Navigating Unsignalized Intersections: Deep RL-Based Decision-Making and Control Framework for Autonomous Vehicles with Pedestrian Integration

by

Faizan Sana

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis consists in part of one manuscript written for publication.

Research presented in Chapters 1 and 2:

In collaboration with *Dr. Nasser Lashgarian Azad* and *Dr. Kaamran Raahemifar*, I led the investigation presented in these chapters, which has been published in [1]. My role encompassed designing the study, conducting the literature survey, and drafting and revising the manuscript. I received valuable guidance and feedback from my co-authors throughout this process.

Abstract

Unprotected left turns at unsignalized intersections, alongside pedestrians and adversarial vehicles, pose significant challenges for Autonomous Vehicle (AV)s. These challenges stem from the absence of traffic signals or signs, the dynamic nature of the environment shaped by human interactions at crosswalks, and the variability in intersection layouts. This thesis delves into addressing these challenges through the application of a hierarchical Deep Reinforcement Learning (DRL) approach, where the DRL policy governs high-level decision-making (or planning), and low-level Proportional-Integral-Derivative (PID) controllers handle actuation.

To evaluate and train DRL policies, it was necessary to create a simulation environment within a high-fidelity environment with realistic behaviors and dynamic vehicle models. To the best of our knowledge, this research marks a pioneering effort in simulating pedestrian interactions within a high-fidelity environment, coexisting alongside adversarial vehicles within the CARLA simulation platform. We have dedicated extensive efforts to the development of this simulation, enabling straightforward customization of parameters such as the number of pedestrians, adversarial vehicles, and reward functions amongst others. This is made available open-source at `https://github.com/faizansana/intersection-carla-gym`.

The study evaluates five distinct model-free DRL algorithms, namely Deep Q-Learning (DQN), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Recurrent PPO, and Soft Actor Critic (SAC). The primary focus of this work is to conduct a comprehensive comparative analysis of these DRL algorithms within a hierarchical framework to enhance AV decision-making in complex and uncontrolled intersection scenarios. The training code, with its versatile software architecture is made available at `https://github.com/faizansana/intersection-driving`.

Our findings reveal that Recurrent PPO, coupled with a discretized action space, outperforms the other algorithms, displaying the highest success rate and the lowest accident rate for executing unprotected left turns in chaotic intersections. This outcome underscores the potential of Recurrent PPO to navigate such complex traffic scenarios effectively.

The thesis concludes by discussing potential extensions of the proposed hierarchical DRL system and outlining promising avenues for future research in the field of autonomous vehicle navigation at challenging and dynamic intersections.

Acknowledgements

In the name of Allah, the Most Gracious, the Most Merciful.

I would like to express my sincere gratitude to my supervisors *Professor Nasser Lashgarian Azad* and *Professor Kaamran Raahemifar* for their invaluable support, guidance, and encouragement throughout my masters journey. Their expertise, patience, and continuous inspiration have been instrumental in shaping this research and my personal growth as a student.

Special thanks to *Professor Siby Samuel* and *Professor Gennaro Notomista* for their dedicated reading and constructive feedback on my thesis. Your insights and suggestions have greatly contributed to the refinement of this work.

I extend my heartfelt appreciation to my parents and family members for their unwavering love, prayers, and unwavering belief in my abilities. Their constant encouragement and sacrifices have been a source of strength and motivation, pushing me to strive for excellence.

I also wish to extend my thanks to my esteemed teachers and mentors who have played a significant role in shaping my knowledge and intellectual curiosity. Your teachings and mentorship have been instrumental in my academic and personal development.

I am indebted to my friends and colleagues in the Automation and Intelligent Systems (AIS) group for their companionship, stimulating discussions, and support during challenging times. Your camaraderie has made this journey more meaningful and enjoyable. Equally, my heartfelt gratitude goes to the friends who have joined me on this path, enriching it with shared experiences and cherished memories.

I am grateful to the *Department of Systems Design Engineering* at the *University of Waterloo* for providing me with the necessary resources, facilities, and opportunities to pursue my studies and conduct this research.

To all those who have contributed directly or indirectly, your support and encouragement have been invaluable, and I am deeply thankful for your presence in my life.

In conclusion, I offer my profound thanks to everyone who has been part of my journey, from the bottom of my heart. May our paths continue to intersect and our bonds grow stronger with time.

Dedication

To my parents and family, whose love and unwavering support have been my constant motivation, this work is dedicated with heartfelt gratitude.

Table of Contents

Author's Declaration ii

Statement of Contributions iii

Abstract iv

Acknowledgements v

Dedication vi

List of Figures xi

List of Tables xiii

List of Abbreviations xiv

List of Symbols xvii

1 Introduction 1

1.1 Motivation 1

1.2 Levels of Automation 4

1.3 Contributions 6

1.4 Organization 7

2 Related Work **8**
2.1 Graph-Based Approaches . . . 9
2.2 Optimization-Based Approaches . . . 11
2.3 Machine Learning-Based Approaches . . . 12
2.4 Fusion-Based Approaches . . . 14
2.5 Summary . . . 17

3 Reinforcement Learning **18**
3.1 Q-learning . . . 18
3.2 Deep Q-Learning . . . 19
3.3 Deep Deterministic Policy Gradient . . . 20
3.4 Proximal Policy Optimization . . . 21
3.4.1 Surrogate Objective Function . . . 21
3.4.2 Dual Neural Networks: Policy and Value Function . . . 21
3.4.3 Balancing Exploration and Exploitation . . . 22
3.5 Recurrent Proximal Policy Optimization . . . 22
3.5.1 Recurrent Neural Networks . . . 22
3.5.2 Training Procedure . . . 23
3.6 Soft Actor Critic . . . 24
3.6.1 Entropy Maximization and Stochastic Policies . . . 24
3.6.2 Value Networks and Entropy Regularization . . . 24

4 High-Fidelity Simulation Environment & Software Architecture **26**
4.1 Adversarial Vehicles . . . 27
4.2 Pedestrians . . . 27
4.3 Software Architecture . . . 30

5 Methodology **35**

5.1 Problem Formulation . . . 35

5.2 Algorithm Architecture . . . 36

5.2.1 Feature Extractor Module . . . 37

5.2.2 PID Controllers for Longitudinal and Lateral Control . . . 38

5.3 Observation/State Representation . . . 38

5.4 Action Representations . . . 41

5.4.1 Continuous Action Space . . . 41

5.4.2 Discrete Action Space . . . 41

5.5 Reward Function . . . 42

5.5.1 Dense Components . . . 42

5.5.2 Sparse Components . . . 43

5.6 Hyperparameters . . . 44

5.7 Strategic Training: Unprotected Left Turns and Comprehensive Intersection Scenarios . . . 44

6 Results **46**

6.1 Distributed Training Environment and Machine Specifications . . . 47

6.2 Training on Unprotected Left Turns Only . . . 48

6.2.1 With Pedestrians . . . 48

6.2.2 Without Pedestrians . . . 50

6.3 Training in All Intersection Scenarios . . . 51

6.3.1 With Pedestrians . . . 51

6.3.2 Without Pedestrians . . . 52

6.4 Evaluation Metrics . . . 54

6.4.1 Training on Unprotected Left Turns Only . . . 54

6.4.2 Training on All Intersection Scenarios . . . 55

7 Conclusion and Future Work **57**

7.1 Conclusion 57

7.2 Future Work 58

References **60**

Glossary **66**

List of Figures

1.1 California AV collision statistics for 2014–2019 based on the defined 8 types of collisions [2, 3]. 3

1.2 Levels of driving automation as defined by Society of Automotive Engineers (SAE) J3016 (2021). Figure courtesy of [4]. Copyright ©2021, SAE International. 5

2.1 Flowchart of the Rapidly Exploring Random Tree (RRT) algorithm [5]. . . 10

4.1 Top-down view of the simulation environment consisting of the ego vehicle (in red), adversarial vehicles (in blue) and pedestrians (in green). The locations of the pedestrians, adversarial vehicles, and ego vehicle are randomly initialized for each episode. In this case, 3 adversarial vehicles were spawned in each lane, totaling to 9 other vehicles and 10 pedestrians for each crosswalk totaling to 40 pedestrians. Hence there were a total of 50 traffic participants in this scene including the ego vehicle. 29

4.2 Screeenshot of top down view of the CARLA simulation environment within the container, accessible via a Virtual Network Computing (VNC) client software. 32

4.3 Illustration showcasing the Dockerized containers, comprising N instances of CARLA servers alongside a singular main container, with the added capability of establishing a connection to the main container through Secure Shell (SSH). 33

4.4 An illustrative representation displaying multiple instances of the service shown in Figure 4.3, highlighting the architecture's versatility in being utilized on a single machine. Furthermore, this adaptability extends to replication, allowing instances to operate on individual development machines with seamless integration and robust functionality. 34

5.1 Proposed architecture for the ego vehicle decision-making and control. . . . 37

6.1 Episode mean length and reward during training for *unprotected left turns only* with four pedestrians. Recurrent PPO has the lowest episode mean length while SAC demonstrates the highest. Recurrent PPO demonstrates the highest episode mean reward while DDPG demonstrates the lowest. DDPG could potentially benefit from further training since the reward is seen to be increasing. 49

6.2 Episode length and reward during training for *unprotected left turns only* without pedestrians. Recurrent PPO demonstrates the lowest episode length alongside DQN and PPO while DDPG has the highest. PPO has the highest episode mean reward although there is a significant drop during the training. This is likely due to the retraining when the simulation environment segmentation faults due to a bug in CARLA. 50

6.3 Episode length and reward during training for *all intersection scenarios* with four pedestrians. PPO has the lowest episode mean length although, in all algorithms, there seem to be significant oscillations. All algorithms have a similar mean episode reward with SAC potentially benefiting from further training. 51

6.4 Episode length and reward during training for *all intersection scenarios* without pedestrians. Recurrent PPO has the lowest episode length with all algorithms having a mean episode length less than the one observed in the case when training with pedestrians (see Figure 6.3). The mean episode reward is highest for DDPG with DQN being the lowest. As compared to Figure 6.3, the reward is higher for all algorithms due to the absence of pedestrians, as expected. 53

List of Tables

2.1 Results of the unsignalized intersection experiments conducted in [6]. . . . 14
2.2 Summary of the papers. . . . 15

3.1 Comparison of Reinforcement Learning Algorithms . . . 25

4.1 Parameters for Simulation Environment . . . 28
4.2 Reward Weights within Simulation Environment . . . 30

5.1 PID Controller Parameters . . . 38
5.2 PID Controller Limits . . . 40
5.3 Values of Reward Constants . . . 43
5.4 Training Hyperparameters for the DRL Architectures . . . 44
5.5 The different environmental configurations used to train the DRL algorithms. 45

6.1 System specification for main server machine. . . . 47
6.2 System specification for local development machines. . . . 48
6.3 Comparison of the DRL algorithms within the hierarchical approach when trained for *unprotected left turns only* with four pedestrians. . . . 54
6.4 Comparison of the DRL algorithms within the hierarchical approach when trained for *unprotected left turns only* with no pedestrians. . . . 55
6.5 Comparison of the DRL algorithms within the hierarchical approach when trained for *all intersection scenarios* with four pedestrians. . . . 56

List of Abbreviations

ACG Automatic Curriculum Generation 12, 13

AGC Autonomous Golf Cart 59

AIS Automation and Intelligent Systems v

AV Autonomous Vehicle iv, xi, 1–4, 8, 12, 13, 26, 27, 35–37, 41, 42, 57, 59

CAV Connected and Autonomous Vehicle 8

CL-RRT Closed-Loop RRT 9

CPU Control Processing Unit 47, 48

CTR Collision-to-Timeout Ratio 14

DARPA Defense Advanced Research Projects Agency 9

DDPG Deep Deterministic Policy Gradient iv, xii, 6, 12, 20, 21, 24, 25, 44, 46, 48–50, 52, 53, 55, 57

DMV Department of Motor Vehicles 1

DNN Deep Neural Network 19, 25, 36–38, 57

DQN Deep Q-Learning iv, xii, 6, 12–14, 16, 19, 25, 44, 46, 48, 50, 53, 56–58

DRL Deep Reinforcement Learning iv, xiii, 6, 7, 13, 17, 18, 31, 36, 38, 44–46, 50, 52, 54–58

GPR Gaussian Process Regression 11

GPU Graphical Processing Unit 47, 48

IRL Inverse Reinforcement Learning (RL) 13, 16

LSTM Long Short-Term Memory 22, 44

MDP Markov Decision Process xvii, 17, 35, 36

MIT Massachusetts Institute of Technology 9

MPC Model Predictive Control 11, 13–16, 59

NHTSA National Highway Traffic Safety Administration 1, 2, 8

OS Operating System 47, 48

PID Proportional-Integral-Derivative iv, xiii, 6, 27, 36, 38, 40, 41, 57

PPO Proximal Policy Optimization iv, xii, 6, 21–23, 25, 44, 46, 48–53, 55–58

RAM Random Access Memory xvi, 47, 48

ReLU Rectified Linear Unit 44

RL Reinforcement Learning xv, 12–14, 16, 18, 21–26, 42

RNN Recurrent Neural Network 22, 23, 25

RRT Rapidly Exploring Random Tree xi, xiv, 9–11, 15

SAC Soft Actor Critic iv, xii, 6, 24, 25, 44, 46, 48, 49, 51, 52, 55, 57

SAE Society of Automotive Engineers xi, 4, 5

SM Sliding-Mode Control 14

SSH Secure Shell xi, 33

TD3 Twin Delayed Deep Deterministic Policy Gradient 13, 16

TTC Time-to-Collision 12, 27

V2V Vehicle-to-Vehicle 11, 15

VNC Virtual Network Computing xi, 31, 32

VRAM Video Random Access Memory (RAM) 47

WHO World Health Organization 1

List of Symbols

a_t Action taken at time t xvii, 18, 20, 24

A set of all possible actions that an agent can take in the Markov Decision Process (MDP). An action is a decision or choice made by the agent to influence the environment. 18, 35

α Learning rate of the neural network 19

R_t Expected return which is the sum of discounted rewards over a certain time horizon T 18

γ Discount factor 18, 35, 44

π Policy function which maps the states deterministically to actions 18, 24

r_t Reward given to a policy at each timestep t 18

R Reward function, which maps a triple of elements (s_t, a_t, $s_t^{'}$ to a real-valued reward value $R(s, a, s^{'})$ 18, 35

$s_t^{'}$ Next state after taking action a_t in state s_t xvii, 18

s_t State at time t xvii, 18, 20, 23, 24

P State transition probability function that describes the likelihood of transitioning from one state to another given a specific action. It is defined as $P(s^{'}|s, a)$ 18, 35

S Set of all possible states in the MDP. Each state represents a specific situation or configuration of the environment. 18, 35

θ Weights of the neural network 19

t Continuous time xvii, 18

Chapter 1

Introduction

1.1 Motivation

Several organizations, both in academia and industry, are rigorously working towards developing AVs which have the potential to save thousands of lives every year and create significant societal benefits [7, 8]. According to a report by the World Health Organization (WHO), approximately 1.35 million road traffic deaths occur every year with an additional 20–50 million people suffering nonfatal injuries [7]. A study carried out by the National Highway Traffic Safety Administration (NHTSA) in the United States concluded that 94% of accidents occurred due to human errors while only 2% were caused by vehicle failures [9]. Hence, encouragingly, the increased adoption of AVs will likely reduce vehicle accidents and hence decrease the fatalities due to road traffic. If their widespread deployment is successful, the projected annual social benefits of AVs—which include reducing traffic congestion and the number of accidents on the road, consuming less energy, and boosting productivity as a result of reallocating driving time—will reach nearly $800 billion by 2050 [10].

Although the technology for vehicles moving in static environments is well developed [11], the dynamic nature of the real-world environment has made it extremely challenging for widespread AV adoption [12]. To experiment with AV technologies in the real world, the Department of Motor Vehicles (DMV) in California started issuing permits to manufacturers in 2014 under the Autonomous Vehicle Tester program [13]. California's DMV requires manufacturers to test AVs and report any collision that resulted in bodily injury, property damage, or death. As of 24 June 2022, 483 collision reports have been filed since the start of the program in 2014 [2].

Since these reports are individually filed by each manufacturer, an analysis needs to be conducted to understand the trends. Based on the analysis conducted by [3] on the data between 2014 and 2019, it was realized that the most challenging situation for AVs occurred when there were changes in the road surface conditions. For instance, when the road is wet, AVs are more vulnerable to accidents. The study used the Pearson chi-square test for calculating the relation between various elements and considered both manual override and autonomous modes. Although the authors did discuss hit-and-run cases, there was no mention of how AVs respond to such situations. The study also demonstrated that none of the AVs struck any stationary vehicle or object, regardless of the weather and road conditions. Although this is remarkable, it is also expected due to the extensive research being conducted on AV perception tasks as well as improvements in perception data acquisition technologies (camera, LiDAR, etc.). Moreover, 38% of the accidents occurred when AVs were in manual mode since the driver often disengaged the autonomous mode during erratic situations. It should also be noted that [3] found that 62% of the collisions were rear-ended, implying that the following vehicle was at fault. A chart showing the types of collisions and their percentages for the years 2014–2019 is indicated in Figure 1.1. A more recent study [14] suggests that these rear-ended crashes are likely due to the conservative behavior of AVs. An analysis combining the collision and disengagement statistics will provide further insight into the performance of AVs and their shortcomings.

