On Differentiable Costs and Local Planning for End-to-End Urban Driving

by

Volkan Aydıngül

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computational Science and Engineering

KOÇ ÜNİVERSİTESİ

August 29, 2023

On Differentiable Costs and Local Planning for End-to-End Urban Driving

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Volkan Aydıngül

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Asst. Prof. Fatma Güney (Advisor)

_____

Asst. Prof. Barış Akgün (Co-Advisor)

_____

Assoc. Prof. Nazım Kemal Üre

_____

Asst. Prof. Erdal Aydın

Date: _____

ABSTRACT

On Differentiable Costs and Local Planning for End-to-End Urban Driving

Volkan Aydıngül

Master of Science in Computational Science and Engineering

August 29, 2023

Self-driving is a challenging task that requires decision-making under uncertainties due to interactions in complex urban environments. We design objectives to provide the driving agent with feedback quantifying the eligibility of the action taken by the agent. For planning with possible actions, previous work introduces a differentiable cost function for highway driving. However, urban driving is more complex than highway driving due to the density of vehicles and scene structures like intersections in urban environments. To this end, we propose a new differentiable cost function that accounts for multiple possible features that frequently occur in urban driving. For future prediction, we propose forward models that predict the future state of the agent and the environment separately. Via separate predictions for the agent (ego) and environment (world), we can integrate domain knowledge into our framework. Empirical results demonstrate that we can reliably predict the future for the ego agent, while the world model exhibits limitations in less frequent cases, such as turning left or right. We further test our differentiable cost function with a gradient-based model predictive control (MPC) and a policy network. Both approaches demonstrate a plausible driving performance in urban driving setup with unique limitations. MPC faces local optima problems due to its sensitivity to the hyper-parameters under different situations, and the policy network depends on supervision from an expert. In conclusion, we design a differentiable cost function for urban driving and show its importance for action as well as limitations for future work to generalize to diverse urban scenarios by building on our work.

ÖZETÇE

Uçtan Uca Şehir İçi Sürüş için Türevlenebilir Maliyetler ve Yerel Planlama Üzerine
Volkan Aydıngül
Hesaplamalı Bilimler ve Mühendislik, Yüksek Lisans
29 Ağustos 2023

Otonom sürüş, karmaşık şehir ortamlarındaki etkileşimler nedeniyle belirsizlik altında karar verme gerektiren zorlu bir görevdir. Sürüş etmenine, etmenin gerçekleştirdiği eylemin uygunluğunu ölçen geri bildirim sağlayacak şekilde hedefler tasarlıyoruz. Önceki çalışmalar, olası eylemlerle planlama yapmak üzere otoyol sürüşü için türevlenebilir bir maliyet fonksiyonu tanıtır. Ancak şehir içi sürüş, araç yoğunluğu ve kentsel ortamlardaki kavşaklar gibi sahne yapıları nedeniyle otoyolda sürüşten daha karmaşıktır. Bu amaçla, şehir içi sürüşte sıklıkla meydana gelen birçok olası özelliği hesaba katan yeni bir türevlenebilir maliyet fonksiyonu öneriyoruz. Gelecek tahmini için, etmen ve çevrenin gelecekteki durumunu ayrı ayrı tahmin eden ileri modeller öneriyoruz. Etmen (ego) ve çevre (dünya) için ayrı tahminler aracılığıyla alan bilgisini çerçevemize entegre edebiliriz. Ampirik sonuçlar, ego etmeninin geleceğini güvenilir bir şekilde tahmin edebildiğimizi gösterirken dünya modeli, sola veya sağa dönme gibi daha az sıklıkta gerçekleşen durumlarda sınırlamalar sergiliyor. Türevlenebilir maliyet fonksiyonumuzu, gradyan tabanlı model öngörülü kontrol (MÖK) ve bir politika ağıyla test ediyoruz. Her iki yaklaşım da şehir içi sürüş düzeninde benzersiz sınırlamalara sahip makul bir sürüş performansı sergiliyor. MÖK, farklı durumlarda hiper parametrelere duyarlılığı nedeniyle yerel optimum sorunlarıyla karşı karşıyadır ve politika ağı bir uzmanın denetimine bağlıdır. Sonuç olarak, bu çalışmada şehir içi sürüş için türevlenebilir bir maliyet fonksiyonu tasarladık. Aynı zamanda gelecekteki çalışmaların, çalışmamızı temel alarak çeşitli şehir içi senaryolara genelleme yapmanın sınırlamalarını ve zorluklarını ortaya koyduk.

ACKNOWLEDGMENTS

First and foremost, I would like to express my utmost gratitude to Asst. Prof. Fatma Güney and Asst. Prof. Barış Akgün. Quite literally, I could not have attained this Master's degree without them. They came to my rescue when I was least hopeful, helping me to overcome my challenges. In particular, I am indebted to Asst. Prof. Fatma Güney for being a professional and determined role model, and to Asst. Prof. Barış Akgün for his precise and never-ending guidance.

I want to extend my heartfelt thanks to the members of the Robotics and Mechatronics Laboratory (RML) for their warm friendship. In particular, I would like to express my deep gratitude to Easa, who has been the best deskmate one could wish for. I cannot imagine how I would have endured my first year at Koç without our endless imitations, gossip, and engaging conversations. Furthermore, I must thank Berk for being my steadfast all-nighter companion. Our nights filled with laughter, entertainment, and research will forever be etched in my memory. Lastly, I extend my thanks to Berke for being my trusted *mate* in every endeavor. While I pursued my studies at Koç, it was you who enriched my experience there, bringing the spirit of Koç to life daily. I cherish our genuine friendship, which blossomed in the RML and has grown to fill our shared home with camaraderie.

Additionally, I would like to extend my gratitude to each and every one of the members of the Memory, Attention, and Cognitive Control (MACC) Laboratory at Sabancı University, where I have had the honor to be considered a member. Under the guidance of their PI, Asst. Prof. Eren Günseli, they have consistently showcased what a *perfect* lab environment should be, encouraging me to seek out the same.

I want to express my heartfelt thanks to the members of the Autonomous Vision Group (AVG) for their warm and sincere friendship. It has truly been an honor to share this time with all of you. Despite our limited opportunities for collaborative

research or external activities, getting to know and spending time with Mısra and Sadra enriched my experience immensely. I am grateful to Taher for his friendly welcome when I first arrived at the lab, a gesture that allowed me to feel like a *member* from day one. My thanks also go to Shadi, who patiently answered all my inquiries, even those that ventured into the absurd, covering topics from the fundamentals of computer science to intricate details of deep learning. Beyond this, I thank Shadi for feeding me with peanuts and thank his crows for making him buy peanuts. A special note of appreciation goes to Onat, whose ever-present assistance proved invaluable when I encountered obstacles with CARLA and other aspects of our research. His support enabled me to accelerate my research, achieving a year's worth of progress in a condensed timeframe. I must also thank Merve for being a steady break companion, offering moments of light-hearted chats and bursts of laughter. The knowledge that I had someone to turn to for a mental reset was a consistent source of comfort. Last but certainly not least, my deepest thanks go to Nazir, the individual who has epitomized kindness, humor, and thoughtfulness during our time in the lab. His unwavering support and immense faith in my abilities have been nothing short of a blessing. I shudder to think of attempting to complete my thesis without the daily encouragement provided through his morning wake-up calls. Our sushi orders and AoT viewing sessions have been the highlight of my evenings, and I am immeasurably grateful for them.

I must extend my thanks to Koç University, for their consistent approach to administration and support, which never varied no matter how often I sought help. Their remarkable consistency in placing other concerns over those of accommodation, scholarship, or other student benefits has been nothing short of a learning experience. Finally, a special thank you to Koç for embedding a deep understanding of where graduate students stand in the grand scheme of the university hierarchy; a lesson that has been, without doubt, unforgettable.

Special thanks to Adil Kaan Akan for his unparalleled research advice. I remain in awe of the depth and breadth of your knowledge, which seems to encompass

everything. Engaging in fruitful discussions with you has consistently proven to be an enlightening experience.

I would like to thank my entire family for their unwavering moral and material support. In particular, I am infinitely grateful to my mother, Sevim, and my father, Fikret, who never ceased to believe in me, not even for a moment.

Without hesitation, my utmost gratitude is extended to Nursima, my best friend, my love, and my everything. Simply put, none of this would have been possible without the unyielding knowledge that you would stand by me in every situation, both physically and virtually. You have consistently reminded me of my capabilities and potential achievements. Your support was indeed the cornerstone that enabled me to persevere in pursuing this degree. Thank you for enveloping me with your love and care at every juncture, and in every moment.

Finally, I would like to thank myself for stepping onto this difficult path and garnering this experience. This whole journey has demonstrated how people can transform their solid lines, called principles or rules, into a dashed line by compromising them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Network
IL	Imitation Learning
IRL	Inverse Reinforcement Learning
IOC	Inverse Optimal Control
RL	Reinforcement Learning
AD	Autonomous Driving
V2V	Vehicle-to-Vehicle
E2E	End-to-End
BC	Behavior Cloning
ADAS	Advanced Driver Assisted Systems
VAE	Variational Autoencoder
MLP	Multi-Layer Perceptron

Chapter 1

INTRODUCTION

Autonomous driving (AD) in an urban setup remains one of the most challenging problems today, primarily due to the complexities involved in making decisions in intricate real-world scenarios. These complexities arise from uncertainties related to dynamic elements in traffic and the inherent noise in sensor measurements. Yet, achieving autonomy in driving is highly desirable for several reasons. Firstly, an intelligent system that is capable of perceiving and acting on vast amounts of real-world data can make decisions that are more informed than traditional rule-based systems [Bertozzi et al., 2000]. Secondly, a network of self-driving vehicles could improve vehicle-to-vehicle (V2V) communication, proving beneficial in dense traffic scenarios. Finally, by eliminating human error – a major contributor to road incidents – autonomous vehicles have the potential to drastically reduce traffic accidents [Fagnant and Kockelman, 2015].

However, given the high expectations and significant stakes associated with autonomous driving research, real-life testing of AD systems poses challenges. Safety concerns and financial implications [Kaur et al., 2021] have accelerated the interest in developing and testing these systems within simulation environments.

