEEE Transactions on robotics and automation*, vol. 16, no. 5, pp. 615–620, 2000.
- [71] J. Borenstein, Y. Koren, *et al.*, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE transactions on robotics and automation*, vol. 7, no. 3, pp. 278–288, 1991.
- [72] C. I. Connolly, J. B. Burns, and R. Weiss, “Path planning using laplace’s equation,” in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 2102–2106, IEEE, 1990.
- [73] E. Rimon, *Exact robot navigation using artificial potential functions*. Yale University, 1990.
- [74] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, “Uav path planning using artificial potential field method updated by optimal control theory,” *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016.
- [75] H. Xu, Z. Yan, J. Xuan, G. Zhang, and J. Lu, “Improving proximal policy optimization with alpha divergence,” *Neurocomputing*, vol. 534, pp. 94–105, 2023.
- [76] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.
- [77] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.
- [78] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, *et al.*, “Model-based reinforcement learning: A survey,” *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.
- [79] K. Chua, R. Calandra, R. McAllister, and S. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” *Advances in neural information processing systems*, vol. 31, 2018.
- [80] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [81] H. Yan, M. A. Zohdy, E. Alhawsawi, and A. Mahmoud, “Trajectory state model-based reinforcement learning for truck-trailer reverse driving,” in *2024 8th*

International Conference on Robotics, Control and Automation (ICRCA), pp. 197–205, 2024.

- [82] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, “Benchmarking model-based reinforcement learning,” *arXiv preprint arXiv:1907.02057*, 2019.
- [83] M. Janner, J. Fu, M. Zhang, and S. Levine, “When to trust your model: Model-based policy optimization,” *arXiv preprint arXiv:1906.08253*, 2019.
- [84] M. Deisenroth and C. E. Rasmussen, “Pilco: A model-based and data-efficient approach to policy search,” in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.
- [85] F. M. Barbosa, L. B. Marcos, M. M. da Silva, M. H. Terra, and V. G. Junior, “Robust path-following control for articulated heavy-duty vehicles,” *Control Engineering Practice*, vol. 85, pp. 246–256, 2019.
- [86] R. S. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.
- [87] G. Kalweit and J. Boedecker, “Uncertainty-driven imagination for continuous deep reinforcement learning,” in *Conference on robot learning*, pp. 195–206, PMLR, 2017.
- [88] A. Santara, A. Naik, B. Ravindran, D. Das, D. Mudigere, S. Avancha, and B. Kaul, “Rail: Risk-averse imitation learning,” *arXiv preprint arXiv:1707.06658*, 2017.
- [89] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, “Sample-efficient reinforcement learning with stochastic ensemble value expansion,” *Advances in neural information processing systems*, vol. 31, 2018.
- [90] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, “Risk-constrained reinforcement learning with percentile risk criteria,” *Journal of Machine Learning Research*, vol. 18, no. 167, pp. 1–51, 2018.
- [91] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg, “Dynamics randomization revisited: A case study for quadrupedal locomotion,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4955–4961, IEEE, 2021.
- [92] D. Kumari and S. Bhat, “Application of artificial intelligence technology in tesla-a case study,” *International Journal of Applied Engineering and Management Letters (IJAEML)*, vol. 5, no. 2, pp. 205–218, 2021.