One of the most complicated and difficult situations for AV navigation is intersections in urban areas. According to several studies on road accidents, junctions account for 60% of serious traffic injuries in Europe [15]. According to the NHTSA report on traffic accidents, 29% of all car crashes and 18% of pedestrian fatalities occurred at intersections in 2019 [16]. Furthermore, Cruise, the self-driving division of General Motors, revealed that their currently deployed AVs in San Francisco make 1400 unprotected left turns every 24 hours, claiming it to be one of the most difficult maneuvers [17]. They also had to recall their entire fleet in early September 2022 due to these left turns [18]. Similarly, Waymo, the Google self-driving car company also struggled with taking left turns in Phoenix.

This is due to the highly interactive nature of intersections which requires AVs to perceive and take into account large amounts of information from the environment. In particular, the AV needs to pay attention to traffic indicators such as stop signs and traffic lights while also being mindful of other vehicles, estimating their speeds and intentions as well as other road users including pedestrians. As a result, implementing a rule-based agent that allows for safe as well as reliable decisions is extremely challenging.

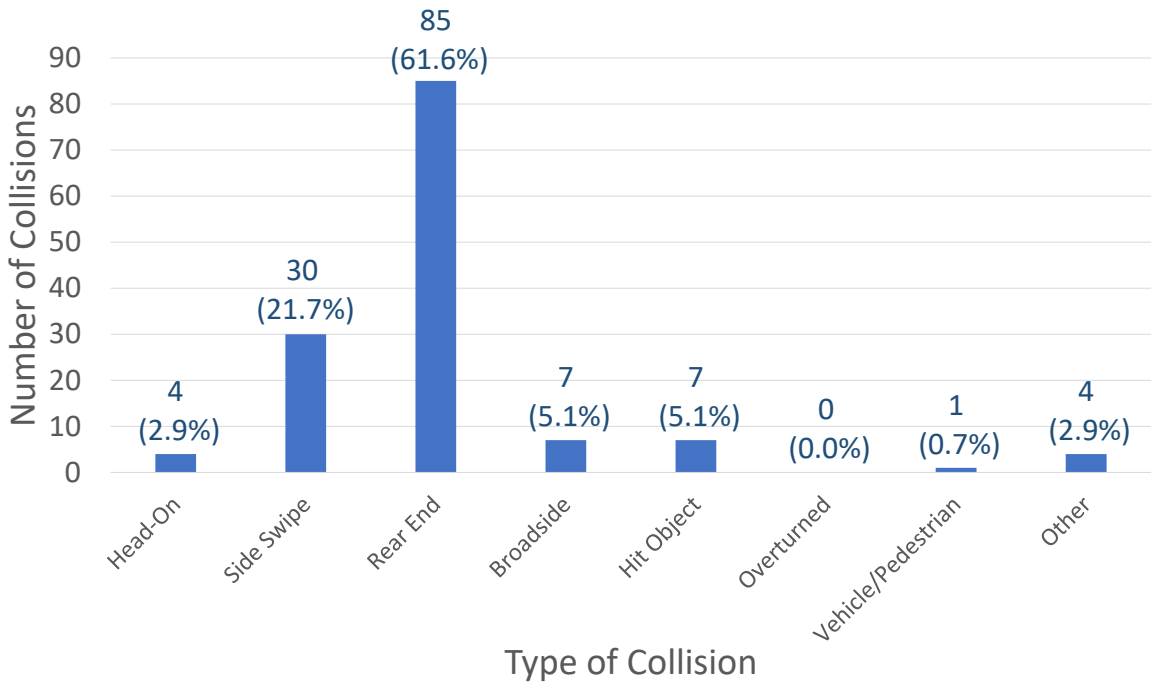

Figure 1.1: California AV collision statistics for 2014–2019 based on the defined 8 types of collisions [2, 3].

1.2 Levels of Automation

In 2014, the SAE released a seminal document that outlined a comprehensive framework for categorizing the levels of automation in AVs. Over the years, this document has undergone several revisions, with the most recent update occurring in April 2021 [4].

The SAE's classification system divides automation into six distinct levels, each delineating the extent of human involvement in the driving process. Levels 0–2 place the primary responsibility on the human driver, with automation providing supplementary support. In contrast, levels 3–5 introduce advanced automated driving features, wherein the vehicle assumes a more dominant role. Figure 1.2 provides a visual representation of these levels, while a brief description of each level is presented below:

- **Level 0: No driving automation.** The vehicle relies entirely on the human driver, with no automated assistance available.
- **Level 1: Driver assistance.** The vehicle offers limited assistance, such as steering or brake/acceleration support, to the driver.
- **Level 2: Partial automation.** The vehicle provides both steering and brake or acceleration support simultaneously, but the driver remains responsible for monitoring the driving environment.
- **Level 3: Conditional automation.** The vehicle can manage most aspects of driving under specific conditions, but the driver must be ready to intervene when necessary.
- **Level 4: High driving automation.** The vehicle is capable of operating without driver input or intervention but is limited to predefined conditions and environments.
- **Level 5: Full automation.** The vehicle operates autonomously without requiring any driver input, functioning in all conditions and environments.

More detailed information about each level can be found in the official SAE J3016 document [4]. In the context of this research study, our focus lies on evaluating the capabilities of AVs at the highest level of automation, namely, SAE level 5.

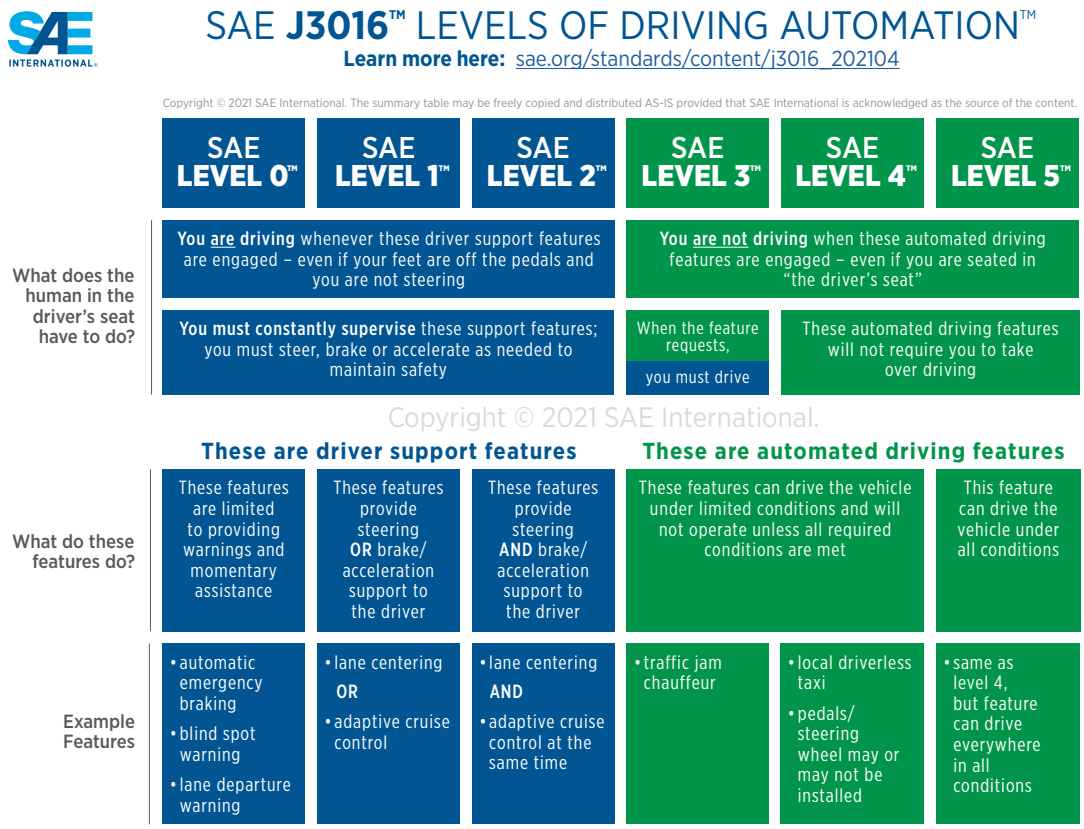

Figure 1.2: Levels of driving automation as defined by SAE J3016 (2021). Figure courtesy of [4]. Copyright ©2021, SAE International.

1.3 Contributions

The main contributions of this thesis are as follows:

1. An advanced hierarchical framework is developed, comprising a global planner, high-level decision-making policy, and low-level PID controllers. The A* algorithm sets the global plan, while the DRL algorithm sets the high-level decision making policy, particularly the target speed, and the low level PID controllers attempt to follow this target speed. The following state-of-the-art discrete and continuous action space DRL algorithms are used: DQN, DDPG, PPO, Recurrent PPO and SAC.

2. A high-fidelity simulation environment consisting of crosswalks, pedestrians, and adversarial vehicles is developed using CARLA [19]. The environment is designed to be stochastic with the pedestrian behavior also controlled by an artificial agent, in particular, the AIController within the Unreal Engine. The entire simulation environment is made publicly available for further research and experimentation at https://github.com/faizansana/intersection-carla-gym.

3. Catering to individual driver preferences, a customizable speed parameter is integrated into both continuous and discrete action spaces. This parameter can be adjusted during deployment, offering flexibility to accommodate various driving styles.

4. All algorithms are trained and evaluated within the high-fidelity simulation environment with extensive testing and analysis. Each algorithm is trained for one million time steps and tested for 1000 episodes which translates to approximately 12 hours of training and 2 hours of test time per algorithm. The total time commitment would be 70 hours if all of the algorithms were to be trained and tested one after the other.

5. A containerized software architecture is carefully crafted to fulfill the dual functions of testing and training, allowing for quick and iterative experimentation on a variety of server systems. This carefully designed framework offers a smooth and flexible environment for thorough analysis and validation, laying the groundwork for rigorous research endeavors. This is demonstrated in the algorithm repository available at https://github.com/faizansana/intersection-driving.

It is important to note that unsignalized intersections in this context refer to fully uncontrolled intersections.

1.4 Organization

This thesis is organized as follows:

- Chapter 2 reviews existing research in the field, categorized into graph-based approaches, optimization-based approaches, machine learning-based approaches, and fusion-based methods.
- Chapter 3 gives a brief overview of reinforcement learning and discusses in details the DRL approaches.
- Chapter 4 explains the details and uniqueness of the CARLA simulation environment design that is used to train the algorithm. This chapter offers researchers looking to easily repeat and evaluate results a deep investigation of the subtleties of the simulation environment as well as an in-depth look at the containerized software architecture. The thorough explanation guarantees an open and repeatable experimental design for the research and future work.
- Chapter 5 delves into the technical aspects of the research, starting with problem formulation and defining the observation and action spaces. It also discusses the hierarchical algorithm, reward function design as well as the defined metrics, providing a comprehensive overview of the methodological framework employed in this study.
- Chapter 6 discusses the training and test results in simulation.

Chapter 2

Related Work

In metropolitan locations, driving across unsignalized, fully uncontrolled, intersections is always a challenging problem for AVs. The vehicle should decide when and how to cross the junction safely instead of just being cautious in these circumstances as there typically is not a traffic light to regulate priorities. Furthermore, these crossroads are frequently obstructed, making the vehicle's decision-making task more complex. This is also evident in the 2019 Traffic Safety report by the NHTSA [16] in which it is observed that 25% of the accidents occurred at intersections. To tackle intersection scenarios, research on the proposed solutions can be roughly classified into two categories: (a) research on traffic elements, and (b) research on AV navigation. Research about part (a) focuses on signal control at intersections and topological characteristics of the road infrastructure. Some researchers have proposed using different intersection topologies through demonstrating their effectiveness to improve traffic flow while ensuring safety [20], while others focus on optimizing the control of traffic lights at intersections [21, 22, 23]. These control strategies improve traffic flow if all approaches to an intersection are not equally congested. However, regardless of the level of traffic, these *methods cannot completely eliminate the stop delay of vehicles* [24]. Hence, to increase traffic efficiency and reduce pollution to a greater extent, several studies focus on the autonomous driving strategy at intersections. Since AVs and human-driven vehicles are expected to be on the roads in mixed traffic conditions for the foreseeable future, only research within mixed traffic conditions is considered. Also, those studies focusing on Connected and Autonomous Vehicle (CAV)s are ignored as part of this review. The reader is referred to [25, 26, 27] for further discussions on the navigation of CAVs.

These navigation algorithms to traverse unsignalized intersections can be broadly categorized into four groups, namely, graph-based, optimization-based approaches, machine-

learning-based methods, and approaches that combine two or more of these methods. These approaches are discussed below.

2.1 Graph-Based Approaches

Graph-based methods such as A* search [28], Dijkstra, and other search algorithms are commonly used in mobile robotics for path planning purposes [11]. Since dynamic path planning is required in the case of intersections, the RRT algorithm [5] is commonly used. RRT is a sampling-based method that can find a possible path within a comparatively short amount of time and react to changes in the environment. The tree expansion phase and the path construction step are two fundamental components of the RRT method. The algorithm generates an initial random configuration and attempts to link it to the current tree during the tree expansion stage. The distance measure is used to first locate the node in the tree that is closest to the new configuration which is then used to generate a new configuration. If the new arrangement avoids collisions, it is linked to the closest node and inserted as a new node to the tree. During the path construction step, the algorithm attempts to find a path from the start configuration to the goal configuration. Starting with the goal configuration, the algorithm locates the node in the tree that is closest to the goal. It then moves upwards, being terminated when the start configuration is reached. This algorithm is shown in Figure 2.1.

Since the proposal of RRT, several variants have been developed, including the RRT* algorithm [29] and Closed-Loop RRT (CL-RRT) [30] which are often used as benchmarks. CL-RRT was developed by the Massachusetts Institute of Technology (MIT) team for the 2007 Defense Advanced Research Projects Agency (DARPA) Grand Challenge for their motion planning and control subsystem. As compared to the traditional RRT, the vehicle model is used to generate more feasible paths considering the kinematics during the path generation process. In [31], a faster version of the RRT algorithm was introduced by developing a rule-template set based on traffic scenes and an aggressive extension strategy of the search tree itself. These rules were generated offline based on the context of the traffic scenes and saved as templates to be selected based on the short-term goal state. The search tree was repeatedly regenerated at high frequency to enable obstacle avoidance. In [32], the authors proposed a spline-based RRT* approach where the minimum turning radius (or kinematics) of vehicles was satisfied using cubic Bezier curves. They approximated the shape of the vehicle as an oriented rectangle and used it to confirm the required space of a moving vehicle. By using Bezier curves, they were able to ensure that the generated paths were continuous and satisfied the constraints of the vehicle motion [33]. In [34], a prediction

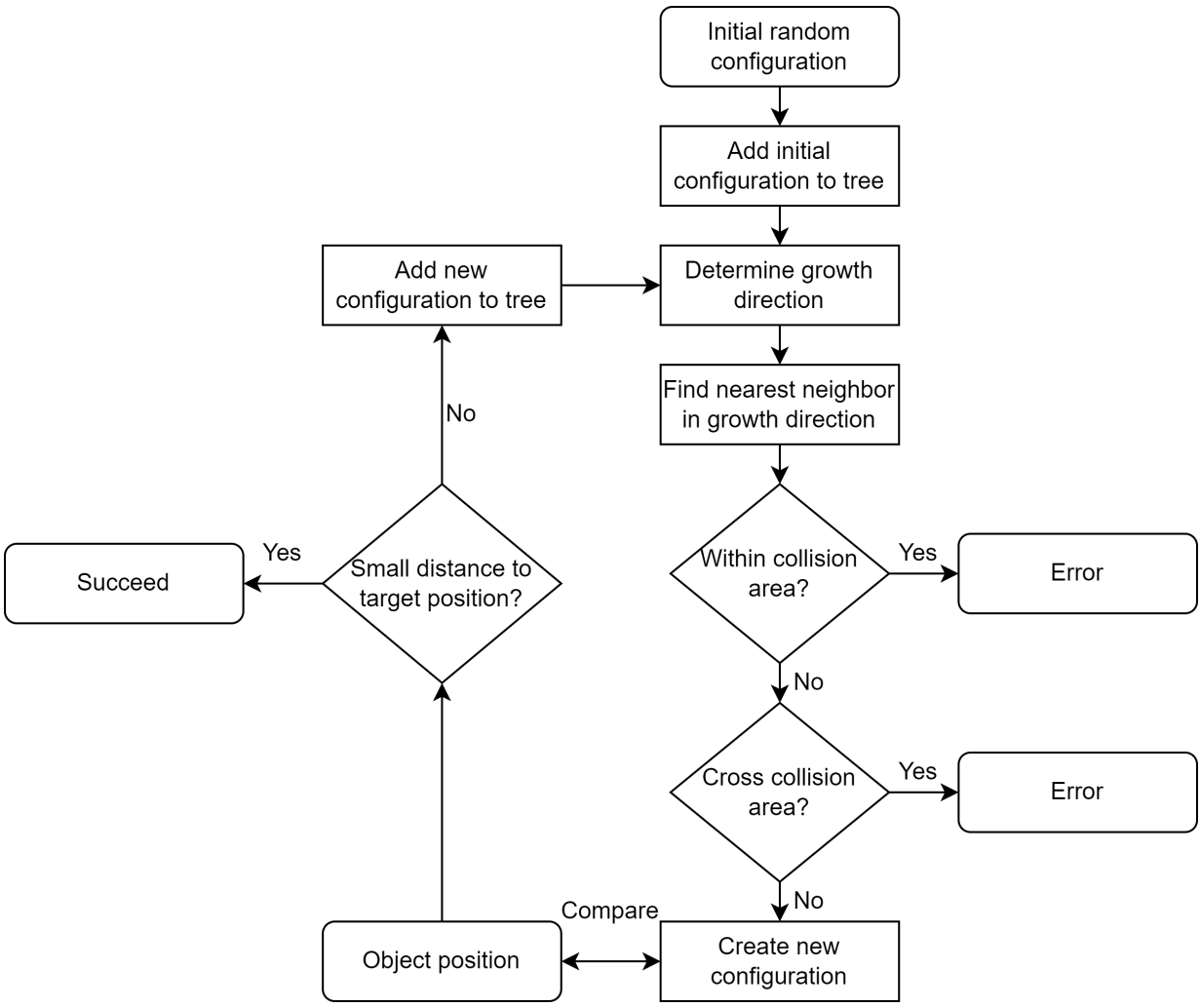

Figure 2.1: Flowchart of the RRT algorithm [5].

algorithm based on Gaussian Process Regression (GPR) to predict the future locations of vehicles was combined with the RRT algorithm for motion planning. The proposed method was evaluated in simulations at a four-way intersection and the capabilities of fusing probabilistic maps with sampling-based planning methods was demonstrated.