The field also witnesses a debate between the *modular* and *end-to-end* approaches. In the modular strategy, specific subsystems address individual sub-tasks, such as perception, prediction, planning, and control. In contrast, the end-to-end method offers a unified model that directly maps observations to actions. While the modular approach remains popular in the industry, end-to-end learning methods have seen growing interest in recent years [Bronstein et al., 2022, Hu et al., 2022, Toromanoff et al., 2019].

Within *end-to-end* learning, behavior cloning (BC), a unique imitation learning (IL) method, is frequently employed in self-driving research. In BC, the agent learns behavior from an expert dataset. Despite its simplicity, BC has its challenges. Notably, obtaining expert data might not always be feasible, and BC is susceptible to *covariate shift*, where the agent struggles to recover if it deviates from expert behavior. One solution is *on-policy* supervision, allowing the agent to recognize and rectify its mistakes.

Another promising approach in end-to-end learning is reinforcement learning (RL). RL identifies a relationship between observations and actions through *trial and error*. Aside from its distinct advantages, RL offers solutions to some issues related to BC. For instance, RL learns from interacting with an environment, thus negating the need for expert data. Furthermore, the reward signal in RL allows for *on-policy* supervision. Despite its success in areas like Atari games [Mnih et al., 2015] and Go [Silver et al., 2016], it hasn't delivered similar results in self-driving, primarily due to challenges in defining an effective reward function and issues like the *curse of dimensionality* [Sutton et al., 2018].

The integration of the strengths of both IL and RL seems promising. The idea is to extract the reward strategy, a non-differentiable supervision signal, from RL and adapt it for use in a gradient-based optimization framework. Such integration can lead to more efficient learning by offering the agent *on-policy* supervision in an end-to-end and differentiable manner. We thus propose a differentiable policy cost for urban self-driving, quantifying agent actions based on future state predictions. We demonstrate that our proposed cost structure is versatile and suitable for both gradient-based model predictive control (MPC) and policy learning frameworks.

Recently, in terms of future prediction and look-ahead planning, *world models* have gained traction [Henaff et al., 2019, Hafner et al., 2020, Hafner et al., 2021, Sobal et al., 2022]. These models, capturing the dynamics of the environment, can predict its future states, thereby allowing agents to anticipate and plan ahead. However, as studies like [Chen et al., 2021] and [Sobal et al., 2022] suggest, a one-size-fits-all module is not necessary. Instead, *decoupled* modules predicting future ego and world states separately can be more efficient. For instance, predicting ego states

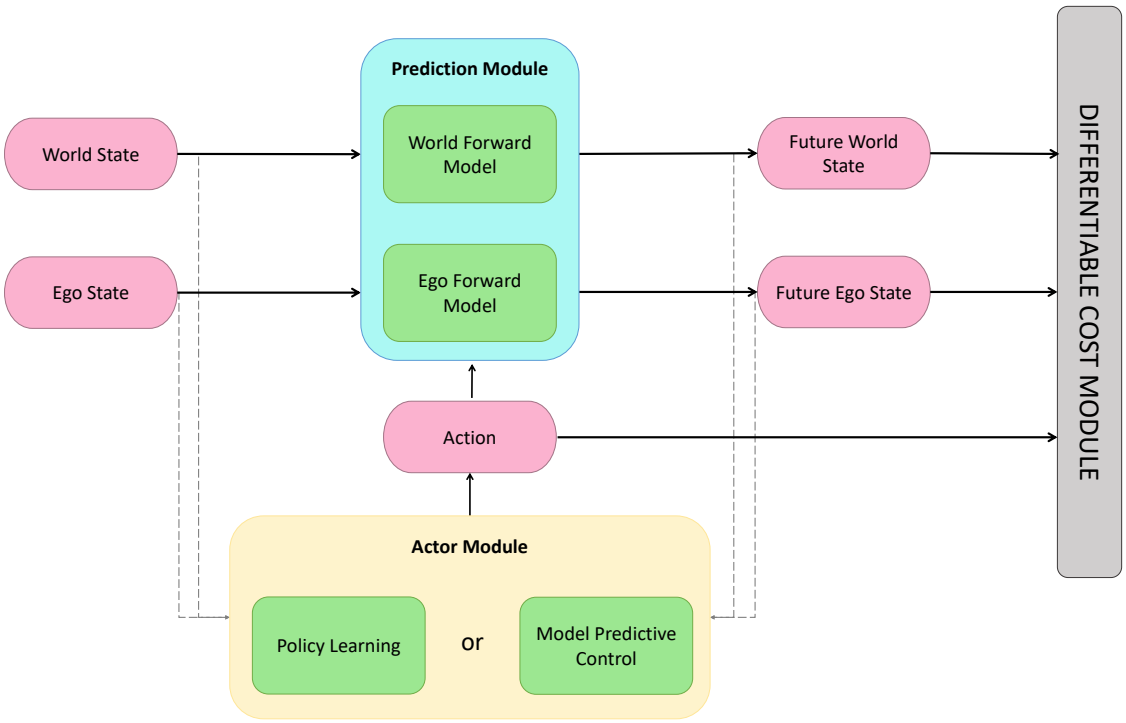

Figure 1.1: **General architecture of the proposed method.** Our proposed framework consists of three components: A future prediction module that predicts the future ego and world state, a differentiable policy cost module to optimize driving behavior, and an actor module that utilizes either an optimal control technique or a learning-based approach.

doesn't require intricate modeling since a simple kinematic model often suffices.

Our proposed framework, illustrated in Figure 1.1, is a model-based driving pipeline comprising three core components. Firstly, it focuses on predicting the future states of both the ego and the world. Secondly, it utilizes a differentiable policy cost structure to optimize driving behavior. Lastly, we employ both an optimal control technique and a learning-based approach to guide the ego's actions within the environment.

We contend that *future*-aware features are invaluable for decision-making, allowing for future planning. Conversely, a differentiable cost module designed specifically for urban driving introduces task-specific guidance into the learning or optimization process. We also explore two methodologies, namely gradient-based model predictive control and policy learning, for navigating the complexities of urban driving, as

detailed in Figure 1.1.

The subsequent chapters of this thesis are structured as follows: Chapter 3 delves into the urban driving challenge, giving detailed information regarding the state definition. Chapter 4 introduces the future prediction module and associated forward models. Chapter 5 elucidates the differentiable cost module, both mathematically and visually, followed by a discussion on gradient-based model predictive control and policy learning. Finally, Chapter 6 provides a concluding overview, outlines current limitations, and suggests potential future directions.

Chapter 2

RELATED WORK

In this chapter, we present a literature review on self-driving in urban areas. It starts with describing the existing simulation frameworks in Section 2.1. Then, in Section 2.2, we review the existing approaches for the self-driving. Section 2.3 delves deeper into the examination of end-to-end (E2E) learning methods. Finally, Section 2.4-2.5 gives a brief summary of existing literature on world models and model predictive control.

2.1 Real World and Simulation

Generally, testing AD systems in real life is challenging and risky due to numerous safety-critical and financial reasons [Kaur et al., 2021]. Essentially, it is not always feasible to have a test vehicle to measure the performance of the proposed AD algorithm on a test track where all precautions are taken. Additionally, reproducibility remains a major challenge for testing in real life. Therefore, researchers and industry partners have developed different simulators over the years to increase the pace of AD research. MATLAB/Simulink™[Inc., 2022] offers Automated Driving Toolbox™, which enables the design, simulation, and testing of ADAS systems. Industry frequently employs the CarSim [Johansson et al., 2004] ADAS simulation framework to simulate diverse sensor modalities and scenarios involving moving objects. PreScan [Siemens PLM Software, 2023] simulation framework offers realistic environments and sophisticated traffic simulations to test ADAS models. LGSVL [Rong et al., 2020] is a high-fidelity multi-robot simulator for autonomous driving that offers realistic environments, a highly customizable sensor stack, and HD Maps. Finally, CARLA [Dosovitskiy et al., 2017] is an open-source autonomous driving simulator developed in Unreal Engine [Epic Games,]. Because of its highly modular architec-

ture and continuous maintenance, CARLA stands out as one of the most extensively used simulators in both research and industry. CARLA offers multiple realistic environments (i.e., towns), highly rational physical interaction models (i.e., collisions with dynamic and static objects), and a large stack of sensor modalities such as camera, LiDAR, radar, semantic segmentation, instance segmentation, depth, GPS, and IMU.

In addition to being a developer-friendly simulator, CARLA offers an evaluation benchmark called CARLA Leaderboard Benchmark [Dosovitskiy et al., 2017] to allow researchers to evaluate their self-driving mode under a predefined set of scenarios. The benchmark measures the performance of the models based on three metrics: route score, infraction score, and driving score. Route score is the percentage of the route completed by an agent. The infraction score is the aggregate metric computed as a geometric series to quantify the effect of infractions. The score starts from 1.0 and it is multiplied by a number between 0 and 1 for each infraction type. Finally, the driving score for a route is defined as the multiplication of the route score and the infraction score of that route.

2.2 Approaches

One can formulate a self-driving problem as an extraction of the mapping function between observations and actions. There are mainly two approaches for self-driving: the modular approach and the end-to-end (E2E) approach. In the modular approach, distinct subsystems, which can be either a deep neural network or a highly engineered rule-based system, handle each sub-task (i.e., perception, prediction, planning, and control). On the other hand, the end-to-end approach maps observations to actions directly using a single module, primarily a deep neural network.

Both approaches have their own advantages and disadvantages [Chen et al., 2023]. While the modular approach tends to offer more explainability and ease of development, it necessitates separate training for each sub-module. Because we train or design each subsystem for its own respective task, it is hard to guarantee that the assembled model can handle the main task, namely *self-driving*. On the other hand,

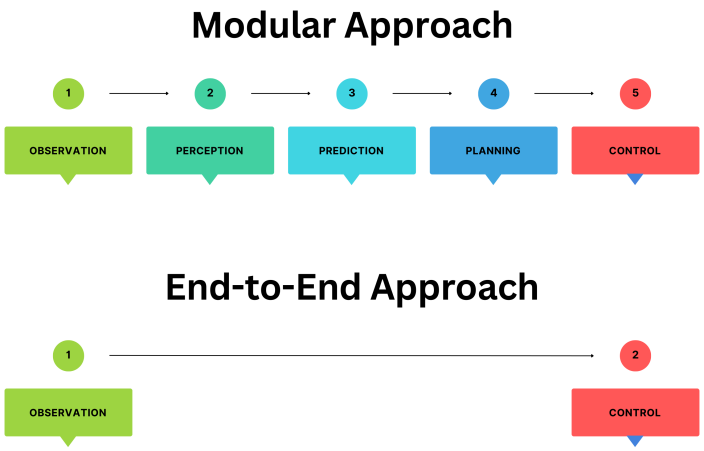

Figure 2.1: **The approaches for self-driving.** The modular approach deals with each sub-task (e.g., perception, prediction, planning) separately, while the end-to-end approach maps observations to actions directly.

the end-to-end approach enables researchers to train their models in an end-to-end manner, thus allowing the possibility of learning unified representations for driving. Additionally, unlike the modular approach, it's possible to optimize the final model specifically for the driving task. However, in an end-to-end learning setup, it is usually more challenging to interpret what the model predicts and detect its failure modes.

2.3 End-to-End Learning Methods

In the context of end-to-end learning for autonomous driving, there are two prominent methods: imitation learning and reinforcement learning.

2.3.1 Imitation Learning

In broad terms, imitation learning (IL) [Hussein et al., 2017] is a supervised learning method where the agent tries to learn the optimal policy by mimicking an expert. Behavior cloning and inverse reinforcement learning approaches are popular within

the imitation learning domain.

Behavior Cloning

Behavior cloning (BC) learns the optimal action from expert data. There is no doubt that behavior cloning is one of the most used methods in the self-driving literature. In the late 80s, [Pomerleau, 1988] made the initial endeavor towards end-to-end learning for autonomous driving. The proposed vehicle, ALVINN, attempts to follow a path with the help of a simple neural network that works primarily with camera and range finder inputs. Later, [Lecun et al., 2004] extends the idea of ALVINN by introducing stereo camera input and convolutional neural network-based (CNN) architecture. Finally, [Bojarski et al., 2016] introduces multi-view (left-right-middle) camera inputs, data augmentation, and deeper CNNs to process camera modality more effectively.

Conditional Imitation Learning (CIL) [Codevilla et al., 2018] argues that although the existing imitation learning agents show competitive performance in benchmarks, these agents are not controllable during test time. CIL [Codevilla et al., 2018] introduces a branched architecture that is conditioned on the high-level navigational command to control model in test time. In this way, they can navigate their model accordingly when there are multiple possible futures (i.e., a car at a road junction). CILRS [Codevilla et al., 2019] improves CIL by introducing several improvements, such as incorporating a better image encoder (i.e., ResNet34) and pretraining it on ImageNet [Russakovsky et al., 2015], and adding a speed prediction module as an inertial inductive bias. CAL [Sauer et al., 2018] proposes framework where it first learns a descriptive intermediate feature of the environment called *affordance* (e.g., distance to the lane center, traffic light state, speed sigh, distance to the nearest vehicle, etc.) with a neural network. Then, Conditional Affordance Learning (CAL) uses these affordances as an input to the rule-based system to decide actions. Finally, Learning by All Vehicles(LAV) [Chen and Krähenbühl, 2022] incorporates a learning paradigm where it does not only learn from the observations within the ego vehicle but it also uses the knowledge gathered from other vehicles.

To this end, LAV detects nearby vehicles from 3D LiDAR input and forecasts their trajectory as an additional training objective.

Multi-Modality. One milestone in developing end-to-end driving systems is the introduction of multi-modal sensor fusion. One of the first examples, LAV [Chen and Krähenbühl, 2022] and TransFuser [Prakash et al., 2021], fuse camera and LiDAR sensor modalities to obtain a unified representation. More specifically, while TransFuser [Prakash et al., 2021] uses an attention-based transformer architecture [Vaswani et al., 2023] to fuse the camera images and LiDAR scans, LAV [Chen and Krähenbühl, 2022] uses PointPillars [Lang et al., 2019] 3D backbone to utilize multi-modal input.

Policy Distillation. Another important landmark is the application of policy distillation paradigm in end-to-end learning. One of the simplest application of policy distillation in self-driving domain is the LbC [Chen et al., 2019]. Learning by Cheating(LbC) first learns a Bird's Eye View (BEV) representation-based privileged agent. Then, it uses the privileged agent as an oracle that provides an *on-policy* supervision to the vision-based sensorimotor agent. World on Rails (WoR) [Chen et al., 2021] learns an interactive vision-based policy by distilling another forward policy, which is an action-value table computed using dynamic programming. They show a competitive performance in CARLA Leaderboard although they assume that the *world* is *on rails*, that is, ego's action does not affect the world. Roach [Zhang et al., 2021a] first trains an privileged RL agent and then uses the privileged agent as a coach/expert during the IL training. They argue that the privileged agent can give more explanatory supervision signals as it has access to the internal dynamics of the simulation environment. In addition to be used as an expert, Roach also incorporates additional *feature* supervision to induce privileged information to the IL student. CaT [Zhang et al., 2023] extends the idea of Roach by providing multiple level of *feature* supervision through intermediate layers. They also introduces a *alignment module* to hinder the negative effects that arises due to the differences in input modality, model complexity, and optimization procedure.

Output Representations In literature, there are two main approaches for predicting actions through behavior cloning. One approach is to predict the actions or their distributions (i.e., throttle, steer, brake) [Zhang et al., 2021b, Zhang and Ohn-Bar, 2021, Codevilla et al., 2019, Wu et al., 2022]. Another approach is to predict waypoints and feed them to separate PID controllers to generate longitudinal (i.e., throttle, and brake) and lateral (i.e., steering angle) actions [Prakash et al., 2021, Chitta et al., 2022, Chen and Krähenbühl, 2022, Chitta et al., 2021]. [Wu et al., 2022] investigate that these two approaches differ in performance under definitive circumstances. Specifically, while the waypoint-based approach fails at intersections, the control-based approach is more prone to fail in situations where the agent should take action immediately. Therefore, TCP [Wu et al., 2022] proposes to combine these two approaches to extract the best of both. The proposed model uses a branched architecture to incorporate two techniques (i.e., control and waypoint) in parallel, and it adapts an attention-like mechanism to guide the control branch with the waypoint branch. Finally, they place a rule-based *fusion* module to further boos the performance.

Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) [Russell, 1998], sometimes referred to as inverse optimal control (IOC), learns the reward or the cost structure that guides the actions of an observed expert. Once we approximate the inherent reward or cost structure, it becomes applicable in optimal control or reinforcement learning methodologies. GAIL [Ho and Ermon, 2016] and its derivatives [Li et al., 2017, Lee et al., 2020] utilize a reward-based adversarial framework where the model learns to imitate the expert actions while a discriminator tries to differentiate between the model's and expert's behavior. In the context of self-driving, NPM [Zeng et al., 2021] learns a 3D cost volume and additional intermediate representations in the form of 3D object detections and their future trajectories with LiDAR and HD Map inputs. Then, NPM [Zeng et al., 2021] samples a set of trajectories to select the one with the minimum cost in a non-differentiable manner.

2.3.2 Reinforcement Learning

Reinforcement learning (RL) [Sutton et al., 2018] is a type of learning algorithm where an agent learns the optimal policy by interacting with the environment through *trial and error*. This method often relies on feedback in the form of rewards or penalties.

One of the first examples of collaboration between *deep learning* and *reinforcement learning* is deep Q-network [Mnih et al., 2015]. It uses deep neural networks to learn successful policies directly from high-dimensional sensory inputs by parameterizing Q-values. [Mnih et al., 2015] can only work on discrete action space. [Lillicrap et al., 2019] solves this issue by introducing an *actor-critic*, model-free algorithm based on the deterministic policy gradients (DPG), where the *actor* learns how to map states to continuous actions, and *critic* estimates the Q-value of the current policy via using Bellman Optimality Equation (BOE). On the other hand, policy gradient methods improve the agent's policy by maximizing the expected reward through gradient ascent on the policy parameters. As a new set of policy gradient methods, [Schulman et al., 2017] attempts to update policies by clipping the probability ratio of actions taken in the current and previous policies to prevent large policy updates.

In the context of self-driving, [Kendall et al., 2018] demonstrates the first example of deep reinforcement learning policy installed to a real vehicle. They use a continuous action, model-free reinforcement algorithm to learn a *lane following* policy in a *day* with an affordable reward structure. Later, [Toromanoff et al., 2020] introduces the term *implicit affordances* to describe the output of a pretrained image encoder. They train the image encoder in a supervised fashion with semantic segmentation, traffic light, and intersection detection objective. Then, they use the output of the frozen image encoder as a feature vector for their RL algorithm. Finally, [Zhang et al., 2021a] trains a *privileged* RL agent with the purpose of creating an expert driver that can provide *on-policy* supervision. It is important because it is not possible to provide *on-policy* supervision with rule-based experts and they also argue that an agent having an access to internal dynamics of a simulation can

provide more descriptive supervision signals in an imitation learning setup. Later, they train imitation learning agents by distilling a policy from the privileged RL expert, and they show the distilled IL agent can provide competitive performance in the CARLA benchmark.

2.4 World Models

A world model is a computational tool that mimics the internal dynamics of an environment (e.g., simulated or real). [Schmidhuber, 2015] and [Ha and Schmidhuber, 2018] introduce RNN-based world models to extract useful representations of *future* states. The proposed world models [Schmidhuber, 2015, Ha and Schmidhuber, 2018] operate in a compact latent space, in which a variational autoencoder can encode and decode, and use the learned representations in another *controller* module to facilitate short-term planning via reward maximization. Similarly, PlaNet [Hafner et al., 2019] facilitates a world model to learn the internal dynamics of the simulation environment; and then uses the world model to imagine trajectories, which enables a population-based search to find the *optimal* action. Subsequently, Dreamer [Hafner et al., 2020] and DreamerV2 [Hafner et al., 2021] extends the idea of learning world model in a compact latent space by introducing a Dyna-style [Sutton, 1991] off-policy update to learn an action model. Finally, [Henaff et al., 2019] and [Sobal et al., 2022] utilize world models to predict future states of the environment, which they use to calculate a differentiable policy cost. Especially, [Sobal et al., 2022] follows a *decoupling* strategy to separate future predictive models for world and ego, since modeling ego is not as exhaustive as modeling world.