Overall, the main issue of graph search algorithms is that, although they work well in static environments, with intricacies and swift transitions in intersection environments, they usually produce conservative decisions. The refresh rate of the planner can also be challenging, particularly in critical corner-case scenarios. Nonetheless, algorithms based on graph search methods are interpretable when failures occur.

2.2 Optimization-Based Approaches

Another common idea for navigation at unsignalized intersections is to formulate the situation as a real-time optimization problem. This involves setting up a cost function, boundary conditions, and constraints. One of the most popular methods used in this category is the Model Predictive Control (MPC) approach. The key components of an MPC problem formulation include obstacle models, ego vehicle models, and a proper optimization solver.

In [35], a Monte Carlo simulation was created to predict the probabilistic occupancy of other objects on the map, and then, MPC was used to optimize the reference trajectory based on the current state of the vehicle in a hierarchical fashion. The probabilistic occupancies of the road traffic users were computed offline and the results were subsequently used to reduce the real-time computational load. In [36], a bilevel controller was described, consisting of (a) a coordination level and (b) a vehicle level. At the coordination level, the occupancy time slots at an intersection were calculated while the control commands were given at the vehicle level. However, they made use of Vehicle-to-Vehicle (V2V) communications to coordinate that planning. A unified path planning approach using MPC was devised in [37] with the capability of automatically selecting appropriate parameters for various types of maneuvers. By modeling the surrounding vehicles as polygons and developing a lane-associated potential field, the authors could provide better driving comfort while ensuring safety.

Other optimal control techniques such as the Bezier curve optimization method have also been implemented for intersection traversing. In [38], the Bezier curve optimization method was used to cope with the constraints of obstacles at an intersection and solve an optimization problem through a combination of Lagrangian and gradient-based methods. To consider kinematic constraints, the authors used a nonlinear kinematic model with

slip-free rolling conditions and formalized it into an optimization problem. They used the Bezier curve parameters to find a new path in the presence of an obstacle via minimizing quadratic errors between an initial reference path and the newly generated path.

Although these optimization-based approaches are deterministic, several unrealistic assumptions have to be made to formulate the optimization problem and solve it efficiently.

2.3 Machine Learning-Based Approaches

Another common methodology is to use machine learning, particularly neural networks, due to their universal approximation property. The authors of [39] presented a RL-based approach, namely, DQN, to drive an AV through occluded intersections. Instead of using a sparse rewarding scheme in which rewards are based on collisions, they proposed the use of a risk-based reward function for punishing risky situations. The risk was defined as follows: a safe stop condition where the ego vehicle could stop behind a conflict zone, and a safe leave condition wherein the ego vehicle could enter the conflict zone before another vehicle or if another vehicle had already left the zone. The RL agent learned a high-level policy where the action space only consisted of stop, drive fast, and drive slow actions. The actuation was handled by the low-level controllers, which the authors claimed improved the quality of learning and allowed for lower update rates by the RL policy. The risk-aware DQN approach was compared against collision-aware DQN and a rule-based policy in CARLA [19]. More information on the rule-based policy can be found in [39]. It was found that the collision-aware DQN approach was less stable during the training, which was done for 400 thousand training steps. During their experiments, it was seen that the rule-based policy was the most conservative while the collision-aware DQN approach was the most aggressive. They also assessed these algorithms on more challenging scenarios with dense traffic, severe occlusion, increased sensor noise, and a shorter sensor range (40 m). Risk-aware DQN had the highest success rate in all scenarios with the lowest being 80% in dense traffic. However, the intersections used in that work did not have crosswalks and every vehicle was assumed to be of the same length.

In [40], curriculum learning was used to learn driving behavior at four-way urban intersections. Curriculum learning was first introduced in [41] to speed up the learning process by first training a model with a simpler task and gradually increasing the complexity of the problem. Since designing the curriculum itself is a challenging task, the authors proposed an Automatic Curriculum Generation (ACG) algorithm. They trained a DQN and a DDPG algorithm using the ACG and random curricula. They also used a rule-based algorithm and the Time-to-Collision (TTC) to compare the results. The algorithms were

evaluated in two scenarios: (a) intersection approaching in which the ego vehicle had to stop at the stop line and (b) intersection traversing. The algorithm trained using the ACG had the best mean reward with a success rate of 98.7% for the intersection-approaching scenario and 82.1% during the intersection traversing. It is also important to note that the ACG-based curricula required the lowest number of training steps, implying a more efficient training. However, the algorithm needed to be tested in other complex scenarios to show more robustness. Moreover, the authors only considered other vehicles as road users.

In [42], an RL agent was introduced that was aware of the effects that the ego vehicle would have on other human-driven vehicles and leveraged that information to improve efficiency. The authors trained a model of a human driver using Inverse RL (IRL) which was used to signify how a human driver would react to the actions of other vehicles. That model was then used as part of the reward function of the AV's RL agent. They tested their algorithm in simulations and demonstrated that the RL agent could be taught to let human participants go first by reversing itself at an intersection. This was a human-interpretable result that was not explicitly programmed. However, they assumed that the agent had a bird's-eye view of the environment and therefore had access to all the states. They also only considered interaction with a single human driver and argued that modeling interactions with multiple road users was not immediately clear.

In [43], a DRL agent using DQN was trained using CARLA to control the AV within a complex scenario involving motorcycles, pedestrians and adversarial vehicles. The input data comprised four consecutive frames from a frontal camera, resized to 144x144, along with a 2x1 vector specifying the AV's destination. The action space encompassed nine discrete actions, allowing for variations in steering, throttle, and brake combinations. The reward function was designed to penalize crashes, invasion of other lanes and rewarded positively for reaching the destination. The agent was tested within five different scenarios involving various number of pedestrians and adversarial vehicles. However, the destination was always set to a waypoint which involved the ego vehicle only going straight. Although training was done for 3,000 episodes, testing was only done for a mere 10 episodes for each scenario. It was also realized that the different scenarios, although containing increasing number of pedestrians and vehicles, were spawned in different parts of the town randomly rather than at the intersection itself. Authors of [44], combine a DRL agent with MPC to derive the optimal policy. The coupled RL and MPC architecture is run in parallel and control output is selected depending on the safety discrete controller. They trained a Twin Delayed Deep Deterministic Policy Gradient (TD3) to output longitudinal control and a path selector for lateral control.

2.4 Fusion-Based Approaches

To deal with the drawbacks of the individual methods, some researchers have focused on combining them.

For instance, in [45], a hierarchical algorithm was introduced in which a high-level decision-maker made decisions for how the vehicle had to drive through an intersection and a low-level planner optimized the motion trajectories to be safe. The high-level decision-maker was implemented using DQN, an RL framework, while the low-level planner used MPC. The MPC method was not used directly since it would significantly add to the computational complexity due to an increasing number of vehicles at an intersection. Using the traffic configuration, the MPC module optimized the trajectory while the high-level decision-maker solved the planning problem. The authors in [6] examined their algorithm in simulations and compared it using Sliding-Mode Control (SM). They performed two experiments, an intersection with a single crossing and one with a double crossing, and benchmarked it using the success rate and the Collision-to-Timeout Ratio (CTR). Table 2.1 shows the results of their experiments.

Table 2.1: Results of the unsignalized intersection experiments conducted in [6].

Controller	**Success Rate**		**CTR**	
	Single	Double	Single	Double
SM	96.1%	90.9%	72%	93%
MPC	97.3%	95.2%	45%	76%

As seen in Table 2.1, the algorithm with MPC outperformed the SM in both cases. In addition, they found that the agent involving MPC converged much faster due to the immediate reward available to the RL policy (10^4 vs. 4.10^4 training episodes). However, they limited the scope of the problem by only considering at most four vehicles at any moment. They also focused on the longitudinal control while assuming the lateral control already existed. Additionally, they only considered other vehicles although they did mention that the method could be extended to other road users. An enhancement to the model would be the incorporation of a safety layer after the decision-making algorithm to limit the acceleration values for the system to stay safe. A collision avoidance system that uses the environment status and desired accelerations to determine if the present path has a collision risk and enables much better bounds to prevent the collision would also likely enhance the outcomes. A summary of the papers discussed is shown in Table 2.2.

Table 2.2: Summary of the papers.

Paper	Pros	Cons
[31]	Allowed for obstacle avoidance due to the high-frequency regeneration of a tree.	Computationally expensive. Required much offline computation.
[32]	Ensured that paths generated were continuous and satisfied vehicle constraints. Tested in the real world.	Assumed that vehicles followed the path exactly with minimal error.
[34]	Fused stochastic maps with a sampling-based RRT algorithm. Tested at a 4-way unsignalized intersection.	The model was created for predicting other vehicle's future locations and could not account for varying vehicle speeds.
[35]	Incorporated a safety process within the decision-making stage. The whole process of planning and safety assessment took less than 100 ms.	Assumed other vehicles kept a constant velocity along the road section. They did not account for unusual events such as jaywalking, sudden reversing, etc.
[36]	Improved stability and performance at intersections by using a bilevel controller. Demonstrated that the controller had a high performance even in cases of high positioning error.	Used V2V communication which does not always exist in current systems. Controller needs to be tested in the situation of continuously oncoming vehicles.
[37]	Allowed maneuvers such as ramp merging, lane change, etc., to be determined by the MPC generated path. Infused both safety as well as comfort within the MPC constraints.	Assumed fully observable environment.
[38]	Low computational cost of replanning.	Assumed no noise in perception.

Continued on next page

Table 2.2 – continued from previous page

Paper	Pros	Cons
[39]	Accounted for occlusions that occur at intersections. Defined a risk-based reward function instead of a sparse rewarding scheme.	Discretized action space (3 actions). Assumed all other vehicles were of the same length.
[40]	Used curriculum learning [41] to speed up training. High success rate for both intersection-traversing and -approaching scenarios.	Considered traffic users to only consist of vehicles. Required further testing in more complex scenarios.
[42]	Used IRL and incorporated a policy into the reward function while training the RL agent.	Assumed a fully observable environment with a bird's-eye view of the environment. Only considered interaction with a single other human driver.
[43]	Trained a DQN agent within CARLA to go straight at a complex intersection including pedestrians, cyclists and adversarial vehicles. Outputted both lateral and longitudinal control.	Testing was limited to only 10 episodes. Variation of pedestrians and adversarial vehicles was in entire region rather than specifically at intersection.
[44]	Trained a TD3 algorithm in parallel with MPC where each model was selected based on a binary safety controller.	Trained and tested within custom environment.
[45]	Used a hierarchical controller to separate high-level decision-making from lower-level control. Faster model convergence.	No safety layers. Only outputted longitudinal control, assuming lateral control existed.

2.5 Summary

A couple of key issues in the above studies along with a possible solution are summarized as follows:

- **Complexity of Intersection Scenarios:**
  - Many existing studies employ overly simplistic intersection scenarios for algorithm testing and development. Pedestrian dynamics are often neglected, and simple kinematic models are used, leading to a lack of realism in the simulated environment.
  - **Solution:** Develop a more realistic and dynamic environment using a high-fidelity physics engine, incorporating pedestrian presence at each crosswalk.
- **Limited Accessibility and Collaboration:**
  - Most studies customarily develop unique environments, restricting accessibility for other researchers to test and validate findings. This hinders the ability to benchmark between algorithms and impedes collaborative progress in the field.
  - **Solution:** Develop an open-source simulation environment, fostering collaboration and advancement in the field.
- **Challenges with Model-Based Techniques:**
  - Model-based techniques, while deterministic, pose challenges in real-world applicability due to the necessity of an MDP formulation and therefore the accurate estimation of transition probabilities.
  - **Solution:** Propose a balanced approach by combining the deterministic nature of traditional control algorithms with the adaptability and learning capabilities of DRL. This hybrid solution aims to address the complexity of real-world scenarios, ensuring both predictability and adaptability in algorithmic decision-making.

Chapter 3

Reinforcement Learning

This chapter provides a brief overview of RL and explains the DRL algorithms used for training. Within the RL framework, an agent is trained to take an action a_t at time t within a state s_t. The policy is normally referred to as π. The agent then transitions to state $s_t^{'}$ receiving a reward of r_t. The state transition probability $P : S \times A \times S \rightarrow [0, 1]$ represents the system dynamics, the reward function $R : S \times A \times S \rightarrow \mathbb{R}$ gives the real valued reward at each time step, and $\gamma \in (0, 1]$ is the discount factor which prioritizes earlier rewards and provides stability in the cases of infinite time horizon problems [46].

The goal of RL is to select a sequence of actions to maximize the expected return $R_t = \sum_{k=0}^{T} \gamma^t r_{t+k}$ according to a policy $\pi : S \rightarrow A$ on state s_t.

In essence, the overarching objective of RL is to devise a policy π that strategically guides the agent through the environment to maximize the expected return R_t, thus achieving the underlying goal with optimal efficiency. This endeavor entails striking a balance between exploring new avenues and exploiting acquired knowledge to secure the best possible long-term rewards.

3.1 Q-learning

In Q-learning, the state-action value function $Q^{\pi}(s, a) = \mathbb{E}[\sum_{t=1}^{\infty} \gamma^t r_t | s_0 = s]$ for a policy π represents the discounted accumulated reward obtained by the agent when taking action a_t from state s_t and then following policy π. Given an optimal policy defined by the following Bellman equation:

$$Q^*(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \max_{a'} Q^*(s, a) | s, a] \tag{3.1}$$

the optimal policy can then be found using the maximum Q value at every time step.

$$\pi^*(s) = arg\max_{a'} Q^*(s,a) \tag{3.2}$$

Q-learning is a model-free reinforcement learning algorithm, meaning it does not require a model of the environment's dynamics. Instead, it directly learns the Q-values through interactions with the environment. Once the Q-values are sufficiently learned, the agent can select actions by choosing the one with the highest Q-value in a given state, resulting in effective decision-making.

Although equation 3.1 can be solved using value iteration for finite state and action space, approximate methods must be used for continuous and high dimensional state spaces. This is the reason behind the development of DQN [47].

3.2 Deep Q-Learning

In order to be able to approximate the optimal state-action value function for continuous state spaces, Deep Neural Network (DNN)s are used where $Q^*(s,a) \approx Q(s,a;\theta)$ where θ represents the weights of the network [48]. Therefore, the solution to equation 3.1 can be approximated by this neural network by minimizing the following loss function:

$$L(\theta) = \mathbb{E}_{s'}[(r(s,a) + \gamma \max_{a'} Q^*(s,a;\theta^-) - Q(s,a;\theta))^2] \tag{3.3}$$

where θ^- represents the parameters of a fixed and separate target network. This is required in order to ensure stability and convergence of the training along with a experience replay buffer that is used to generate batches of training samples. This algorithm is known as DQN. Given an experience sample $(s,a,r,s^{'})$, the neural network weights are updated as follows:

$$\theta \leftarrow \theta + \alpha(r + \gamma \max_{a'} Q^*(s,a;\theta^-) - Q(s,a;\theta))\nabla_\theta Q(s,a;\theta) \tag{3.4}$$

where α is the learning rate.