2.5 Model Predictive Control and Planning

Model predictive control [García et al., 1989] stands as a renowned control method. It relies on a clear definition of both the system and the cost function, allowing it to operate without expert supervision over extended time horizons. With certain assumptions in place, methods like quadratic programming [Maciejowski, 2002], nonlinear programming [Camacho and Alba, 2013], explicit MPC [Bemporad et al.,

2002], and the iterative Linear Quadratic Regulator (iLQR) [Li and Todorov, 2004] address the traditional MPC problem. Recently, though, there has been a shift towards gradient-based and sampling-based solution methods. Such methods gain traction as they facilitate the application of MPC within an end-to-end learning framework, known as *differentiable planning*. UPN [Srinivas et al., 2018] employs a gradient-based MPC for planning in latent spaces. [Amos et al., 2019] introduces a method to compute analytical gradients with respect to an MPC module, notably accelerating the *back-propagation* process by a factor of 10^2. In their discussion, [Bharadhwaj et al., 2020] suggests that both gradient and sampling-based optimization techniques present unique merits and shortcomings. As a solution, GradMPC [Bharadhwaj et al., 2020] unveils a framework merging the sampling-based cross entropy method (CEM) with gradient-based MPC for faster convergence. Further, TD-MPC [Hansen et al., 2022] incorporates a sampling-based CEM module to enhance the action suggestions from the policy network. Lastly, [Romero et al., 2023] utilizes an actor-critic architecture to forecast elements of weight matrices in the MPC module, which subsequently guides action prediction for quadcopter control.

Chapter 3

URBAN DRIVING SYNOPSIS

In this chapter, we give detailed information related to our urban driving environment. In Section 3.1, we, first, describe the *urban driving* problem by giving all necessary state and action definitions. Then, we introduce the simulation environment in Section 3.2. Finally, Section 3.3 gives a detailed explanation of the bird's-eye-view (BEV) representation, which we use as our world state in our experiments.

3.1 Problem Definition

In an urban driving task, for a given time $t \in \mathbb{N}$, the environment yields a state $\mathbf{s}_t \in \mathbb{S}$ which consists of ***ego*** state $\mathbf{s}_t^{ego} \in \mathbb{S}^{ego}$, and ***world*** state $\mathbf{s}_t^{world} \in \mathbb{S}^{world}$. Accordingly, *ego* refers to the agent/vehicle being controlled, and the *world* corresponds to everything in the environment except the ego. Figure 3.1 presents an example scene from the environment that shows the distinction between *ego* and *world*. The agent generates an action $\mathbf{a}_t \in \mathbb{A}$ in the form of gas $\mathbf{a}_t^{gas} \in \mathbb{A}^{gas}$, brake $\mathbf{a}_t^{brake} \in \mathbb{A}^{brake}$, and steering wheel angle $\mathbf{a}_t^{steer} \in \mathbb{A}^{steer}$. The high-level aim is to travel from one point to another through a predetermined set of routes as quickly as possible. Moreover, the agent should have as few infractions (i.e., traffic violations per kilometer) as possible. Towards this end, in each time step, the agent receives the target state $\mathbf{s}_t^{target}$, which is a tuple of $(x^{target}, y^{target}, \psi^{target}, c)$, where x^{target} and y^{target} represent the target location in world coordinates, ψ^{target} represents the target yaw angle, and c represents the high-level navigational command (e.g., lane follow, turn right, turn left, etc.) provided by the simulation. Mimicking real life, the simulation provides target information in a sparse manner; that is, the distance between two target locations might reach a few hundred meters, which eliminates

Figure 3.1: **An example scene from the environment.** Here, the green-shaded vehicle represents the *ego*, and all other elements in the scene (i.e., traffic lights, pedestrians, cars, etc.) represent the *world*. **Note:** This is just an example to describe *ego* and the *world*. We do not use this representation (i.e., 3rd person view) in our environment.

the possibility of creating a *naive* path follower system.

Specifically, $\mathbf{s}_t^{ego} \in \mathbb{R}^4$ is a tuple of $(x^{ego}, y^{ego}, \psi^{ego}, v^{ego})$ where x^{ego} and y^{ego} represent the *location* in *world* coordinates, ψ^{ego} represents the *yaw* angle (i.e., heading), and v^{ego} represents the *speed* of the ego vehicle. On the other hand, $\mathbf{s}_t^{world} \in \mathbb{R}^{C \times H \times W}$ is a rasterized bird's-eye-view (BEV) representation of a localized region around the ego vehicle, where C is the number of semantic classes, H and W is the height and width of the rasterized representation in pixels. In our experiments, we specify the localized region such that it spans the region only in front of the ego vehicle, starting from the center of gravity of the vehicle. In this manner, we place the ego vehicle at the bottom center of the BEV representation. The localized region has the dimensions of 38.4×38.4 m^2, which we discretize into 192×192 pixels. Finally, we utilize both $\mathbf{s}_t^{ego}$ and $\mathbf{s}_t^{world}$ as ground-truth.

3.2 Simulation Environment: CARLA

We use CARLA 0.9.13 [Dosovitskiy et al., 2017] as an urban driving simulator. Using a custom-prepared code base, we establish the entire communication with CARLA in Python. We directly acquire the ego state from the simulation, which consists of the agent's ground-truth location, yaw angle, and speed. We calculate the world state (i.e., BEV representation) using HD Maps provided by the CARLA simulator. We achieve the final version of the rasterized representation by parsing and postprocessing the HD Map. Figure 3.2 illustrates an example BEV representation.

Similar to our work, [Sobal et al., 2022] and [Henaff et al., 2019] use NGSIM data [U.S. Department of Transportation Federal Highway Administration, 2016] to simulate highway driving scenarios. In discordance with our urban driving environment, their simulation environment is much simpler because the highway driving problem has fewer uncertainties and less interaction with other environmental elements. Moreover, the high-level goal is clearly defined in the highway driving problem, which is to follow the lane and avoid collisions. However, in urban driving, although the relatively simple high-level goal is to reach the target location with minimal infractions, the existent low-level goals (e.g., changing lanes, stopping at traffic lights, yielding to pedestrians, etc.) make the problem harder.

3.3 Bird's-Eye-View (BEV) Representation

Bird's-eye-view (BEV) representation is a top-down view of the semantic world, meaning each pixel represents a specific class or set of classes. For driving, we argue that the BEV representation serves as a more natural representation of the world, as it is more aligned with the 2D geometry of driving [Hu et al., 2022], compared to the pure RGB-based representations.

We include 12 different semantic classes in our BEV representation. We illustrate these classes in Figure 3.2. Table 3.3 presents the color-coded semantic classes.

The complex nature of urban driving task requires the fusion of multiple information emitted from different sources. Therefore, we add every piece of information that we can obtain from the simulator into the BEV representation. For example,

Agent	Lanes affected by the green light	Green light
Vehicle	Lanes affected by the red or yellow light	Red or yellow lights
Pedestrians	Lanes that agent can drive	Lane markings
Road	Lanes that agent cannot drive	Offroad

Table 3.1: **Semantic classes used in the BEV representation.** Each color corresponds to the color used to visualize the BEV representation.

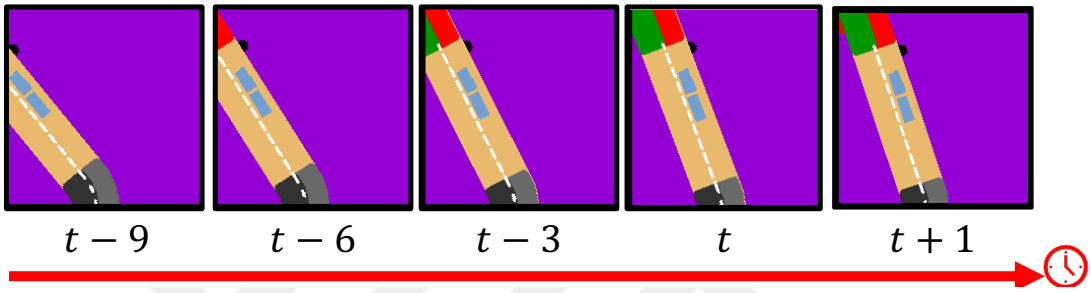

Figure 3.2: **An example BEV representation of the world.** This specific set of examples presents a top-down representation of the world while the ego vehicle is rotating. Each pixel corresponds to one or multiple semantic classes distinguished by the color. Table 3.3 presents the color-coded semantic classes. The time difference between each frame is 0.05 *sec* since we collect the data with a sampling frequency of 20 Hz.

since the gradient-based MPC scheme (see Section 5.2.2) depends on the spatial information to navigate in complex scenarios, we include lane-specific channels (e.g., lanes that agent can drive and lanes affected by the green light) at different levels. At various phases of our study, we experimented with the different subsets of the semantic classes presented above, and we observed that the choice of the subset of semantic classes affects each model differently. In other words, a subset with a few semantic classes might be a proper choice to train a world forward model since it will reduce the model's complexity. However, the same subset might not be a good choice for the policy model, as it will not provide enough information for the ego vehicle to drive.

Chapter 4

FORWARD MODELS FOR FUTURE PREDICTION

In this chapter, we handle the future prediction problem and investigate the forward models used in our study. First, in Section 4.1, we briefly introduce the future prediction problem and mathematical formulation of forward models. Then, in Section 4.2, we introduce our ego and world forward models, along with the details regarding their architectures, training, and evaluation. Next, in Section 4.3, we present the evaluation results of our forward models and discuss the results. Finally, in Section 4.4, we conclude the chapter with a summary and a discussion of limitations and future work.

4.1 Introduction

This section describes the future prediction problem and the forward models. Formally, we start by assuming that we are a given a sequence of states $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$, where $\mathbf{x}_i$ represents the state at time i. The future prediction objective is to find a sequence of states $\mathbf{Y} = (\mathbf{y}_{t+1}, \mathbf{y}_{t+2}, \ldots, \mathbf{y}_{t+h})$, where $\mathbf{y}_i$ represents the predicted state at time i and h represents the prediction horizon. One way to solve future prediction problems is to *learn* a forward model. Henceforth, *forward models* refer to the models that predict the future state given the current state (e.g., ego state $\mathbf{s}^{ego} \in \mathbb{R}^4$ or world state $\mathbf{s}^{world} \in \mathbb{R}^{N \times C \times H \times W}$) and some additional information (e.g., action, previous state, etc.). Mathematically speaking, we can represent a *deterministic* forward model as follows:

$$\mathbf{s}_{t+1} = f_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}) \tag{4.1}$$

$$= f(\mathbf{s}_t, \mathbf{a}; \boldsymbol{\theta}) \tag{4.2}$$

where $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$ represent the current and future state; $\boldsymbol{\theta}$ represents the true model parameters, $\mathbf{a}$ represents the additional information (i.e., action), and, finally, f stands for the nonlinear forward function that represents the inherent dynamics. In general, the term *learning* refers to the process of finding a parameter set $\boldsymbol{\beta}$ such that the discrepancy between $f_{\boldsymbol{\theta}}$ and $f_{\boldsymbol{\beta}}$ is minimized.

For a *stochastic* forward model, some noise or randomness also accompanies the calculation of the next state. We represent the stochastic forward model as follows:

$$\mathbf{s}_{t+1} \sim \mathcal{N}\left(\mu_{\boldsymbol{\theta}_1}\left(\mathbf{s}_t, \mathbf{a}\right), \sigma_{\boldsymbol{\theta}_2}\left(\mathbf{s}_t, \mathbf{a}\right)\mathbf{I}\right) \tag{4.3}$$

where, $\mathcal{N}$ denotes normal distribution where $\mathbf{s}_t$, $\mathbf{a}$, and $\boldsymbol{\theta}_{[.]}$ parameterize the mean and the variance. Similar to the deterministic case, we aim to learn a set of parameters $\boldsymbol{\gamma}$ such that we minimize the discrepancy between true model's probability distribution $\mathcal{N}\left(\mu_{\boldsymbol{\theta}_1}\left(\mathbf{s}_t, \mathbf{a}\right), \sigma_{\boldsymbol{\theta}_2}\left(\mathbf{s}_t, \mathbf{a}\right)\mathbf{I}\right)$ and the estimated distribution
$\mathcal{N}\left(\mu_{\boldsymbol{\gamma}_1}\left(\mathbf{s}_t, \mathbf{a}\right), \sigma_{\boldsymbol{\gamma}_2}\left(\mathbf{s}_t, \mathbf{a}\right)\mathbf{I}\right)$.

4.2 Methods

In this section, we present the forward models that we design to predict future states of ego and world.

4.2.1 Ego Forward Model

Following [Chen et al., 2021] and [Sobal et al., 2022], we adopt the kinematic bicycle model [Rajamani, 2012] as the ego forward model. We can formally describe the kinematic bicycle model as follows:

$$\dot{x} = v\cos(\psi + \beta) \tag{4.4}$$

$$\dot{y} = v\sin(\psi + \beta) \tag{4.5}$$

$$\dot{\psi} = \frac{v\cos(\beta)\tan(\delta)}{L} \tag{4.6}$$

$$\dot{v} = a \tag{4.7}$$

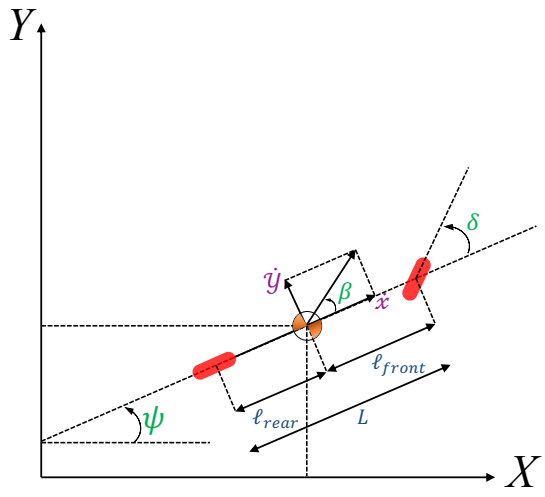

Figure 4.1: **A schematic of the kinematic bicycle model in *world* $(X-Y)$ coordinates.** Here, $\dot{x}$ and $\dot{y}$ represent the longitudinal and lateral speed in *body* coordinates, respectively. ψ represents the yaw angle; β represents the slip angle, which is the angle between the longitudinal axis of the vehicle and the velocity vector. l_{rear} and l_{front} define the rear and front wheelbase length, respectively. Finally, δ, steer angle, represents the control command provided to the vehicle.

where x and y are the location in world coordinates, ψ is the yaw angle, and v is the speed of the vehicle. In addition, the intermediate parameter β stands for the slip angle, representing the angle between the velocity vector and the vehicle's longitudinal axis. According to [Rajamani, 2012], we can calculate the slip angle as follows:

$$\beta = \arctan\left(\frac{l_r \tan(\delta)}{L}\right) \tag{4.8}$$

where l_r is rear wheelbase and L is wheelbase of the vehicle. Finally, the control commands fed to the vehicle are the longitudinal acceleration a and the steering angle δ. For simplicity, we approximate the relation between gas-brake and longitudinal acceleration, and the relation between steering angle and steering wheel angle as linear transformation:

$$a = \alpha_1 \hat{a} \tag{4.9}$$

$$\delta = \alpha_2 \hat{\delta} \tag{4.10}$$

where $\hat{\delta}$ stands for the real steering wheel angle command provided to the vehicle, and $\hat{a}$ stands for the piece-wise linear function of gas and brake command fed to the vehicle:

$$\hat{a} = \begin{cases} \text{gas} & \text{if gas} > 0 \\ -\text{brake} & \text{if brake} > 0 \end{cases} \tag{4.11}$$

provided that the gas and brake cannot be greater than zero simultaneously. Adapting the (4.1), we can represent ego forward model as follows:

$$x_{t+1}, y_{t+1}, \psi_{t+1}, v_{t+1} = f_{ego}\left(x_t, y_t, \psi_t, v_t, a_t, \delta_t \ ; \ l_r, L, \alpha_1, \alpha_2\right) \tag{4.12}$$

In (4.12), l_r, L, α_1, and α_2 represent the independent variables we fit in the learning process.

Data Collection

Following [Chen et al., 2021], we collect a dataset that includes random actions to capture a wide range of state transitions. We perform the data collection process within the CARLA simulation environment. In each episode, we spawn the ego vehicle at a random location and provide the agent with a random action at each time step until collision with an object occurred. In total, we collect 20k data points, which correspond to 120 episodes of simulation runs. We split the dataset such that there are 80 episodes for training, 20 episodes for validation, and 20 episodes for testing. We repeat the actions a few times to make state transitions more distinct. The simulation runs in 20 Hz, meaning that the time difference between each sample is $\Delta t = 0.05$ *sec*. Figure 4.2 shows an example of a random data collection episode.

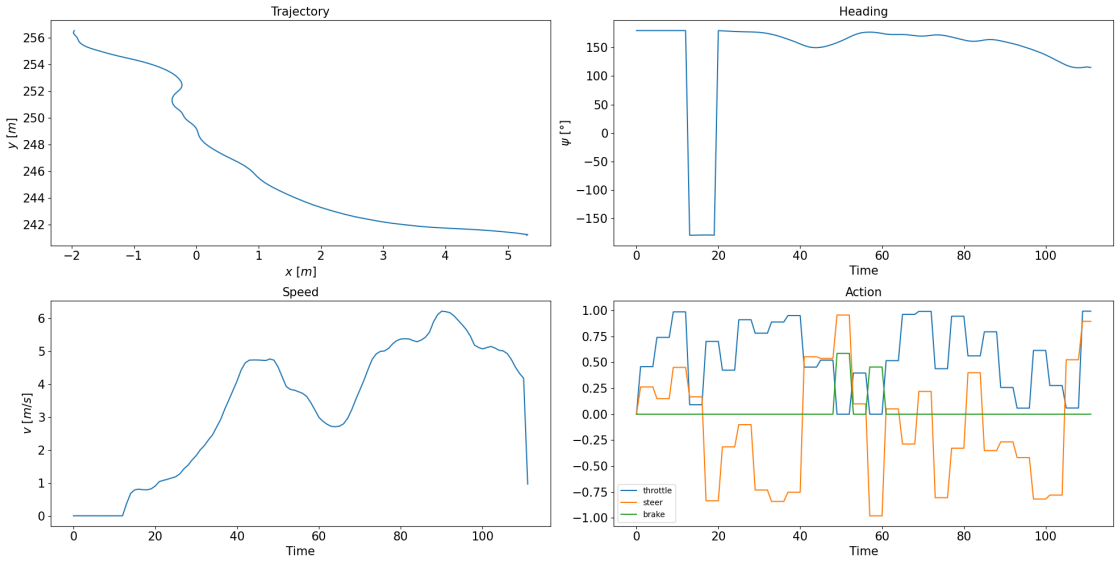

Figure 4.2: **An example episode from the dataset collected for the training of the ego forward model.** The top-left graph visualizes the trajectory of the ego vehicle. The top-right and bottom-left graphs illustrate the change of heading and speed, respectively, throughout the episode. Finally, the Bottom-right graph shows the *random* control commands fed to the ego vehicle.