As seen in this case, although DQNs can handle continuous state spaces, they can only output a discrete set of actions. Although a continuous state space can be discretized, this leads to the problem of the curse of dimensionality [49].

3.3 Deep Deterministic Policy Gradient

To address the challenges posed by high-dimensional continuous action spaces in reinforcement learning, a breakthrough approach was introduced by Lillicrap et al. in their work [50], where they presented the DDPG algorithm. This method extends the Q-learning framework to continuous action domains, offering a potent solution for a wide range of real-world applications.

The DDPG algorithm is characterized by its utilization of an actor-critic architecture, which involves two distinct neural networks: an actor network represented by $\mu(s, \theta^{\mu})$, where θ^{μ} denotes the network parameters, and a critic network denoted as $Q(s, a; \theta^{Q})$, with θ^{Q} being its associated parameters. In practice, when provided with a state s_t, the actor network produces an action a_t, which is subsequently passed to the critic network. Given the current state s_t and the action a_t, the critic network evaluates the quality of the action by predicting its state-action value or Q value.

To ensure training stability and convergence, the DDPG algorithm employs the concept of target networks, denoted as μ^{-} and Q^{-}. In essence, these target networks are periodically updated to track the learning progress. Consequently, the DDPG architecture is comprised of four neural networks: two actor networks and two critic networks.

The target policy network is updated using the following equation:

$$\theta^{\mu^{'}} = \tau\theta^{\mu} + (1 - \tau)\theta^{\mu^{-}} \tag{3.5}$$

Similarly, the target critic network is updated using a similar equation:

$$\theta^{Q^{'}} = \tau\theta^{Q} + (1 - \tau)\theta^{Q^{-}} \tag{3.6}$$

Here $\tau << 1$ is a hyperparameter used for the target update rate.

To balance the trade-off between exploration and exploitation, an exploratory noise term ζ_t is commonly added to the deterministic action derived from the actor network:

$$a_t = \mu(s_t|\theta^{\mu}) + \zeta_t \tag{3.7}$$

This noise introduces a controlled level of randomness, allowing the agent to explore different regions of the action space while leveraging its learned policy. The interplay between deterministic and stochastic actions enables effective learning and adaptation in complex and continuous environments.

In summary, the DDPG algorithm provides a powerful framework for addressing the challenges of continuous action spaces in RL. By employing an actor-critic architecture, target networks, and a balanced exploration strategy, DDPG facilitates the training of agents in real-world scenarios that involve intricate and high-dimensional action spaces [51].

3.4 Proximal Policy Optimization

PPO represents a significant advancement in RL, specifically within the realm of on-policy policy gradient methods. This approach offers a robust and efficient framework for training agents in complex environments by optimizing a surrogate objective function. PPO tackles the fundamental challenge of achieving a balance between exploration and exploitation, a pivotal aspect in RL [52].

3.4.1 Surrogate Objective Function

At the core of PPO lies the optimization of a surrogate objective function, which serves as an approximation of the true objective—maximizing the expected cumulative reward. The surrogate objective guides the updates to the agent's policy in a manner that ensures gradual and consistent improvement.

The cornerstone of PPO's effectiveness is the use of a clipped surrogate objective, given by the following equation:

$$L_{\text{CLIP}}(\theta) = \min(r_t(\theta) \cdot A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t) \tag{3.8}$$

In this equation, $r_t(\theta)$ denotes the probability ratio between the new and old policies for a specific action taken in a particular state. This probability ratio provides a measure of how much the policy has changed. The term A_t represents the Advantage, a value indicating the relative advantage of taking an action in a particular state. The clipping operation constrains the policy update to a range defined by $(1-\epsilon)$ and $(1+\epsilon)$. This prevents overly large policy updates that could destabilize the learning process.

3.4.2 Dual Neural Networks: Policy and Value Function

Similar to the DDPG approach, PPO employs a dual-network architecture consisting of two essential components: the policy network and the value function network.

1. **Policy Network (Actor)**: The policy network, often referred to as the actor, plays a pivotal role in determining the agent's action selection policy. Given a state, the policy network outputs a probability distribution over available actions, guiding the agent's decision-making process.

2. **Value Function Network (Critic)**: The value function network, or critic, serves as an essential component for estimating the expected cumulative reward associated with a particular state. By approximating the value of states, the critic assists the agent in evaluating the desirability of various actions.

3.4.3 Balancing Exploration and Exploitation

PPO's strength lies in its ability to balance exploration and exploitation effectively. The clipped surrogate objective enforces controlled policy updates, preventing drastic shifts that might hinder the learning process. This measured approach to policy updates ensures that the agent gradually improves its performance while mitigating the risks of destabilization [53].

In conclusion, PPO presents a powerful paradigm for training agents in RL. Through its surrogate objective optimization and clipped policy updates, PPO achieves a delicate equilibrium between exploration and exploitation. By utilizing dual neural networks for policy and value function estimation, PPO tackles the challenges posed by complex environments, contributing significantly to the advancement of RL techniques.

3.5 Recurrent Proximal Policy Optimization

Although PPO has gained prominence due to its ability to effectively optimize policies for sequential decision-making tasks, it still may fall short in capturing underlying dynamics in environments where temporal dependencies and sequential patterns are prevalent. Recurrent PPO is an extension of PPO that incorporates Recurrent Neural Network (RNN)s (also known as Long Short-Term Memory (LSTM) networks) into the policy network architecture.

3.5.1 Recurrent Neural Networks

RNNs are a class of neural networks well-suited for processing sequential data. Unlike traditional feedforward networks, RNNs possess a hidden state that maintains memory

of past inputs, enabling them to capture temporal dependencies. In Recurrent PPO, the policy network is transformed from a conventional feedforward structure to an RNN-based architecture. The RNN takes both the current state and the previous hidden state as inputs and produces a probability distribution over actions as output.

Mathematically, the hidden state update in the RNN can be described as follows:

$$h_{t+1} = f(h_t, s_t) \tag{3.9}$$

where h_{t+1} represents the updated hidden state at time $t+1$, f is the recurrent function capturing the temporal relationships, and s_t denotes the current state at time t.

3.5.2 Training Procedure

The training procedure of Recurrent PPO follows the fundamental principles of the original PPO algorithm. The primary distinction lies in the utilization of the RNN-based policy network. The surrogate objective function remains a cornerstone of optimization, and clipped policy updates continue to be employed to ensure stability during training. The modified policy network enhances the agent's ability to capture sequential dependencies within the observed data.

An essential feature of Recurrent PPO is its ability to handle variable-length sequences of observations, a characteristic particularly useful in tasks with sequential data. The recurrent architecture empowers the agent to effectively process and interpret data points in a sequential manner. This adaptability is especially advantageous in scenarios such as time-series analysis or tasks with partial observability.

Mathematically, the recurrent state update equation (equation 3.9) enables the agent to maintain contextual information across time steps, ensuring that the agent benefits from historical observations.

In conclusion, Recurrent PPO presents a valuable extension of the PPO algorithm, introducing RNNs to handle sequential data in RL. The integration of RNNs into the policy network architecture equips the agent with the ability to model temporal dependencies and effectively process variable-length sequences. Through the utilization of a clipped surrogate objective function and an enhanced policy network, Recurrent PPO strikes a balance between exploration and exploitation while accommodating the demands of temporal data. The algorithm's potential impact spans across various applications, demonstrating its significance in advancing the capabilities of RL in dynamic and sequential contexts.

3.6 Soft Actor Critic

The SAC algorithm represents a notable advancement in the realm of RL, particularly within the domain of model-free methods. Similar to its counterparts, SAC employs actor and critic networks to facilitate optimal policy learning. However, what sets SAC apart is its distinctive approach to policy optimization, where the maximization of both expected cumulative reward and entropy are paramount.

3.6.1 Entropy Maximization and Stochastic Policies

In contrast to DDPG, SAC introduces an essential modification in its policy optimization objective. SAC explicitly aims to maximize the entropy of the policy distribution. Entropy serves as a measure of the randomness or uncertainty in the policy's action selection, effectively quantifying the diversity of actions chosen in a given state.

The key motivation behind maximizing entropy is twofold. First, it encourages the policy to explore a wider range of actions, which is particularly valuable in environments with multiple optimal solutions or where exploration is crucial. Second, the entropy term contributes to a more fine-grained trade-off between exploration and exploitation, allowing the algorithm to adapt its exploration strategy dynamically based on the current state.

Mathematically, the entropy of the policy distribution π is defined by the expression:

$$\mathcal{H}(\pi) = -\mathbb{E}_{a\sim\pi(a|s)}\left[\log \pi(a|s)\right] \tag{3.10}$$

Here, $\pi(a|s)$ represents the probability of selecting action a_t in state s_t according to the policy network π.

3.6.2 Value Networks and Entropy Regularization

SAC introduces another innovation by adopting stochastic policies, which means that the policy network outputs a probability distribution over actions, rather than deterministically selecting a single action. This stochastic nature contributes to the exploration-enhancing properties of the algorithm and aligns well with the goal of entropy maximization.

Furthermore, SAC employs two separate value networks, often referred to as critics, to estimate the state values. These value networks assist in entropy regularization by providing a more comprehensive understanding of the environment's dynamics. By utilizing

two critics, SAC leverages their ensemble predictions to mitigate overestimation bias and enhance the accuracy of value estimation.

The SAC algorithm's distinctive features make it particularly well-suited for scenarios that demand a balance between exploration and exploitation. Applications involving robotic control, continuous control tasks, and scenarios where exploration is vital can greatly benefit from SAC's enhanced exploration capabilities. Furthermore, SAC's adaptability to different levels of exploration and its potential for handling complex action spaces contribute to its effectiveness in real-world applications.

In summary, SAC represents a pioneering advancement in reinforcement learning that prioritizes both expected cumulative reward and entropy maximization. Through the utilization of stochastic policies and the incorporation of two value networks, SAC introduces a nuanced exploration-exploitation trade-off, making it a powerful tool for tackling a diverse range of challenges in autonomous decision-making and control systems.

Table 3.1 shows the comparison of the RL algorithms in this research.

Algorithm	Type	Action Space	Function Approx.	Exploration	Continuous	Temporal
Q-learning	Model-Free	Discrete	No	Epsilon-Greedy	No	No
DQN	Model-Free	Discrete	Yes (DNN)	Epsilon-Greedy	No	No
DDPG	Model-Free	Continuous	Yes (DNN)	Exploration Noise	Yes	No
PPO	Model-Free	Continuous	Yes (DNN)	Clipped Surrogate	Yes	No
Recurrent PPO	Model-Free	Continuous	Yes (RNN)	Clipped Surrogate	Yes	Yes
SAC	Model-Free	Continuous	Yes (DNN)	Entropy Regularization	Yes	No

Table 3.1: Comparison of Reinforcement Learning Algorithms

Chapter 4

High-Fidelity Simulation Environment & Software Architecture

A pivotal cornerstone of this research lies in the creation of a sophisticated simulation environment utilizing a high-fidelity simulation tool. This undertaking plays a vital role in enabling comprehensive comparisons and training of the proposed algorithms. The selection of a high-fidelity environment, encompassing realistic pedestrian and adversarial vehicle behaviors, is paramount for accurate assessment and robust learning.

A notable challenge within the realm of AV control research pertains to the inherent diversity of simulation tools adopted by researchers, often hampering effective result comparisons. Particularly concerning RL algorithms, where agent policies are refined through interactions with the environment, the need for a realistic and intricate simulation setup is of utmost importance.

Addressing these challenges, this research leverages CARLA [19], a widely acclaimed high-fidelity AV simulation tool renowned for its dedication to replicating real-world intricacies. CARLA emulates a meticulously detailed and accurate environment, incorporating physics, sensor models, and graphical attributes. The simulation environment, embracing both pedestrians and vehicles, is constructed using CARLA v0.9.10.1, crafted in alignment with OpenAI's gym format [54]. The containerized solution development ensures that the simulation environment created is independent of CARLA versions, allowing for rapid instantiation of new instances, thereby enhancing adaptability and expediting the setup of the simulation environment.

To imbue the environment with essential variability, a robust randomization strategy is employed. The initiation of the ego vehicle's starting position, alongside the positioning of other vehicles and pedestrians, undergoes randomization. This deliberate randomness augments the richness of the simulation, capturing the stochastic nature inherent in real-world driving scenarios. Apart from crosswalks, pedestrian compliance infrastructure is absent, granting pedestrians unequivocal right of way.

In essence, the development of this high-fidelity simulation environment signifies a pivotal advancement in this research, affording a standardized and realistic platform for algorithm assessment and training, thereby catalyzing meaningful progress in the domain of AV control. This environment has been made open-source and can be found at the following link https://github.com/faizansana/intersection-carla-gym.

4.1 Adversarial Vehicles

The deployment of adversarial vehicles within the simulation environment is strategically orchestrated. Specifically, these adversarial vehicles materialize randomly across the three lanes distinct from the ego vehicle's lane, a design choice harmonizing with the inherent structure of the four-way intersection. Notably, the starting position of each adversarial vehicle is subject to randomized selection within its designated lane.

The dynamic movement of adversarial vehicles is fueled by a randomized route assignment for every instance, a feature meticulously achieved through the fusion of the TTC algorithm and a pair of PID controllers—one dedicated to lateral control and the other steering the longitudinal aspect. To encapsulate a spectrum of driving behaviors, the minimum permissible distance maintained between the adversarial vehicles and other road users is stochastically drawn from a range spanning 5m to 10m. This parameterization ensures that higher values signify a cautious driving approach, in stark contrast to the more assertive demeanor embodied by lower values. Each adversarial vehicle is controlled by CARLA's traffic manager.

4.2 Pedestrians

The intricate integration of pedestrians into the simulation fabric follows a thoughtfully curated methodology. Each crosswalk becomes the stage for a pedestrian's entry, with their

precise emergence points carefully randomized within the crosswalk boundaries. The selection of pedestrians' physical attributes—height and body mass—is inherently stochastic, anchored in pre-established blueprints available within the CARLA library.

The pedestrians' velocity profile adheres to a specified cap of $1.4m/s$, a parameterization designed to emulate a realistic walking pace. The orchestration of pedestrian movements rests upon the capable shoulders of the Unreal Engine's AIController, adept at assimilating environmental perception cues to inform its course of action. Notably, these pedestrians collaborate harmoniously, sharing objectives and knowledge to synchronize their movements and collectively pursue cooperative tasks.

A visual insight into this is shown in Figure 4.1, presenting a top-down view that showcases the interplay between the ego vehicle (in red), adversarial vehicles (in blue), and pedestrians (in green). Each episode ushers in a new symphony of randomness, configuring the positions of pedestrians, adversarial vehicles, and the ego vehicle, culminating in a dynamic tableau of motion and interaction. The route's trajectory is delineated in black, while the ego vehicle's orientation is highlighted by an arrow for visual clarity. This composite portrayal encapsulates the intricate choreography of vehicular and pedestrian dynamics within the simulation milieu.

Table 4.1 shows the various parameters that can be changed within the configuration file and Table 4.2 shows the reward function weights.