Training and Inference Procedure

We use PyTorch [Paszke et al., 2019] as the main framework in our experiments. We model the ego forward model using PyTorch, resembling the kinematic bicycle model, with the parameter set $\boldsymbol{\theta} = (l_r, L, \alpha_1, \alpha_2)$ designated as learnable parameters. Our goal is to make the predicted ego state as close as to the actual ego state according to the per time step objective in (4.13):

$$J_t = |x_t - \hat{x}_t| + \tag{4.13}$$

$$|y_t - \hat{y}_t| + \tag{4.14}$$

$$\left|\cos\psi_t - \cos\hat{\psi}_t\right| + \tag{4.15}$$

$$\left|\sin\psi_t - \sin\hat{\psi}_t\right| \tag{4.16}$$

To achieve a robust training objective, we can minimize the overall training objective by summing up the unrolled J_t over a fixed horizon T in a sliding window manner:

$$J = \sum_{t=0}^{T} J_t \tag{4.17}$$

During training and inference, the ego forward model takes the current ego state $\mathbf{s}_t^{ego} \in \mathbb{R}^4$, and the action $\mathbf{a}_t \in \mathbb{R}^2$ as input and outputs the predicted ego state $\hat{\mathbf{s}}_t^{ego} \in \mathbb{R}^4$, in each time step.

Evaluation Metric

We use the mean absolute error (MAE) as the evaluation metric for the ego forward model. We define the MAE as follows:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} \left| \mathbf{s}_{t+i}^{ego} - \hat{\mathbf{s}}_{t+i}^{ego} \right| \tag{4.18}$$

where N represents the number of time steps in the prediction horizon, $\mathbf{s}_{t+i}^{ego}$ represents the ground-truth ego state at time $t+i$, and $\hat{\mathbf{s}}_{t+i}^{ego}$ represents the predicted ego state at time $t+i$.

4.2.2 World Forward Model

Similar to [Babaeizadeh et al., 2018], [Denton and Fergus, 2018], [Sobal et al., 2022], we adopt a probabilistic approach for the construction of the world forward model. We choose to model the world in a stochastic manner because deterministic approaches fail to predict multiple possible futures [Henaff et al., 2019], as they average out the possible features.

For probabilistic modeling, we use a popular architecture called variational autoencoders (VAEs) [Kingma and Welling, 2022]. VAEs are generative models that learn the underlying distribution of the data. The VAE architecture consists of two main components: an encoder and a decoder. The encoder takes the input data and encodes it into a latent representation. The decoder takes the latent representation and reconstructs the input data. We train the encoder and the decoder jointly to minimize the reconstruction loss. In addition, we train the encoder to minimize the

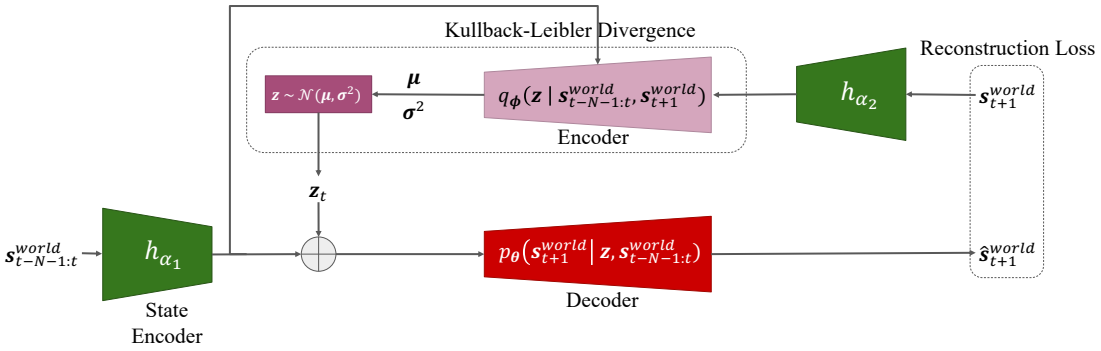

Figure 4.3: **A schematic of the world forward model.** We structure the world forward model as a conditional variational autoencoder. We condition future world states on the past N world states. In each time step, the posterior network q_ϕ takes the past N world states and the future world state as input and outputs the parameters of the latent distribution. We then feed the latent representation conditioned on the past world states to the prior network p_θ to reconstruct the future world state.

KL divergence between the latent representation and a prior distribution. In our case, we use a conditional VAE architecture [Sohn et al., 2015], where the goal is to reconstruct future world states given N past world states. Figure 4.3 illustrates the general architecture of the world forward model. We represent the training objective of the world forward model as follows:

$$\mathcal{L}_\tau(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, \mathbf{c}) = \underbrace{-\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_\tau|\mathbf{x}_\tau,\mathbf{c}_\tau)}[\log p_{\boldsymbol{\theta}}(\mathbf{x}_\tau|\mathbf{z}_\tau,\mathbf{c}_\tau)]}_{\text{reconstruction term}} + \underbrace{\mathrm{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}_\tau|\mathbf{x}_\tau,\mathbf{c}_\tau)||p(\mathbf{z}_\tau))}_{\text{KL divergence term}} \quad (4.19)$$

where $\mathbf{x}_\tau$ represents the future world state $\mathbf{s}^{world}_{t+1}$, $\mathbf{c}_\tau$ represents the past N world states $\mathbf{s}^{world}_{t-N-1:t}$, $\mathbf{z}_\tau$ represents the latent variable, $p_{\boldsymbol{\theta}}$ represents the decoder, and $q_{\boldsymbol{\phi}}$ represents the encoder.

Adapting the (4.1), we represent the world forward model as follows:

$$\mathbf{s}^{world}_{t+1} = f_{world}\left(\mathbf{s}^{world}_{t-N-1:t} \;;\; \boldsymbol{\theta}, \boldsymbol{\phi}\right) \quad (4.20)$$

where $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ represent the parameters of posterior and prior networks, respectively.

Data Collection

We, again, utilize CARLA simulation to collect data for the world forward model. Similar to the procedure in 4.2.1, in each episode, we spawn the ego vehicle in a random location, and CARLA's built-in *Autopilot* controls it for a fixed number of steps. We execute the simulation in 20 Hz, meaning that the time difference between each sample is $\Delta t = 0.05$ *sec*. We illustrate an example sequence in Figure 3.2.

Training and Inference Procedure

The training procedure of the world forward model follows a very similar pipeline to that of the ego forward model as detailed in 4.2.1. The only difference is that we train the world forward model in a conditional manner. In other words, we train the world forward model to reconstruct the future world state given the past N world states. The aim is to minimize the per time step objective given in (4.19) over a fixed horizon T in a sliding window fashion:

$$J = \sum_{\tau=0}^{T-1} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2; \mathbf{x}_\tau, \mathbf{c}_\tau) \tag{4.21}$$

During training, in each time step, we feed the past N world states $\mathbf{s}^{world}_{t-N-1:t} \in \mathbb{R}^{N \times C \times H \times W}$ and future world state $\mathbf{s}^{world}_{t+1} \in \mathbb{R}^{1 \times C \times H \times W}$ to separate encoders (i.e., $h_{\boldsymbol{\alpha_1}}$ and $h_{\boldsymbol{\alpha_2}}$) to extract both past and future features and to obtain common dimensionality between past and future states. Note that, in this input-output representation, N corresponds to the number of past world states fed to the model, which we refer to as *context horizon*, C is the number of semantic classes, H and W are the height and width of the BEV representation. Then, we sum the extracted features and feed them to the probabilistic encoder of the VAE, which produces the parameters of the prior distribution. With the help of the reparameterization trick (i.e., $\epsilon \sim \mathcal{N}(0, 1)$ and $z = \mu + \sigma \odot \epsilon$), we sample a latent vector from the prior distribution. Finally, after combining the past world state features (i.e., conditioning) with the latent vector, the decoder reconstructs the future world state

$\hat{\mathbf{s}}_{t+1}^{world}$. We use the per-channel binary cross-entropy loss as a reconstruction loss in a multi-label prediction setup. We calculate the KL divergence term between the posterior distribution and the prior distributions, generally modeled as a standard normal distribution. Eventually, we calculate the overall loss term as the weighted combination of the reconstruction term and KL divergence term:

$$\mathcal{L} = \mathcal{L}_{reconstruction} + \beta \mathcal{L}_{KL} \tag{4.22}$$

where β is the weight of the KL divergence term. In our experiments, we set $\beta = 10^{-4}$.

The world forward model predicts the future world state during inference given the past N world states. In each time step, we feed the past N world states $\mathbf{s}_{t-N-1:t}^{world} \in \mathbb{R}^{N \times C \times H \times W}$ to the encoder to extract the past features. Then, we sample a latent vector from the prior distribution, which we already modeled as a standard normal distribution. Finally, after combining the past world state features (i.e., conditioning) with the latent vector, the decoder reconstructs the future world state $\hat{\mathbf{s}}_{t+1}^{world}$.

Evaluation Metric

We use the per channel mean Intersection over Union (mIoU) as the evaluation metric for the world forward model. We define mIoU as follows:

$$\text{mIoU}_c = \frac{1}{N} \sum_{i=1}^{N} \frac{{}_c\mathbf{s}_{t+1}^{world} \cap {}_c\hat{\mathbf{s}}_{t+1}^{world}}{{}_c\mathbf{s}_{t+1}^{world} \cup {}_c\hat{\mathbf{s}}_{t+1}^{world}} \tag{4.23}$$

where N represents the number of time steps in the prediction horizon, c represents the semantic class, ${}_c\mathbf{s}_{t+1}^{world}$ represents the ground-truth future world state for the semantic class c, and ${}_c\hat{\mathbf{s}}_{t+1}^{world}$ represents the predicted future world state for the semantic class c. Additionally, we present per-channel metrics such as precision, recall, and F1 score for completeness.

	Location MAE	**Rotation MAE**
Test set	0.0742	0.0533
Train set	0.0647	0.0467

Table 4.1: **Ego forward model evaluations results.** MAE is calculated in meters for location and unitless for rotation.

4.3 Results and Discussion

In this chapter, we present and discuss evaluation results of ego and world forward models, quantitatively and qualitatively.

4.3.1 Ego Forward Model

Table 4.1 shows the evaluation results of the ego forward model. The ego forward model achieves 7.42 *cm* MAE in the test set and 6.47 *cm* MAE in the train set, which indicates that the proposed ego forward model is able to predict future ego location with an error of less than 10 *cm*. We find 10 *cm* MAE acceptable for the ego forward model since it corresponds approximately to %2 of the length of the ego vehicle and %5 of the width of the ego vehicle.

Qualitative Results. Figure 4.4 shows the rollout performance of the ego forward model for a random action sequence taken from the test set, and Figure 4.5 shows the rollout performance of the ego forward model for a sequence of actions taken from a realistic driving scenario. In both cases, the ego forward model is able to predict the future ego location with a high accuracy.

4.3.2 World Forward Model

We experiment with two pairs of semantic class subsets and sampling rates in world forward model training. In the first pair, which we denote as *simple* case, we use the following subset of semantic classes: roads, lane markings, vehicles, green lights, red or yellow lights, and offroad with the 20 Hz sampling rate. In the second pair,

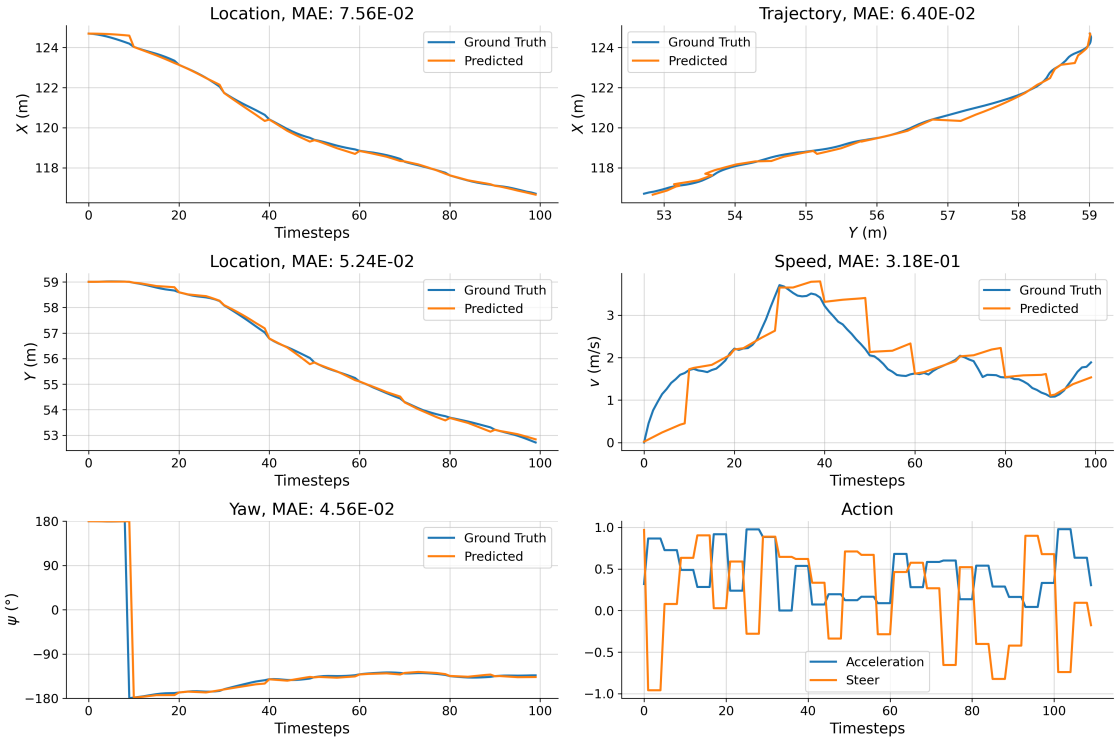

Figure 4.4: **Rollout of the ego forward model under a random action sequence.** We present MAE scores of each property next to the corresponding graph title.

which we denote as *hard* case, we use the following subset of semantic classes: roads, lanes that the agent can drive, lanes that the agent cannot drive, lanes affected by the red or yellow light, lanes affected by the green light, lane markings, vehicles, and offroad with the 5 Hz sampling rate. Although some of our models have different prediction horizons during training, but for evaluation, we set all the model's prediction horizons to 10 in order to make a fair comparison.

In the context of urban driving, we are mostly interested in the vehicle mIoU score, since predicting the future locations of other vehicles has a crucial importance for navigating safely in the world. Therefore, our discussion revolves around the vehicle mIoU score. Table 4.2 shows the evaluation results of the *simple* case for four different context and prediction horizon configurations. According to Table 4.2, we observe that increasing context length has a positive effect on the mIoU score of vehicle class. We argue that it is because the increased context horizon

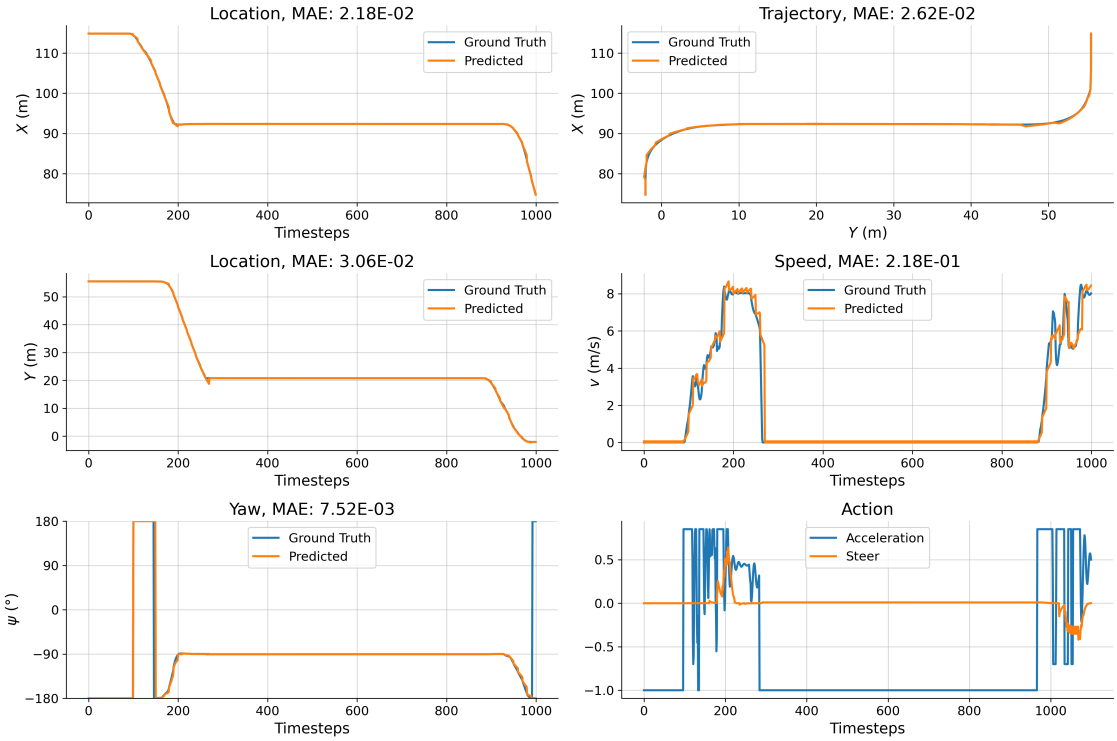

Figure 4.5: **Rollout of the ego forward model under an action sequence taken from a realistic action sequence.** We present MAE scores of each property next to the corresponding graph title.

enables the model to preserve more information regarding the motion of the other vehicles. Similarly, investigating the first (N=20, T=30) and second (N=20, T=10) configuration in Table 4.2, for a fixed context horizon, we observe that increasing the prediction horizon results in an improvement in the vehicle mIoU score. We argue that back-propagating the loss accumulated through multiple number of time steps enables a more robust training and it increases the performance of the world forward model.

On the other hand, we observe a decrease in performance when transitioning to the *hard* case. We believe this is due to the changes between frames increasing significantly, making it more challenging for the model to learn accurate representations for extended horizons. One indication of this is that, while the model performs better with an increased prediction horizon in the *simple* case, it shows markedly poorer performance in the *hard* scenario. In the *hard* case with $T = 30$, the scene

Simple Case			**Hard Case**		
Context Horizon (N)	**Prediction Horizon** (T)	**Vehicle mIoU**	**Context Horizon** (N)	**Prediction Horizon** (T)	**Vehicle mIoU**
5	10	0.539	5	10	0.353
10	10	0.624	10	10	**0.440**
20	10	0.707	20	10	0.435
20	30	**0.737**	20	30	0.369

Table 4.2: **World forward model evaluation results.** We only show the vehicle mIoU for brevity. For the rest, see the Appendix A.1.

undergoes such drastic changes that the model struggles to reconstruct it using only the motion cues present in the context frames. Finally, as in the case of *simple* case, the model trained for *hard* case suffers from the lack of sufficient context when $N = 5$, which causes a relative performance drop compared to the $N = 10$ and $N = 20$ cases.

Qualitative Results. Figure 4.6 shows three examples from the world forward model predictions. *Context-prediction* pairs enclosed in a green rectangle show the successful predictions. In the first two successful rows, it is visible that the model is able to reconstruct the future world state with the help of the motion information encapsulated in the context frames. However, the last row shows an example where the model failed. In general, we observe that the model tends to fail when the ego vehicle turns. We argue that it is because we underrepresented the *turning* situations in the dataset since, most of the time, the ego vehicle follows a rectilinear motion, not curvilinear.