Table 4.1: Parameters for Simulation Environment

Parameter	**Type**	**Description**
obs_space	String	Observation space type (`"dict"` or `"normal"`)
continuous	Boolean	Whether the action space is continuous
target_speeds	List[float]	List of discrete target speeds for speed control (m/s)
desired_speed	Float	Desired speed for continuous speed control (m/s)
dt	Float	Time step duration (s)
port	Integer	Port for communication with server
render	Boolean	Whether to render the simulation (`True`/`False`)
ego_vehicle_filter	String	Filter for selecting ego vehicle
num_veh	Integer	Number of adversarial vehicles
num_ped	Integer	Number of pedestrians
max_steps	Integer	Maximum number of simulation steps
CAM_RES	Integer	Camera resolution for rendering
max_waypt	Integer	Maximum number of waypoints in the route
pedestrian_proximity_threshold	Float	Threshold for proximity to pedestrians (m)
vehicle_proximity_threshold	Float	Threshold for proximity to vehicles (m)

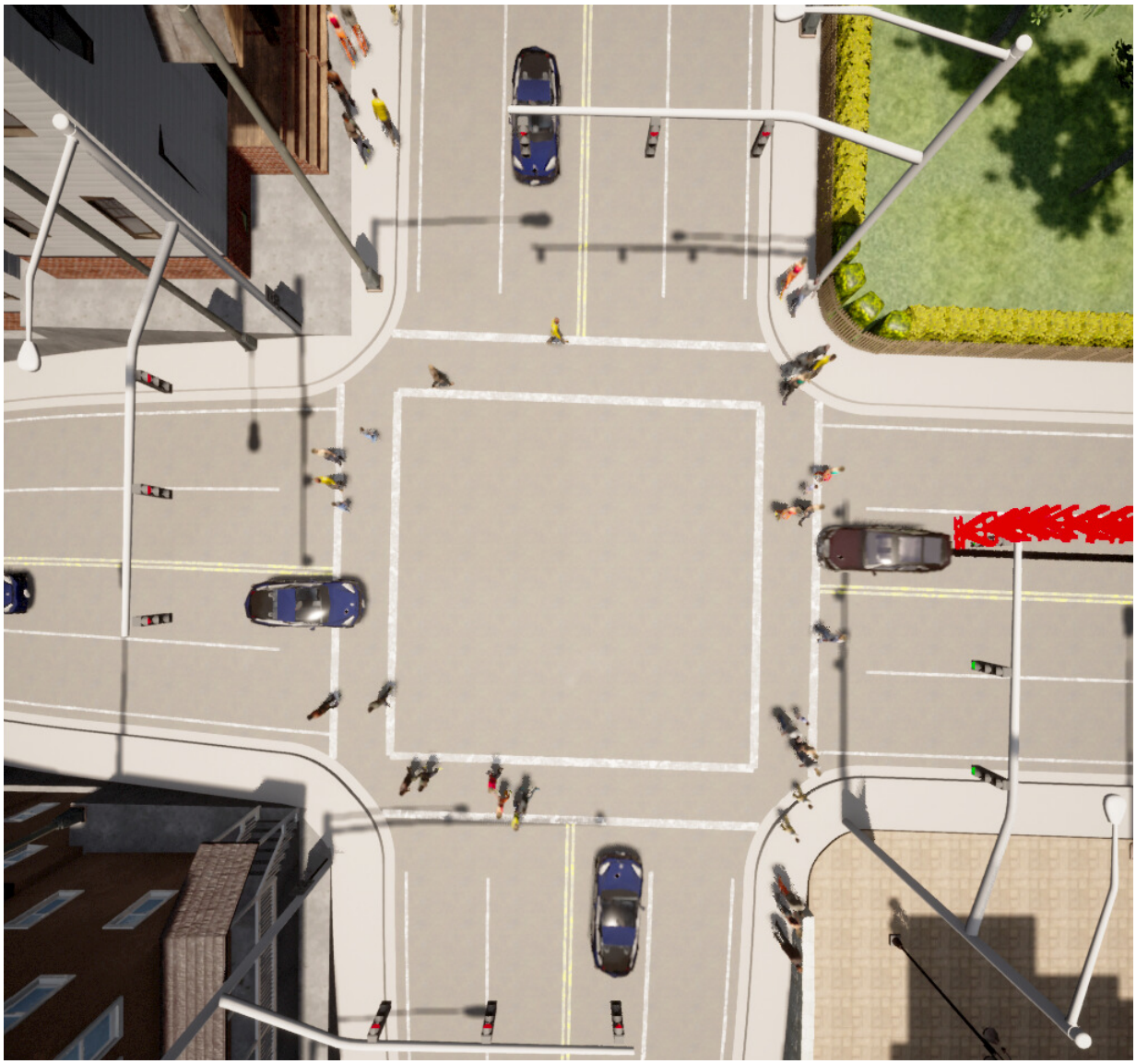

Figure 4.1: Top-down view of the simulation environment consisting of the ego vehicle (in red), adversarial vehicles (in blue) and pedestrians (in green). The locations of the pedestrians, adversarial vehicles, and ego vehicle are randomly initialized for each episode. In this case, 3 adversarial vehicles were spawned in each lane, totaling to 9 other vehicles and 10 pedestrians for each crosswalk totaling to 40 pedestrians. Hence there were a total of 50 traffic participants in this scene including the ego vehicle.

Table 4.2: Reward Weights within Simulation Environment

Reward Component	Description
completion	Route completion reward
terminal_collision	Collision penalty with vehicle
terminal_pedestrian_collision	Collision penalty with pedestrian
terminal_timeout	Timeout penalty
v_eff_under_limit	Velocity reward for being under speed limit
v_eff_over_limit	Velocity penalty for exceeding speed limit
step	Penalty for needing another step
action_reg	Penalty for non-smooth actions
yaw_delta	Penalty for yaw delta w.r.t. road heading
lat_dev	Penalty for lateral deviation
dist_from_goal	Penalty for distance from goal
progress	Progress reward
pedestrian_proximity	Penalty for being close to pedestrians
vehicle_proximity	Penalty for being close to vehicles

4.3 Software Architecture

One of the key works in this research was designing a modular containerized solution with Docker that not only facilitates seamless development but also enables multiple developers, whether on the same machine or different machines, to collaboratively work on and test various software versions with the CARLA backend, ensuring flexibility and efficiency in the development and testing processes. This was done due to the following reasons:

- Conducting simultaneous training for five distinct algorithms necessitates the initiation of five separate CARLA instances. This imperative arises from the need to isolate the internal state of the physics engine within each CARLA instance, ensuring independent and accurate runs for each algorithm.
- Enabling collaborative use of the simulation environment among research team members poses a substantial setup challenge. Without containerization, this involves configuring binary packages, installing CUDA and its dependencies, setting up Python packages, and meticulously following installation instructions for the specific version of CARLA, among other tasks. However, with Docker, all these previously time-consuming and error-prone steps are consolidated into a single, straightforward command, vastly simplifying and expediting the setup process.

- Executing the training process for five algorithms across four configurations, especially in this case, proves to be an exceptionally demanding task, necessitating substantial computational power. To streamline and distribute this intensive workload across multiple server machines with minimal setup effort, the implementation of this architecture becomes indispensable. The architecture chosen enables efficient parallelization and utilization of resources, facilitating the intricate task of training complex algorithms on diverse configurations while optimizing the use of available compute power.

While the comprehensive adoption of a Dockerized setup provides numerous advantages, it does come with a notable drawback – the immediate lack of support for desktop-based visualization. To overcome this limitation, a strategic solution has been implemented by exposing a VNC server, enabling the execution of desktop-based visualization software. This functionality proves invaluable for researchers, as it allows for the visual inspection, debugging, and interactive exploration of the simulation environment. This qualitative observation capability enhances the understanding of the simulated scenarios and facilitates more effective troubleshooting and refinement of the overall system. Figure 4.2 shows a screenshot of the VNC instance that developers can use for qualitative views of the simulation environment.

A detailed illustration of the training process for DRL algorithms within this software architecture is depicted for a singular developer in Figure 4.3. In this particular scenario, where five DRL algorithms undergo training, the initiation of five separate CARLA containers is initiated. These containers establish communication with the main container through TCP/IP, engineered to ensure complete isolation between the CARLA instances. This strategic setup allows for the concurrent training of five DRL algorithms, thereby significantly accelerating the pace of development and testing. The orchestrated interaction among the containers not only optimizes computational resources but also enhances the efficiency of the entire development and testing pipeline. It is also important to note that number of CARLA instances can be easily adjusted based on the requirements by setting a single environment variable. It also allows for switching between various CARLA versions with a singular command.

In scenarios involving multiple developers, an exemplar utilizing a single cloud server is demonstrated in Figure 4.4, wherein multiple developers can instantiate instances of the architecture illustrated in Figure 4.3, ensuring complete isolation. This versatile approach extends beyond cloud servers and can seamlessly be implemented on individual machines or a singular machine, emphasizing the architecture's capability to facilitate complete isolation across diverse development environments.

Figure 4.2: Screeenshot of top down view of the CARLA simulation environment within the container, accessible via a VNC client software.

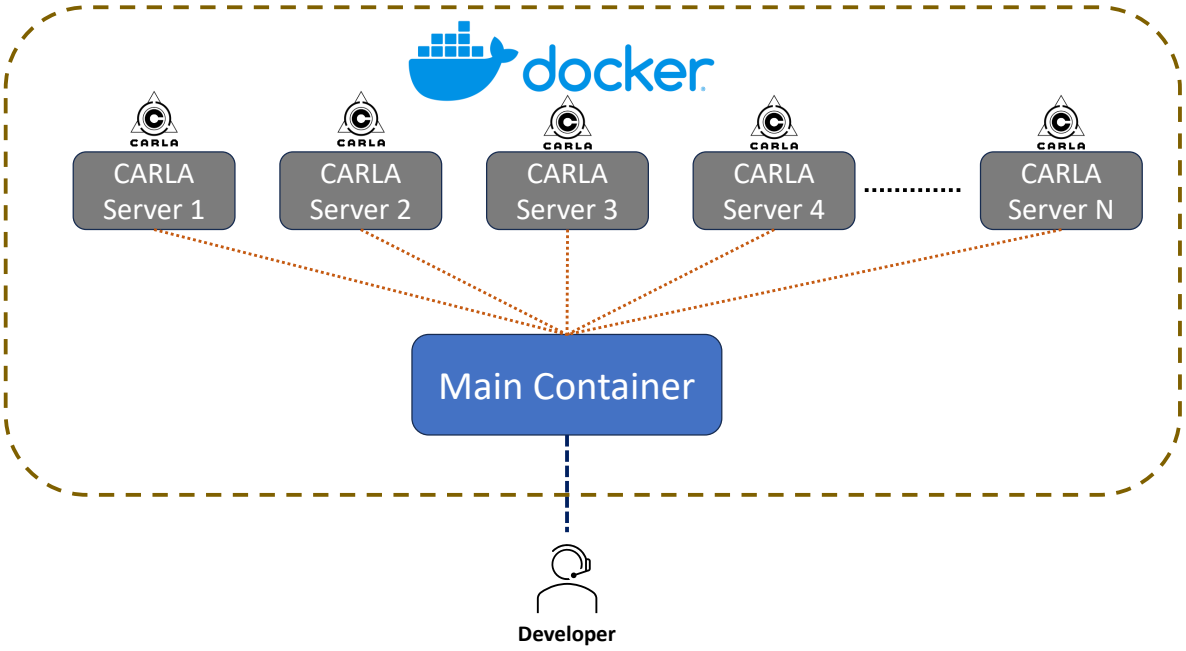

Figure 4.3: Illustration showcasing the Dockerized containers, comprising N instances of CARLA servers alongside a singular main container, with the added capability of establishing a connection to the main container through SSH.

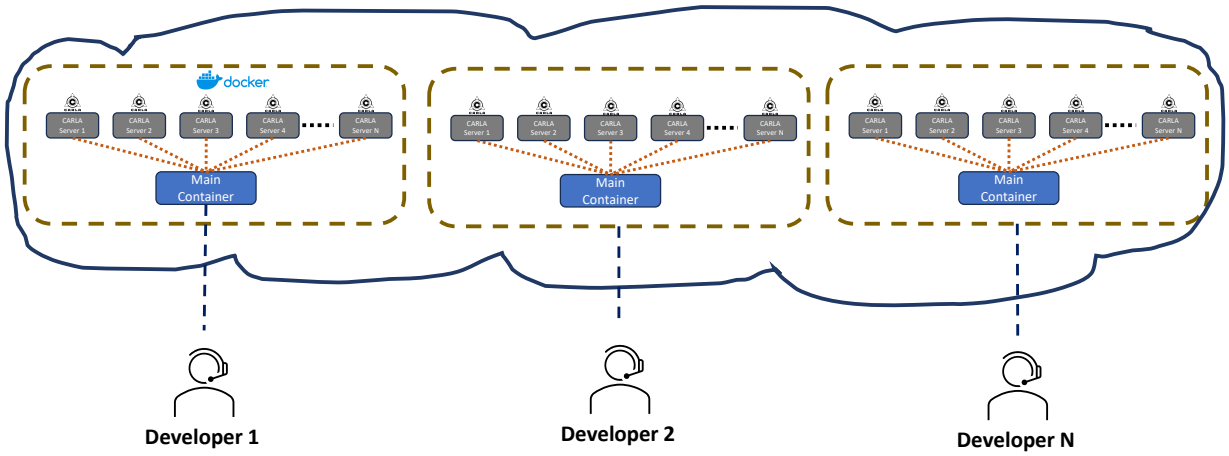

Figure 4.4: An illustrative representation displaying multiple instances of the service shown in Figure 4.3, highlighting the architecture's versatility in being utilized on a single machine. Furthermore, this adaptability extends to replication, allowing instances to operate on individual development machines with seamless integration and robust functionality.

Chapter 5

Methodology

In this chapter, an overview of the problem formulation including the observation and action spaces, algorithm as well as software architecture are presented.

5.1 Problem Formulation

Navigating through an intersection poses a complex challenge for AVs, demanding effective decision-making amidst the uncertainty surrounding the behavior of other road users, including pedestrians and vehicles. To execute junction navigation tasks safely and efficiently, AVs must interact with traffic participants, discern viable gaps, and assertively yield or advance. Balancing the often-conflicting objectives of efficiency, safety, and congestion avoidance adds further intricacy, a challenge not exclusive to human drivers.

In this context, the primary objective of the ego vehicle is to determine an optimal speed profile for traversing a designated route. Achieving this entails generating speed trajectories that adapt to evolving driver intents, intricate interactions, and a landscape of inherent uncertainties.

The intricate nature of these sequential decision-making challenges finds a suitable mathematical representation in MDPs. MDPs offer a formal framework captured by the tuple $\langle S, A, P, R, \gamma \rangle$. Here, S denotes the set of states, A encompasses the available actions for the agent, P encapsulates the state transition probabilities (the transition model), R embodies the reward function, and γ represents the discount factor. MDPs are grounded in the Markov assumption, which stipulates that the probability of transitioning to a new

state relies solely on the present state and action, independent of all prior states and actions. Formally, this is expressed as

$$p(s_{t+1}|s_{0:t}, a_{0:t}) = p(s_{t+1}|s_t, a_t) \tag{5.1}$$

Consequently, the intricate task of traversing an intersection is aptly modeled as an MDP. By encapsulating the interplay of states, actions, uncertainties, and rewards within the MDP framework, the AV's decision-making process becomes amenable to systematic analysis and algorithmic treatment. This approach enables AVs to tackle the multifaceted challenge of intersection decision-making and control while adhering to safety, efficiency, and fluid traffic flow objectives.

5.2 Algorithm Architecture

Since DRL algorithms suffer from sample efficiency issues and require lots of training, a hierarchical approach for AV control is employed. The DRL algorithm is responsible for high-level decision-making, which gives the desired forward speed that provides a setpoint for a (longitudinal) PID controller to output the throttle and brake signals for the longitudinal control. Using the DRL algorithms for selecting the appropriate vehicle speed reduces the sample space size and therefore the algorithms converge faster. In the case of the lateral control, the desired waypoints are ingested by another PID controller. This lateral PID controller outputs the steering angle. Finally, the safety and comfort check limit the maximum throttle, brake, and steering angle based on the following:

- Minimizing jerk to promote passenger comfort.
- Keeping steering changes within 10% of the entire limit.
- Ensuring current road speed limits are followed.

The proposed architecture is shown in Figure 5.1.

It has been found that using a feature extractor module for each actor separately improved performance [55]. Hence, a feature extractor module which consists of a separate DNN for each actor type (adversary vehicles, pedestrians, and the ego vehicle) is used.

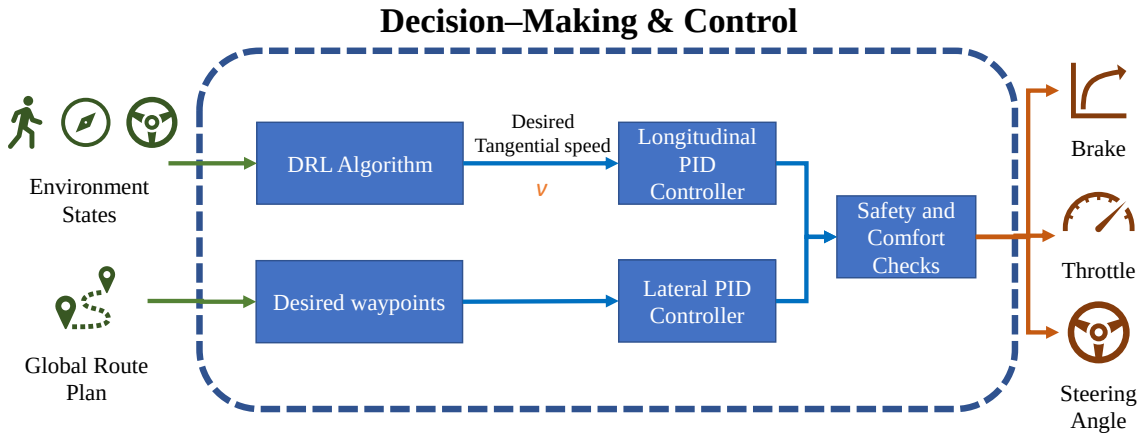

Figure 5.1: Proposed architecture for the ego vehicle decision-making and control.

5.2.1 Feature Extractor Module

A critical aspect of the proposed control architecture involves the integration of a feature extractor module. This module plays a pivotal role in enhancing the decision-making process within the autonomous vehicle's control system. By leveraging individual DNNs tailored for distinct actor types—adversary vehicles, pedestrians, and the ego vehicle—the feature extractor significantly contributes to the system's overall performance and adaptability.

The feature extractor serves as a cognitive filter, dynamically processing raw sensor data and generating informative abstractions for each actor category. Each dedicated DNN within the module specializes in extracting relevant features from the sensor inputs that pertain to a specific type of entity. This adaptive approach ensures that the AV's decision-making process is finely tuned to the characteristics and behaviors of different road users.

By tailoring feature extraction to the distinct characteristics of each actor type, this module empowers the AV's decision-making capabilities. For example, when interacting with adversary vehicles, the module may extract essential features such as relative speed and distance, facilitating informed responses to changing traffic dynamics. Similarly, when dealing with pedestrians, the module can focus on pedestrian movement patterns and potential collision risks. Utilizing separate DNNs for individual actor types introduces a strategic partitioning that capitalizes on the inherent strength of DNNs in handling

specialized and analogous behaviors. It is widely recognized that DNNs exhibit enhanced performance when dedicated to specific, akin scenarios.

5.2.2 PID Controllers for Longitudinal and Lateral Control

For longitudinal control, a PID controller operates in conjunction with the high-level decision-making of the DRL algorithm. The DRL algorithm determines the desired forward speed, which serves as the setpoint for the longitudinal PID controller. By continuously comparing the desired speed with the vehicle's actual speed, the PID controller computes a control signal that efficiently modulates the throttle and brake, ensuring precise speed tracking and optimal acceleration and deceleration profiles.

In lateral control, another PID controller takes center stage to govern the vehicle's steering angle. This controller processes desired waypoints—representing the vehicle's intended path—and continuously adjusts the steering input to accurately follow the desired trajectory. By comparing the vehicle's lateral position to the intended path, the PID controller generates corrective commands that maintain trajectory accuracy, enabling smooth and responsive lane-keeping behavior.

The following Tables 5.1 and 5.2 summarize the proportional (K_p), integral (K_i) and derivative (K_d) gains for both lateral and longitudinal control alongside the maximum limits. Δt was set to 0.05 s. It is important to note that the minimum and maximum values set by CARLA are $[0, 1]$ for throttle, and brake while it is $[-1, 1]$ for steering.

The presented algorithm outlines the control strategy used f

Table 5.1: PID Controller Parameters

Controller	Proportional Gain (K_p)	Derivative Gain (K_d)	Integral Gain (K_i)
Longitudinal	1.0	0.0	0.05
Lateral	1.95	0.2	0.07

5.3 Observation/State Representation

The state space for the ego vehicle and other entities is thoughtfully designed to enable informed decision-making and facilitate safe decision-making and control.

For the ego vehicle, the observation state encompasses the following key variables:

Algorithm 1 Control Strategy

Input: target_speed, waypoint, max_throttle, max_brake, max_steer, past_steering
$acceleration \leftarrow$ Longitudinal PID (target_speed)
$current\_steering \leftarrow$ Lateral PID (waypoint)
if $acceleration \geq 0.0$ **then**
    $throttle \leftarrow \min(acceleration, \text{max\_throttle})$
    $brake \leftarrow 0.0$
else
    $throttle \leftarrow 0.0$
    $brake \leftarrow \min(|acceleration|, \text{max\_brake})$
end if
if $current\_steering > \text{past\_steering} + 0.1$ **then**
    $current\_steering \leftarrow \text{past\_steering} + 0.1$
else if $current\_steering < \text{past\_steering} - 0.1$ **then**
    $current\_steering \leftarrow \text{past\_steering} - 0.1$
end if
if $current\_steering \geq 0$ **then**
    $steering \leftarrow \min(\text{max\_steer}, current\_steering)$
else
    $steering \leftarrow \max(-\text{max\_steer}, current\_steering)$
end if
$\text{past\_steering} \leftarrow steering$
return $current\_steering, throttle, brake$

Table 5.2: PID Controller Limits

Control Variable	Maximum Limits
Throttle	0.75
Brake	0.3
Steering	0.8

- Velocity (v_x, v_y): The ego vehicle's linear velocity components in the longitudinal and lateral directions.
- Acceleration (a_x, a_y): The ego vehicle's acceleration in the longitudinal and lateral directions.
- Heading Angle: The orientation of the ego vehicle with respect to its reference frame.
- Change in Heading Angle ($\Delta\theta$): The rate of change of the ego vehicle's heading angle.
- Angular Velocity (ω): The rotational velocity of the ego vehicle.
- Lateral Distance from Center of Road (d): The lateral offset of the ego vehicle from the center of the road.
- Progress (Distance to Destination (d_{dest})): The distance remaining to the destination, indicating the vehicle's progress.

For adversarial vehicles and pedestrians, the observation space encompasses the following informative attributes:

- Distance in the x-direction to the ego vehicle (x_{rel}): Horizontal distance between the ego vehicle and the observed entity.
- Distance in the y-direction to the ego vehicle (y_{rel}): Vertical distance between the ego vehicle and the observed entity.
- Tangential Velocity (v_{tan}): The component of velocity along the direction tangential to the entity's trajectory.

It is noteworthy that this study assumes the availability of a robust perception system capable of providing the aforementioned state information for other vehicles and pedestrians. Moreover, the study context excludes intersections with stop signs, focusing on intersections where continuous traffic flow is presumed.

5.4 Action Representations

The action representations are formulated for both continuous and discrete action spaces.

5.4.1 Continuous Action Space

The output range is intentionally confined to $[-1, 1]$, aligning with recommendations from existing literature. This symmetrical range is then mapped to the desired speed range $[0, \text{desired speed}]$, where the *desired speed* is user-set and remains adaptable.

5.4.2 Discrete Action Space

Diverging from the conventional dichotomy of *slow* and *go faster* decisions prevalent in the literature, this study employs a nuanced approach by introducing a triad of actions: {*go slower*, *idle*, *go faster*}. This expansion not only aligns with the intricacies of real-world driving scenarios but also contributes to more nuanced and sophisticated decision-making.

The discretization of action space facilitates seamless integration with user preferences and real-time dynamics. The choice of desired speed undergoes a meticulous process, where it is adjusted to the nearest feasible value within a discretized range of speed increments ($[0, 3, 6, 9, 12]$ m/s). For instance, upon selecting the *go faster* decision with an existing desired speed of 3 m/s, the desired speed would be smoothly adjusted to 6 m/s. Analogously, the *idle* decision maintains the existing desired speed, while *go slower* results in a gradual reduction by shifting leftward within the discretized speed options. This nuanced approach not only promotes smoother acceleration and deceleration but also mitigates abrupt and discomforting transitions alongside the PID controllers, ensuring a harmonious AV-user interaction.

It's important to emphasize that the incorporation of the desired speed parameter in both discrete and continuous action spaces introduces an element of personalization, elevating the user experience by offering the flexibility to tailor the AV's speed characteristics according to individual preferences. The dynamic nature of the desired speed parameter enables continual adaptation, even post-training, reflecting the study's commitment to user-centric AV control.

5.5 Reward Function

Developing an appropriate reward function stands as one of the paramount challenges in the realm of RL. The meticulous design of this function serves the dual purpose of preventing collisions while simultaneously fostering efficient, high-speed decision-making and control. The reward function comprises a blend of sparse and dense components, each strategically calibrated to incentivize safe, goal-oriented AV behavior. Sparse components represent the rewards given at the end of the episode while dense rewards signify those rewards given at each timestep. The reward function $R(s, a) \in \mathbb{R}$ is:

$$R(s,a) = R_{seff} + R_{dg} + R_{pcl} + R_{advcl} + R_{goal} + R_{tout} + R_{advcol} + R_{pedcol} \tag{5.2}$$

where R_{seff}, R_{dg}, R_{pcl}, R_{advcl}, R_{goal}, R_{tout}, R_{advcol}, R_{pedcol} are the rewards based on ego vehicle speed efficiency, distance to the goal, pedestrian closeness, adversarial vehicle closeness, reaching the goal, timeout, adversarial vehicle collision, and pedestrian collision, respectively.

5.5.1 Dense Components

The continuous nature of the dense components contributes to fine-grained, step-by-step guidance towards goal attainment.

$$R_{seff} = \begin{cases} r_1 \cdot (v_{\text{desired}} - v_{\text{current}}) & \text{if } v > v_{\text{limit}} \\ v_{\text{current}} \end{cases}$$

$$R_{dg} = r_2 \cdot (-1 + \frac{\text{index of current waypoint}}{\text{len(waypoints)}})$$

$$R_{pcl} = \begin{cases} r_3 \cdot (d_{\text{prox-ped}} - d_{\text{ped}}) & \text{if } d_{\text{ped}} < d_{\text{prox-ped}} \\ 0, & \text{otherwise} \end{cases}$$

$$R_{advcl} = \begin{cases} r_4 \cdot (d_{\text{prox-adv}} - d_{\text{adv}}) & \text{if } d_{\text{adv}} < d_{\text{prox-adv}} \\ 0 & \text{otherwise} \end{cases}$$

Moreover, safety considerations play an integral role in shaping the AV's decision-making. The Euclidean distances between the leading edge of the ego vehicle's bounding box and other pedestrians and adversarial vehicles are continuously assessed. A safety buffer of 2.5m is established for vehicles, and 2m for pedestrians, with negative penalties of -5 and -10 imposed respectively in case of proximity.

To refine the learning dynamics, reward weights are normalized to reside within the interval of [-1, 1]. This normalization procedure contributes to stable and effective learning, aligning with the overarching goal of achieving robust, user-centric AV control.

5.5.2 Sparse Components

The sparse reward components are formulated as follows:

$$
\begin{aligned}
R_{goal} &= r_5 \cdot \mathbf{1}(dis_{\text{goal}} < \delta) \\
R_{tout} &= r_6 \cdot \mathbf{1}(t > 500) \\
R_{advcol} &= r_7 \cdot \mathbf{1}(\text{collision}_{\text{adv}}) \\
R_{pedcol} &= r_8 \cdot \mathbf{1}(\text{collision}_{\text{ped}})
\end{aligned}
$$

where $r_i, I \in \{1, ..., 8\}$, $d_{prox-ped}$ and $d_{prox-adv}$ are constants, d_{ped} and d_{adv} are the Euclidean distance from the front center of the ego vehicle to the closest pedestrian and adversarial vehicle, respectively and $\mathbf{1}(\cdot)$ represents the indicator function which is true when the corresponding condition $(\cdot)$ is true. Table 5.3 lists these constants.

When the agent successfully attains the goal, a substantial positive reward of 100 is bestowed. In scenarios of timeouts or collisions, the agent faces punitive measures, incurring negative rewards of -10 and -100, respectively. A heightened negative penalty of -200 is assigned in case of pedestrian collisions. This configuration not only reinforces goal achievement but also emphasizes collision avoidance as a critical priority.

Table 5.3: Values of Reward Constants

Variable	Value
r_1	-2
r_2	3.5
r_3	-10
r_4	-5
r_5	100
r_6	-10
r_7	-100
r_8	-200
$d_{\text{prox-ped}}$	2
$d_{\text{prox-adv}}$	2.5

5.6 Hyperparameters

To ensure a rigorous and equitable evaluation, a consistent neural network architecture is adopted across all cases, featuring a fully connected layer comprising 400 neurons, followed by an additional 300 neurons layer activated by the Rectified Linear Unit (ReLU) activation function. This was only different in the case of Recurrent PPO in which an LSTM network was used.

Table 5.4 comprehensively presents the harmonized training hyperparameters employed for the diverse DRL architectures under scrutiny. While the majority of these hyperparameters are unified across the architectures, it's noteworthy that certain parameters are architecture-specific, accounting for the nuanced variations inherent in each DRL configuration.

Table 5.4: Training Hyperparameters for the DRL Architectures

Hyperparameter	**DQN**	**DDPG**	**SAC**	**PPO**	**Recurrent PPO**
Batch Size	32	100	256	64	128
Buffer Size	100,000			-	-
Discount Factor (γ)	0.98				
Gradient Steps	Same as Environment			-	-
Learning Rate	0.0001				
Learning Starts (α)	1000			-	-
Training Frequency	Every Episode			-	-

5.7 Strategic Training: Unprotected Left Turns and Comprehensive Intersection Scenarios

The DRL algorithms were trained in two different goal settings:

1. **Specialized Training for Unprotected Left Turns.** In this particular training scenario, the agents underwent focused training exclusively for executing unprotected left turns. Notably, the training objectives consistently directed the agents towards a

predefined destination waypoint positioned on the left side. This specialized training approach aimed to enhance the agents' proficiency in safely and effectively executing the complex maneuver of unprotected left turns, addressing the intricacies associated with varying traffic densities, speeds, and potential obstacles. It's worth noting that such a focused training approach may result in agents that excel at unprotected left turns but might not perform as well in other driving scenarios.

2. **Comprehensive Training in All Intersection Scenarios.** In this scenario, the agent undergoes extensive training encompassing a wide array of intersection scenarios, ranging from left, straight, to right turns. This training regimen has been implemented to facilitate a more thorough and comprehensive learning experience, enhancing the agent's ability to generalize effectively. The ultimate goal is to equip the agent with the proficiency required to navigate seamlessly in diverse real-world scenarios.

For both scenarios, the algorithms were trained within an environment of exclusively vehicles and one which consisted of both vehicles and four pedestrians. Hence, a total of four different configurations were run. The following Table 5.5 demonstrates this:

Table 5.5: The different environmental configurations used to train the DRL algorithms.

Intersection Goal	**Number of Pedestrians**	**Number of Adversarial Vehicles**
Left turn	0	3 (1 in each lane)
	4 (1 at each crosswalk)	
All scenarios (left turn, right turn, straight)	0	
	4 (1 at each crosswalk)	

Chapter 6

Results

This section presents the results obtained from our simulations for five distinct DRL algorithms: DQN, DDPG, PPO, Recurrent PPO, and SAC algorithms. In the context of our experiments, it is important to note that PPO and Recurrent PPO, which can perform in continuous and discrete action spaces, are exclusively trained using discrete action spaces. The algorithms are trained for 1 million time steps corresponding to approximately 6,000 episodes. The choice of using time steps for continuous control tasks ensures a standardized benchmark, despite variations in episode durations. The algorithms underwent training in two distinct environments: the first exclusively comprised instances of left turns, whereas the second encompassed the entirety of intersection scenarios, including straight, left, and right maneuvers.

To establish performance benchmarks, we assessed the algorithm with the highest episode reward every 2000 time steps and saved it for reference. For the comparative analysis of training performance, both the episode mean reward per timestep and the episode length per timestep graphs are used. A smoothing factor of 0.99 is applied to enhance trend identification in conjunction with the raw graphs. This smoothing factor aids in providing a clearer representation of the underlying trends by reducing noise and highlighting the overall trajectory of the training process.

6.1 Distributed Training Environment and Machine Specifications

To expedite the training process, a distributed approach utilizing multiple machines was implemented. This orchestration was made possible by the adoption of a containerized software architecture, as detailed in Chapter 4. The primary machine, whose specifications are predominantly outlined in Table 6.1, assumed the central role during the majority of the training sessions. Two additional machines, mirroring the configuration summarized in Table 6.2, complemented the main server, primarily serving as dedicated environments for testing the trained algorithms.

The main server, running Ubuntu 22.04.3 LTS, boasts an AMD Ryzen Threadripper PRO 3995WX with 64 cores and 120 logical processors. Accompanied by 442 GiB RAM, this powerful server integrates multiple NVIDIA Graphical Processing Unit (GPU)s, including GeForce RTX 4090 and RTX 3090 models, each equipped with substantial Video RAM (VRAM) capacities.

Detailed specifications for the primary server are encapsulated in Table 6.1, providing a comprehensive overview of its Operating System (OS), Control Processing Unit (CPU) architecture, logical processors, RAM, and GPU configurations.

Table 6.1: System specification for main server machine.

Property	**Details**
OS	Ubuntu 22.04.3 LTS
CPU	AMD Ryzen Threadripper PRO 3995WX 64-Cores
Logical Processors	120
RAM	442 GiB
GPUs	NVIDIA GeForce RTX 4090 (24564 MiB VRAM)
	NVIDIA GeForce RTX 3090 (24576 MiB VRAM)
	NVIDIA GeForce RTX 4090 (24564 MiB VRAM)
	NVIDIA GeForce RTX 3090 (24576 MiB VRAM)
	NVIDIA GeForce RTX 3090 (24576 MiB VRAM)
	NVIDIA GeForce RTX 3090 (24576 MiB VRAM)

For local development purposes, two machines, also operating on Ubuntu 22.04.3 LTS, house 13th Gen Intel Core i7-13700K CPUs, 24 logical processors, and 32 GiB of RAM. The GPU on these machines, an NVIDIA GeForce RTX 4080, boasts 16376 MiB of VRAM.

These specifications are encapsulated in Table 6.2 and were primarily employed for algorithm testing during the developmental stages.

Table 6.2: System specification for local development machines.

Property	**Details**
OS	Ubuntu 22.04.3 LTS
CPU	13th Gen Intel(R) Core(TM) i7-13700K
Logical Processors	24
RAM	32 GiB
GPU	NVIDIA GeForce RTX 4080 (16376 MiB VRAM)

On the system specified in Table 6.1, each algorithm requires an approximate training duration of 12 hours. Subsequent testing adds an additional 2 hours to the process. To demonstrate the computational complexity, the total time commitment would be 70 hours if all of the algorithms were to be trained and tested one after the other. This is equivalent to about three days, highlighting the importance of distributed computing and parallelization in accelerating the entire process of testing and assessment.

6.2 Training on Unprotected Left Turns Only

In this scenario, the goal was set to be left turns only and the agent was only trained for that case. A thorough analysis of Figure 6.1 reveals that all algorithms converge after approximately 600,000 time steps. However, an interesting observation is that DDPG may benefit from further training since the mean episode reward demonstrates an ascending trend. In terms of average episode length, DQN, PPO, and Recurrent PPO consistently maintain an average duration of around 130 time steps. In contrast, DDPG exhibits a slightly longer duration of approximately 150 time steps, while SAC exhibits the longest episode length at around 180 time steps.

6.2.1 With Pedestrians

The training results are shown in Figure 6.1.

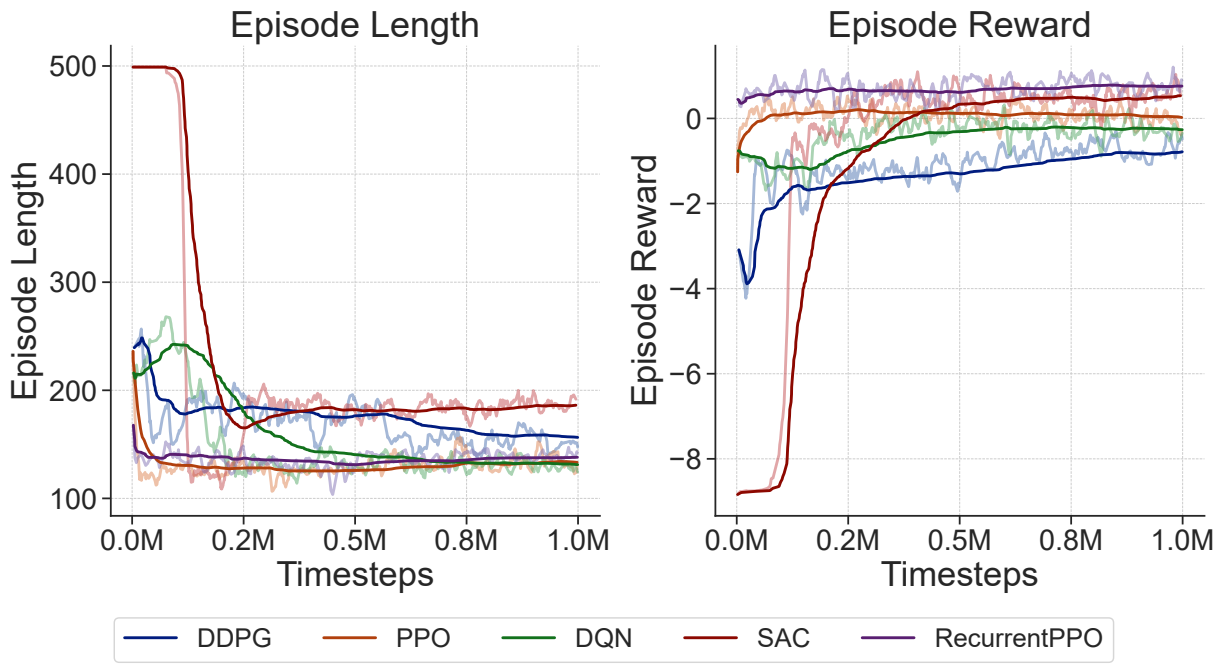

Figure 6.1: Episode mean length and reward during training for *unprotected left turns only* with four pedestrians. Recurrent PPO has the lowest episode mean length while SAC demonstrates the highest. Recurrent PPO demonstrates the highest episode mean reward while DDPG demonstrates the lowest. DDPG could potentially benefit from further training since the reward is seen to be increasing.

6.2.2 Without Pedestrians

Training results are shown in Figures 6.2. Similar to the scenario with pedestrians, PPO maintains the distinction of having the lowest episode length, with DDPG recording the highest. An interesting observation is made of a sudden drop in the episode length for the case of PPO and Recurrent PPO which needs further investigation. The episode length remains consistent with that observed during training within an environment featuring pedestrians, as seen in the Episode Length graph.

In terms of the reward, as anticipated, all DRL policies demonstrate significantly higher average rewards per episode when compared to training within environments featuring pedestrians. This aligns with the expectations as the absence of pedestrian interactions makes the navigation task considerably less challenging, leading to improved performance across all algorithms.

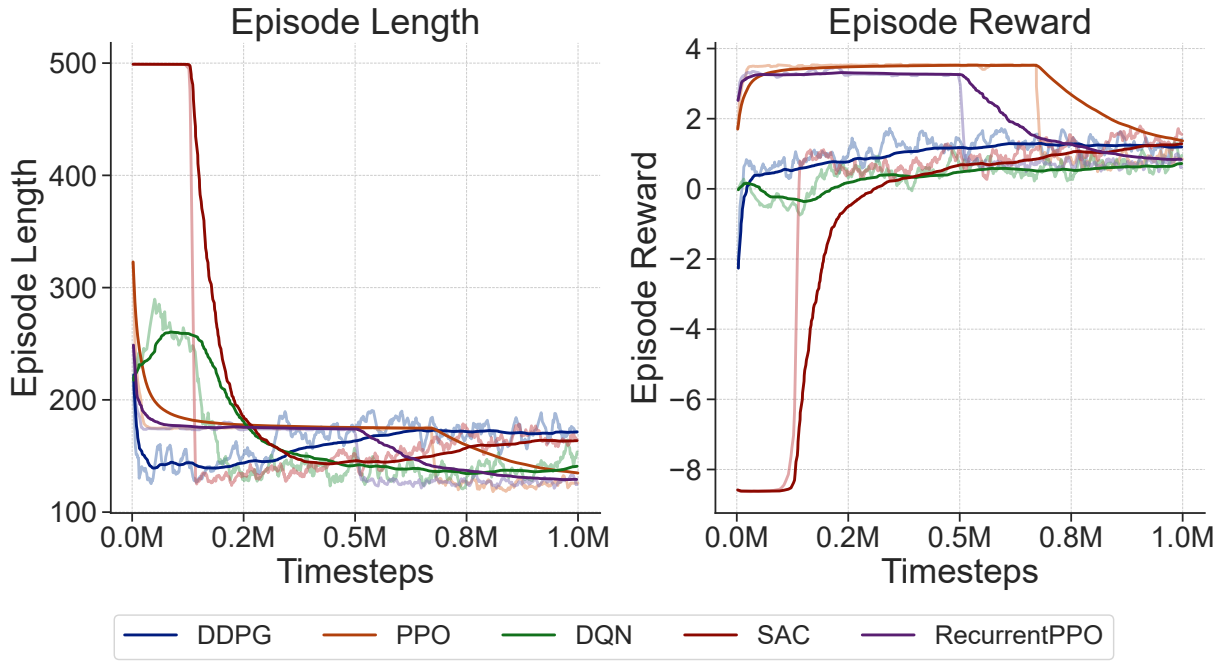

Figure 6.2: Episode length and reward during training for *unprotected left turns only* without pedestrians. Recurrent PPO demonstrates the lowest episode length alongside DQN and PPO while DDPG has the highest. PPO has the highest episode mean reward although there is a significant drop during the training. This is likely due to the retraining when the simulation environment segmentation faults due to a bug in CARLA.

6.3 Training in All Intersection Scenarios

In this case, the agent was trained for all intersection scenarios (right turn, left turn and going straight).

6.3.1 With Pedestrians

Training results are shown in Figure 6.3.

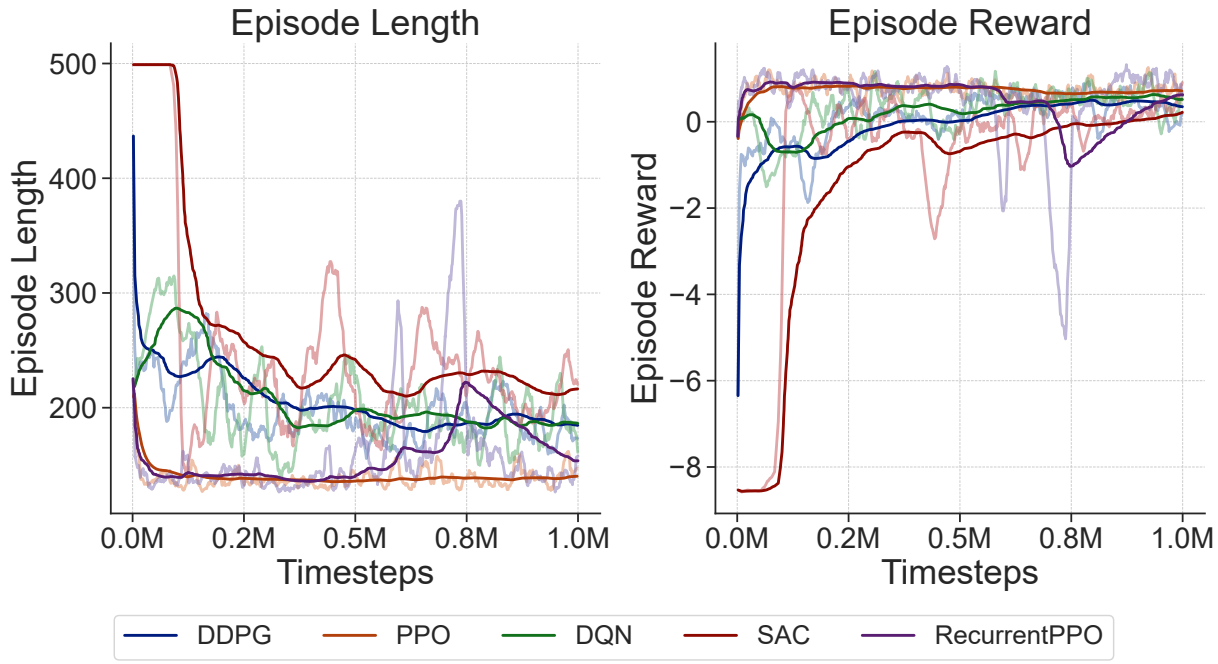

Figure 6.3: Episode length and reward during training for *all intersection scenarios* with four pedestrians. PPO has the lowest episode mean length although, in all algorithms, there seem to be significant oscillations. All algorithms have a similar mean episode reward with SAC potentially benefiting from further training.

As illustrated in Figure 6.3, it is evident that PPO achieves the lowest episode length, while SAC records the highest. A notable observation is the increased variability in episode length compared to the focused training within unprotected left turns with pedestrians, as depicted in Figure 6.1. The raw graphs reveal more pronounced fluctuations, indicating a higher degree of complexity when the agent is exposed to a broader range of intersection

scenarios. Furthermore, it becomes apparent that the episode length is generally longer in the context of training for all intersection scenarios as opposed to the specialized training for unprotected left turns exclusively. This extension in episode length suggests that navigating various intersection scenarios introduces additional challenges, potentially requiring more intricate decision-making processes as well as waiting from the trained agent.

A parallel trend, similar to the one observed during training within unprotected left turns (see Figure 6.1), is evident in the current scenario depicted in Figure 6.3 Episode Reward graph. Notably, SAC appears to exhibit a potential for improvement, as evidenced by the increasing trend in its episode mean reward. This suggests that additional training steps might contribute to further refinement in the performance of the SAC algorithm.

In contrast, Recurrent PPO displays a significant drop in the episode mean reward at around 600,000 timestep, attributed to a restart in training at that specific step due to a CARLA server failure. Despite this setback, the algorithm demonstrates resilience by recovering to its previous performance state. This emphasizes the robustness and adaptability of Recurrent PPO in overcoming unexpected challenges during the training process.

6.3.2 Without Pedestrians

The training results are shown in Figure 6.4 which shows the episode length per timestep and the episode mean reward per timestep.

Figure 6.4 indicates that all algorithms reach convergence at approximately 500,000 timesteps, mirroring the observed convergence patterns in the training scenarios presented in Figure 6.1. Delving into the reward patterns showcased in Figure 6.4, a noteworthy observation emerges: In both training settings, whether with or without pedestrians, Recurrent PPO consistently performs at a comparable level. This suggests a robust adaptability of Recurrent PPO across diverse intersection scenarios.

In contrast, the remaining algorithms exhibit a substantial increase in mean reward per episode. Notably, DDPG stands out with the most significant surge, registering a remarkable 280% increase. This observed augmentation aligns with expectations, considering the reduction in stochasticity within the environment due to the absence of pedestrians. The findings demonstrate the adaptability of continuous DRL algorithms to such variations in the training environment.

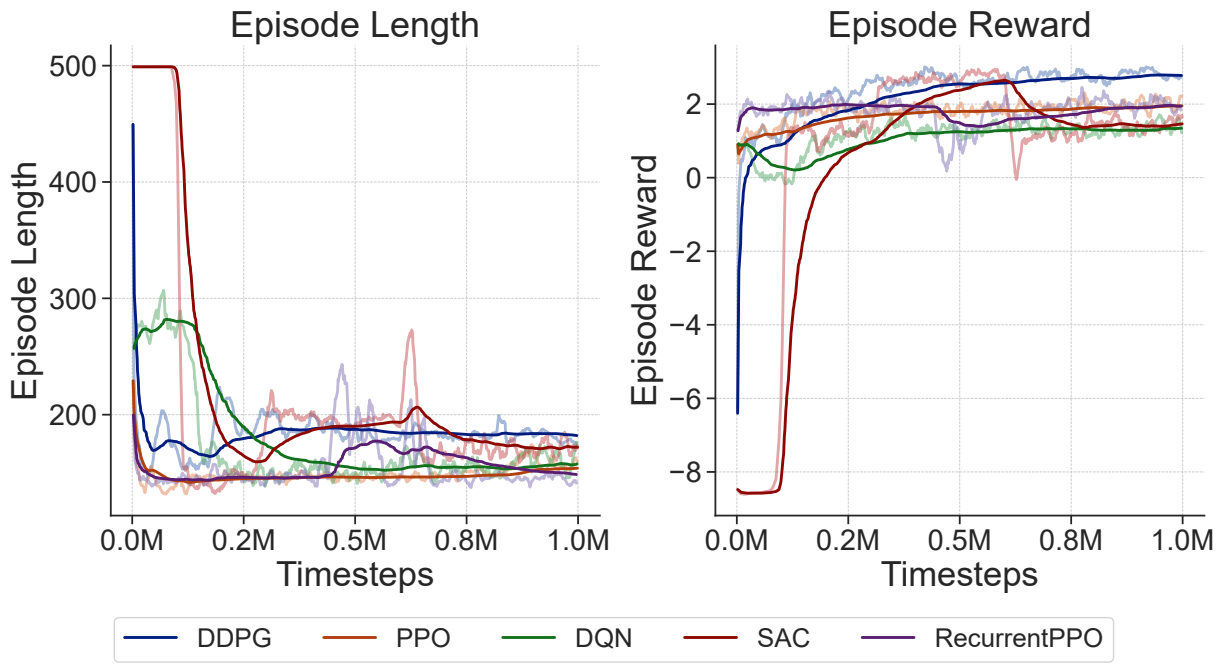

Figure 6.4: Episode length and reward during training for *all intersection scenarios* without pedestrians. Recurrent PPO has the lowest episode length with all algorithms having a mean episode length less than the one observed in the case when training with pedestrians (see Figure 6.3). The mean episode reward is highest for DDPG with DQN being the lowest. As compared to Figure 6.3, the reward is higher for all algorithms due to the absence of pedestrians, as expected.

6.4 Evaluation Metrics

Five metrics are used to evaluate each of these methods, which are obtained by running each DRL method for 1,000 episodes. The metrics are described as follows:

- **Average Episode Length**: how long each successful trial runs for in terms of episode time steps.
- **Average Reward**: the amount of reward obtained by the policy in each episode.
- **Collision Rate**: the percentage of episodes that result in a collision occurring due to the ego vehicle.
- **Pedestrian Collision Rate**: the percentage of accidents that consist of a pedestrian collision.
- **Success Rate**: the percentage of the runs where the ego vehicle successfully reaches the goal. In all cases, the goal is to make an unprotected left turn.

The metrics are only evaluated on the scenarios trained *with* pedestrians. It is important to note that if an episode timed out, it was not counted as part of the metrics and hence some of the collision rates when combined with the success rates fall short of 100%.

6.4.1 Training on Unprotected Left Turns Only

The defined evaluation metrics are shown in Table 6.3, obtained by running the DRL algorithms on 1,000 episodes.

Table 6.3: Comparison of the DRL algorithms within the hierarchical approach when trained for *unprotected left turns only* with four pedestrians.

Metric	**DQN**	**DDPG**	**SAC**	**PPO**	**Recurrent PPO**
Average Episode Length (time steps)	143.45	166.04	161.47	144.87	**122.33**
Average Reward (per episode)	0.7567	0.5811	0.4581	0.7001	**1.085**
Collision Rate	53.5%	52.8%	53.1%	54.1%	**33.6%**
Pedestrian Collision Rate	93.0%	92.0%	93.0%	93.0%	**46.0%**
Success Rate	46.2%	47.1%	46.7%	45.9%	**66.1%**

As seen in Table 6.3, DDPG has the highest average episode length, while Recurrent PPO has the lowest collision rate, pedestrian collision rate, and average episode length, along with having the highest success rate and average reward. However, it is seen that all the methods have relatively high collision rates, particularly concerning pedestrian collision rates. Except for Recurrent PPO, it is observed that over 90% of the accidents consist of a pedestrian collision. This is likely due to the stochastic nature of the pedestrians and needs further investigations into other possible reasons behind such high collision rates. Also, an interesting observation is that all policies demonstrate human-like behavior wherein the ego vehicle creeps, causing the pedestrian to also slightly slow down, wait for the adversarial vehicle within the intersection to pass, and move up slowly to indicate (or "interact") with the adversary vehicle and pedestrian to signal that it is going to pass. It's important to highlight that the majority of collisions involve pedestrians.

It is important to note that when testing the algorithms within no pedestrian scenarios, they demonstrate a success rate of close to 100% as shown in Table 6.4. This further demonstrates the complexity of the environment with pedestrians. In this case, DDPG and SAC appear to be the top-performing algorithms in terms of safety (lowest collision rates) and reliability (highest success rates). However, Recurrent PPO outperforms others in terms of average reward per episode, suggesting potentially more efficient decision-making or better optimization of the learning process.

Table 6.4: Comparison of the DRL algorithms within the hierarchical approach when trained for *unprotected left turns only* with no pedestrians.

Metric	**DQN**	**DDPG**	**SAC**	**PPO**	**Recurrent PPO**
Average Episode Length (time steps)	181.12	174.42	222.22	**174.10**	187.51
Average Reward (per episode)	2.8654	3.2432	2.9679	3.3145	**3.3589**
Collision Rate	1.4%	**0.0%**	**0.0%**	0.1%	1.3%
Pedestrian Collision Rate	-	-	-	-	-
Success Rate	98.6%	**100.0%**	**100.0%**	99.9%	98.7%

6.4.2 Training on All Intersection Scenarios

The evaluation metrics for all five DRL algorithms are presented in Table 6.5. Notably, while these algorithms undergo training for various intersection scenarios, for the purpose of test evaluation, they are exclusively assessed in the context of making unprotected left turns. This ensures a consistent and comparable evaluation setting, allowing for a

focused analysis of each algorithm's performance specifically within the unprotected left turn scenario.

Table 6.5: Comparison of the DRL algorithms within the hierarchical approach when trained for *all intersection scenarios* with four pedestrians.

Metric	**DQN**	**DDPG**	**SAC**	**PPO**	**Recurrent PPO**
Average Episode Length (time steps)	171.26	172.30	184.62	**142.14**	145.33
Average Reward (per episode)	-0.3477	0.9022	0.7561	0.4529	**1.155**
Collision Rate	70.9%	51.7%	49.8%	70.3%	**31.4%**
Pedestrian Collision Rate	**41.0%**	92.0%	94.0%	**41.0%**	42.0%
Success Rate	21.6%	48.3%	49.0%	29.5%	**6 8.1%**

As illustrated in Table 6.5, the performance metrics reveal that PPO stands out with the lowest episode length, while Recurrent PPO, similar to its efficacy in exclusive training for unprotected left turns, exhibits the most resilient performance across scenarios. A noteworthy observation emerges when contrasting these results with the unprotected left turns scenario (refer to Table 6.3): there is an approximate 4% increase in the average episode length across all algorithms. This increase can be attributed to a heightened sense of caution adopted by the algorithms.

Moreover, a deeper analysis unveils a considerable decline in the success rate of both DQN and PPO in comparison to the unprotected left turns scenario. This decrease is likely attributed to the inaccurate initialization of neural network parameters, a well-documented factor influencing the training dynamics of DRL. This underscores the importance of carefully selecting a predetermined seed number for consistent initialization across all DRL algorithms during training.

Chapter 7

Conclusion and Future Work

This chapter concludes by summarizing the work and contributions presented within this thesis as well as discussing future directions. The main focus of this thesis was demonstrating the complexity of pedestrian environments and development of a hierarchical DRL-based approach for AVs to traverse unsignalized intersections using state-of-the-art DRL algorithms in continuous and discrete action spaces. This study employed five DRL algorithms as part of the hierarchical approach: DDPG, DQN, PPO, Recurrent PPO and SAC.

7.1 Conclusion

The hierarchical approach involved two separate controllers for lateral and longitudinal control wherein the DRL was employed within longitudinal control. The DRL algorithms outputted the desired speed while a low-level PID controller, followed by a safety and comfort module, outputted the low-level control commands. The safety and comfort module limited the acceleration and steering rate to avoid jerks and rapid directional changes. A custom DNN based feature extractor was used to extract the pedestrian, adversarial vehicles, and ego vehicle observation states individually, which were then used by the considered DRL algorithms to output the associated longitudinal action. The lateral control was also performed by a PID controller based on the physical constraints of the AV and the final destination given by the ego vehicle's global planner. In the case of the discrete action space, a list of discrete speeds was provided to the considered algorithms, which would then shift through the decision made compared to the traditional stop-and-go methods in the literature. It is important to note that all networks were trained in an end-to-end

manner in four different configurations. It was found that the proposed decision-making and control architecture with Recurrent PPO had the highest success rate of 68.1% and the lowest pedestrian collision rate of 42.0% of collisions. It was observed that maneuvering through chaotic environments posed a significant challenge, with over 90% of collisions involving pedestrians. With no pedestrians, the success rate of all DRL algorithms was close to 100%. Notably, the Recurrent PPO approach emerged as the most time-efficient, requiring the fewest time steps to reach the destination when trained exclusively for unprotected left turns. When trained for all intersection scenarios, most DRL algorithms showed a slight increase in performance metrics with the exception of DQN and PPO. This was likely due to neural network parameter initialization which is random between runs. The hierarchical algorithm with the training, test and analysis scripts has been made available at https://github.com/faizansana/intersection-driving.

A key contribution of this research is the environment used for the training and testing, which consisted of both pedestrians and adversarial vehicles in a complex, high-fidelity simulation tool, CARLA. Notably, the flexibility of adjusting the number of adversarial vehicles and pedestrians facilitates seamless customization for coherent training and testing procedures. This adaptable environment accommodates both discrete and continuous action spaces, presenting observations in the standardized OpenAI Gym format. Furthermore, it supports the training of agents in specific scenarios, encompassing straight, left, or right turns, and their combinations. For accessibility and collaboration, the developed environment is open source and available at https://github.com/faizansana/intersection-carla-gym.

To foster collaboration and streamline development processes, a containerized software architecture using Docker was crafted. This architecture was designed to expedite the setup of development environments with minimal effort, promoting efficient collaboration among team members. This can be found within the GitHub repositories.

7.2 Future Work

To further improve the accuracy of the proposed framework in this study, it is suggested to do the following:

i) Fine-tune the hyperparameters to optimize the model's performance and adaptability.

ii) Explore the application of curriculum learning techniques to progressively improve the obtained reward, potentially accelerating the learning process.

iii) Evaluate the robustness of the considered algorithms by testing their performance under conditions of perception noise, ensuring resilience in real-world scenarios.

iv) Iteratively enhance the design of the reward function, aligning it more precisely with the desired objectives.

A comprehensive investigation of the elevated pedestrian collision rates is also essential. Since the ultimate goal of this research is to develop a suitable navigation algorithm for an experimental AV, the Autonomous Golf Cart (AGC), it was important to perform the training in a high-fidelity simulation environment like CARLA. The future steps include:

i) Incorporating swarms of pedestrians into the simulation, thereby enhancing the realism and complexity of the environment.

ii) Implementing a robust safety layer within the considered algorithms to fortify the system against unforeseen challenges and potential risks.

iii) Exploring the potential integration of model-based controllers such as MPC to augment the precision and adaptability of the autonomous agent.

iv) Broadening the scope of training and testing by exposing the agent to diverse environmental conditions, including scenarios involving rain, which introduces variations in the coefficient of road friction.

v) Assessing the performance of the autonomous agent in real-world scenarios, specifically on the AGC, to validate its adaptability and effectiveness in practical applications.

vi) Enhancing test metrics with the utilization of near-collisions to assess safety concerns for vulnerable road users.

vii) Expanding the study to train and test the performance within four-way intersections with different travel lanes

References

[1] F. Sana, N. L. Azad, and K. Raahemifar, "Autonomous vehicle decision-making and control in complex and unconventional scenarios-a review," *Machines*, vol. 11, no. 7, 2023.

[2] California Department of Motor Vehicles, "Autonomous vehicle collision reports," jun 2022.

[3] S. H. Leilabadi and S. Schmidt, "In-depth analysis of autonomous vehicle collisions in california," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 889–893, IEEE, 2019.

[4] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," standard, Society of Automotive Engineers, 2021.

[5] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[6] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep q-learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3169–3174, IEEE, 2018.

[7] World Health Organization, "Global status report on road safety 2018," tech. rep., World Health Organization, 2018.

[8] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[9] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," tech. rep., National Highway Traffic Safety Administration, 2018.

[10] W. D. Montgomery, R. Mudge, E. L. Groshen, S. Helper, J. P. MacDuffie, and C. Carson, "America's workforce and the self-driving future: Realizing productivity gains and spurring economic growth," 2018.

[11] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[12] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[13] California Department of Motor Vehicles, "Autonomous vehicles testing with a driver," jun 2022.

[14] R. L. McCarthy, "Autonomous vehicle accident data analysis: California ol 316 reports: 2015–2020," *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, vol. 8, no. 3, 2022.

[15] M. Simon, T. Hermitte, and Y. Page, "Intersection road accident causation: A european view," in *21st International Technical Conference on the Enhanced Safety of Vehicles*, pp. 1–10, 2009.

[16] National Highway Traffic Safety Administration, "Traffic safety facts 2019: A compilation of motor vehicle crash data," tech. rep., NHTS Administration, Washington, DC, USA, 2019.

[17] M. Burns, "GM's Cruise Recalls Self-Driving Software Involved in June Crash," *TechCrunch*, 2019.

[18] A. Marshall, "GM's Cruise Recalls Self-Driving Software Involved in June Crash," *Wired*, 2022.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[20] M. S. M. Al-Dabbagh, A. Al-Sherbaz, and S. Turner, "The impact of road intersection topology on traffic congestion in urban cities," in *Proceedings of SAI Intelligent Systems Conference*, pp. 1196–1207, Springer, 2018.

[21] F. Lian, B. Chen, K. Zhang, L. Miao, J. Wu, and S. Luan, "Adaptive traffic signal control algorithms based on probe vehicle data," *Journal of Intelligent Transportation Systems*, vol. 25, no. 1, pp. 41–57, 2021.

[22] T. Wu, P. Zhou, K. Liu, Y. Yuan, X. Wang, H. Huang, and D. O. Wu, "Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8243–8256, 2020.

[23] S. P. Sahu, D. K. Dewangan, A. Agrawal, and T. Sai Priyanka, "Traffic light cycle control using deep reinforcement technique," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pp. 697–702, 2021.

[24] M. A. S. Kamal, J.-i. Imura, T. Hayakawa, A. Ohata, and K. Aihara, "A vehicle-intersection coordination scheme for smooth flows of traffic without using traffic lights," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1136–1147, 2015.

[25] S. Li, K. Shu, C. Chen, and D. Cao, "Planning and decision-making for connected autonomous vehicles at road intersections: A review," *Chinese Journal of Mechanical Engineering*, vol. 34, no. 1, pp. 1–18, 2021.

[26] J. Guanetti, Y. Kim, and F. Borrelli, "Control of connected and automated vehicles: State of the art and future challenges," *Annual reviews in control*, vol. 45, pp. 18–40, 2018.

[27] W. Liu, M. Hua, Z. Deng, Z. Meng, Y. Huang, C. Hu, S. Song, L. Gao, C. Liu, B. Shuai, *et al.*, "A systematic survey of control techniques and applications in connected and automated vehicles," *IEEE Internet of Things Journal*, 2023.

[28] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[29] 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.

[30] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, p. 7166, 2008.

[31] L. Ma, J. Xue, K. Kawabata, J. Zhu, C. Ma, and N. Zheng, "Efficient sampling-based motion planning for on-road autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1961–1976, 2015.

[32] S. Yoon, D. Lee, J. Jung, and D. H. Shim, "Spline-based rrt* using piecewise continuous collision-checking algorithm for car-like vehicles," *Journal of Intelligent & Robotic Systems*, vol. 90, pp. 537–549, 2018.

[33] K. Yang and S. Sukkarieh, "An analytical continuous-curvature path-smoothing algorithm," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[34] X. Wu, A. Nayak, and A. Eskandarian, "Motion planning of autonomous vehicles under dynamic traffic environment in intersections using probabilistic rapidly exploring random tree," *SAE International Journal of Connected and Automated Vehicles*, vol. 4, no. 12-04-04-0029, pp. 383–399, 2021.

[35] Y. Wang, Z. Liu, Z. Zuo, Z. Li, L. Wang, and X. Luo, "Trajectory planning and safety assessment of autonomous vehicles based on motion prediction and model predictive control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8546–8556, 2019.

[36] R. Hult, M. Zanon, S. Gros, and P. Falcone, "Optimal coordination of automated vehicles at intersections: Theory and experiments," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 6, pp. 2510–2525, 2019.

[37] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 174–179, 2017.

[38] J. Moreau, P. Melchior, S. Victor, M. Moze, F. Aioun, and F. Guillemard, "Reactive path planning for autonomous vehicle using bézier curve optimization," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1048–1053, 2019.

[39] D. Kamran, C. F. Lopez, M. Lauer, and C. Stiller, "Risk-aware high-level decisions for automated driving at occluded intersections with reinforcement learning," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1205–1212, IEEE, 2020.

[40] Z. Qiao, K. Muelling, J. M. Dolan, P. Palanisamy, and P. Mudalige, "Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1233–1238, IEEE, 2018.

[41] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

[42] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions.," in *Robotics: Science and Systems*, vol. 2, pp. 1–9, Ann Arbor, MI, USA, 2016.

[43] B. B. Elallid, M. Bagaa, N. Benamar, and N. Mrani, "A reinforcement learning based approach for controlling autonomous vehicles in complex scenarios," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 1358–1364, IEEE, 2023.

[44] R. Bautista-Montesano, R. Galluzzi, K. Ruan, Y. Fu, and X. Di, "Autonomous navigation at unsignalized intersections: A coupled reinforcement learning and model predictive control approach," *Transportation research part C: emerging technologies*, vol. 139, p. 103662, 2022.

[45] T. Tram, I. Batkovic, M. Ali, and J. Sjöberg, "Learning when to drive in intersections by combining reinforcement learning and model predictive control," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3263–3268, IEEE, 2019.

[46] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[47] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep q-learning," in *Learning for dynamics and control*, pp. 486–489, PMLR, 2020.

[48] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[49] E. Debie and K. Shafi, "Implications of the curse of dimensionality for supervised learning classifier systems: theoretical and empirical analyses," *Pattern Analysis and Applications*, vol. 22, pp. 519–536, 2019.

[50] 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.

[51] C.-C. Chang, J. Tsai, J.-H. Lin, and Y.-M. Ooi, "Autonomous driving control using the ddpg and rdpg algorithms," *Applied Sciences*, vol. 11, no. 22, p. 10659, 2021.

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[53] J. Zhang, Z. Zhang, S. Han, and S. Lü, "Proximal policy optimization via enhanced exploration efficiency," *Information Sciences*, vol. 609, pp. 750–765, 2022.

[54] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[55] R. Gutiérrez-Moreno, R. Barea, E. López-Guillén, J. Araluce, and L. M. Bergasa, "Reinforcement learning-based autonomous driving at intersections in carla simulator," *Sensors*, vol. 22, no. 21, p. 8373, 2022.

Glossary

adversarial vehicle Other vehicles present within the ego vehicle's environment. iv, xi, 13, 16, 26–29, 55, 58

CARLA An open source urban driving simulator. See https://carla.org/ for details. iv, xi, xii, 6, 7, 12, 13, 16, 26–28, 30–33, 38, 50, 52, 58, 59

Docker Docker is a container platform that allows for the efficient development, shipping, and running of applications by separating them from the underlying infrastructure. See `https://www.docker.com/` for more details. xi, 30, 31, 33, 58

ego vehicle The vehicle that is the subject of interest which is being controlled through the environment. xi, xii, 11–13, 27–29, 35–38, 40, 55, 57, 66

TCP/IP Transmission Control Protocol/Internet Protocol (TCP/IP) is a suite of protocols that specify communications standards between computers and detail conventions for routing and interconnecting networks. 31

unsignalized intersection A fully uncontrolled intersection devoid of any traffic control devices such as yield signs, stop signs, or traffic lights, where vehicles must navigate based on the right of way and caution. iv, 6, 8, 11, 57