4.4 Conclusion

All in all, we show that the proposed ego forward model can predict the future ego location with an error of less than 10 *cm*. We also show that the proposed world

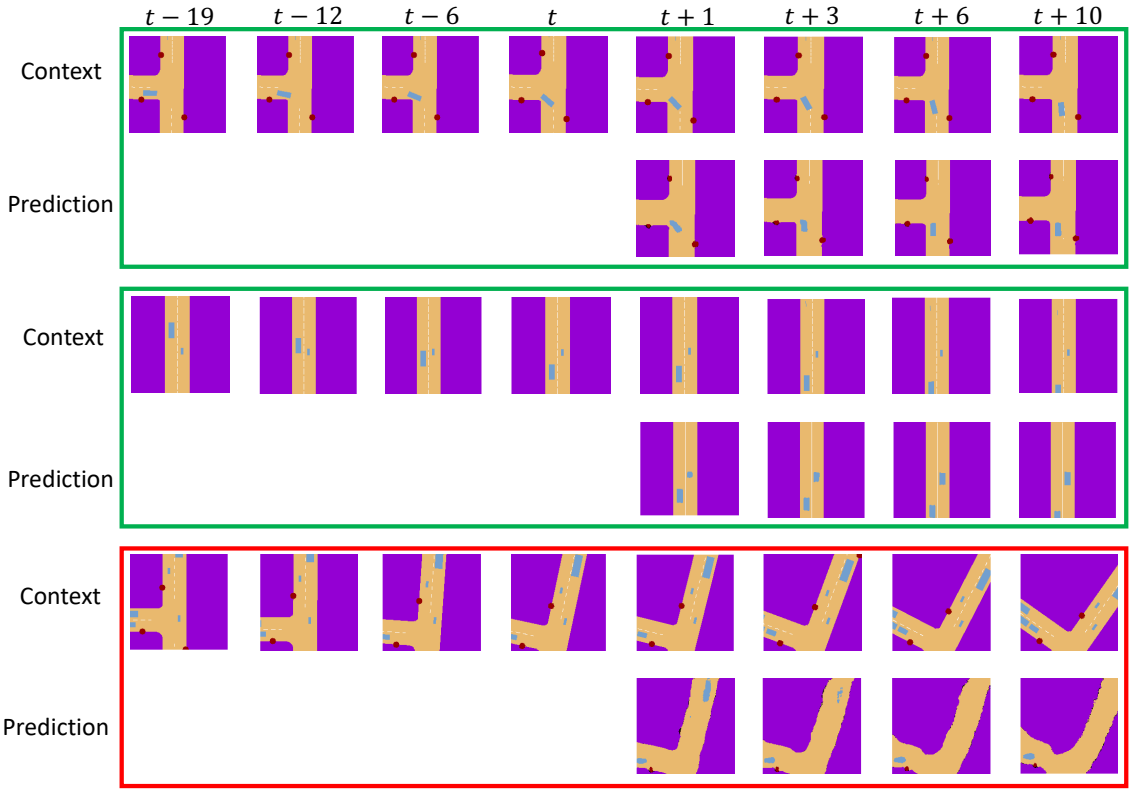

Figure 4.6: **World forward model predictions.** The first four frames in the *context* row represent the context frames fed to the model as input. The next four frames in the context row represent the ground-truth future world states. Finally, the predictions of the world forward model are located in the *prediction* row. *Context-prediction* pairs enclosed in a green rectangle show the successful predictions. On the other hand, a *context-prediction* pair enclosed in a red rectangle displays a case where the model failed. Note that the time steps are downsampled to fit the figure.

forward model can predict the future world state with a high (> 0.7) vehicle mIoU score for *simple* case and a moderate (> 0.4) vehicle mIoU score for *hard* case. We believe that the proposed forward models can be used as a baseline for future prediction in urban driving scenarios.

Limitations and Future Work. One limitation of our proposed ego forward model is that it might be too simple for edge cases. Since it is based on the *kinematic* bicycle model, it does not cover all the situations a vehicle might encounter. For example, when the speed of the vehicle is too large or the given control command is too jerky, the kinematic bicycle model loses its validity. In these cases, it might

not be able to reconstruct the future ego state correctly. As a solution, we can use a more complex model such as a dynamic bicycle model [Rajamani, 2012] or a neural network-based parametric model.

Additionally, one important limitation of our proposed world forward model is the fixed-size constraint on the context frames. Unfortunately, this limitation is a direct cause of the architectural choice. Especially, it becomes a problem if we do not have enough frames to fill the context buffer. In this case, one solution is to fill the buffer with the same frame multiple times. However, it might also cause catastrophic problems at edge cases or while working with a low sampling rate (i.e., 5 Hz). As a solution, in the future, we plan to use a more flexible recurrent neural network-based (RNN) architectures ([Denton and Fergus, 2018],[Franceschi et al., 2020]) that can handle variable-length context and prediction horizons. Another limitation of our world forward model is that it shows a degradation in performance when the ego vehicle starts to turn. We argue that it is because the *curvilinear* motion lacks enough presence in the dataset, compared to the *rectilinear* motion. In the future, we plan to use a more balanced dataset to rectify this problem.

Chapter 5

LEARNING HOW TO ACT

In this chapter, we present our work on learning how to act in urban driving scenarios. In Section 5.1, we explain our motivation to use existing methods in this chapter. Then, in Section 5.2, we present our methodology to learn how to act in urban driving scenarios. Specifically, we first show how we design a differentiable cost module that injects *driving*-specific supervision signals into the learning process. Then, we show how we use the differentiable cost module to incorporate a gradient-based MPC module and train a policy network. In Section 5.3, we present and discuss the evaluation results of our method. Finally, in Section 5.4, we conclude our work and discuss the limitations and future work.

5.1 Introduction

Although the behavior cloning is a widely-adopted method to learn a *policy* in an urban-driving setup, it has several limitations. One notable limitation of behavior cloning approach is that it is susceptible to a phenomenon called *covariate shift* [Ross et al., 2011]. In the context of behavior cloning, we define the covariate shift as the inability of agent to recover from unseen states. One of the main reasons for the covariate shift is that the agent does not have any information about the *cost* of being in a particular state since it only sees a fixed set of states in a dataset. It also embarks another problem with the BC: strong dependence on the expert supervision, which might not always be available.

To deal with the aforementioned limitations, we propose a differentiable policy cost module that can inject *driving*-specific supervision signals into the learning process. The main purpose of the module is to give ability to *learn* how to drive –not *memorize*– by providing constant *on-policy* supervision signals. We design the cost

module to be differentiable for use in gradient-based optimization and supervised learning methods, avoiding RL's sample-inefficiency.

5.2 Methods

5.2.1 Differentiable Policy Cost Design for Urban Driving

We design a differentiable cost module that injects *driving*-specific supervision signals into the learning process, as opposed to the naive behavior cloning scenario where the only supervision signal is the action or state reconstruction term that minimizes the mathematical objective, not physical. We design the cost module to be differentiable for use in gradient-based optimization methods. Besides its differentiability, we also make it interpretable for detecting failure modes of the learning process.

Our cost module depends heavily on the prediction module as it mainly operates on the future predictions of the *world* and *ego* states. The main idea is to point out the *driving* continuously-specific penalties and rewards in the form of spatial and temporal regions. That is, the module assigns a cost for being in a particular ego state $\mathbf{s}^{ego}$ in the future, given that the world state $\mathbf{s}^{world}$ in the future. In this regard, our design is similar to that in [Sobal et al., 2022]. However, they design their cost module to operate in *highway* driving, which is a relatively more straightforward task compared to *urban* driving. Our cost module operates in highly challenging and stochastic urban driving scenarios.

Mathematical Formulation of the Cost Module

Our formulation assumes that we have the information of the future states of *world* and *ego*. To begin with, we represent the ego vehicle as a heatmap $\mathbf{H}^{ego}$ in the BEV representation, which we interpret as an area of impact of the ego vehicle's location in the BEV space. We do not focus on a singular point representing the exact location but rather a proximity region around the exact location. Then, considering each channel of BEV representation as a binary map $\mathbf{M}^{class_i}$ that represents a specific semantic class, a dot product between the heatmap and the binary map

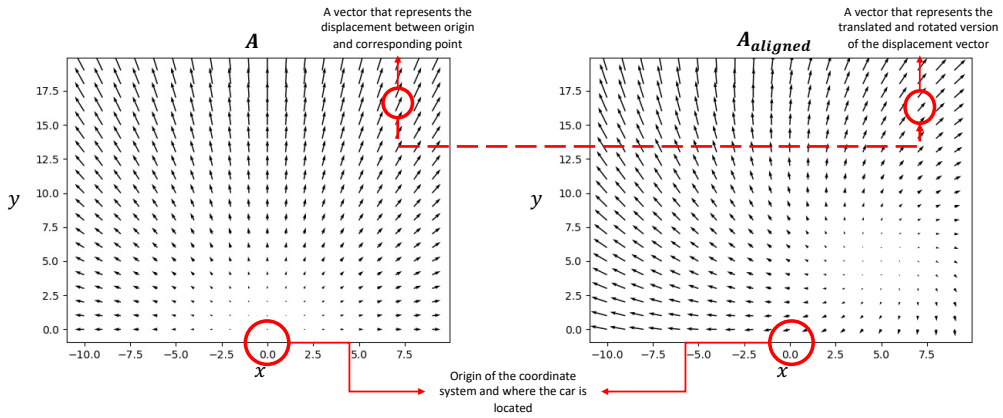

Figure 5.1: **Coordinate matrix A and aligned coordinate matrix $\mathbf{A}^{aligned}$.** The bottom-middle point corresponds to the origin of the coordinate system. Each point in the coordinate matrix $\mathbf{A}$ represents the displacement vector between the origin and the corresponding point. Finally, we obtain the aligned coordinate matrix $\mathbf{A}^{aligned}$ by translating and rotating the elements of the coordinate matrix according to the ego vehicle's location and rotation.

$(\langle \mathbf{H}^{ego}, \mathbf{M}^{class_i} \rangle)$ represents the amount of *overlap* between the ego vehicle and the semantic class. This specific cost definition physically induces a penalty for being in an undesirable location and a reward for being in a desirable location.

Calculation of the Heatmap. The calculation of the heatmap is one of the most critical aspects of the differentiable cost module since it enables gradients to propagate through the physical state (i.e., location, speed, heading) of the ego vehicle. We first start with creating a matrix of coordinates where each element represents the displacement vector between the bottom-middle pixel and the corresponding pixel. The coordinate matrix can be denoted as $\mathbf{A} \in \mathbb{R}^{H \times W \times 2}$, where H and W are the height and width of the BEV representation, respectively. For each element in the coordinate matrix, we calculate the displacement vector as follows:

$$\mathbf{A}_{i,j} = \begin{bmatrix} i - \frac{H}{2} \\ j - W \end{bmatrix} \tag{5.1}$$

Then, we obtain the aligned coordinate matrix $\mathbf{A}^{aligned}$ by translating and rotating the elements of the coordinate matrix according to the ego vehicle's location

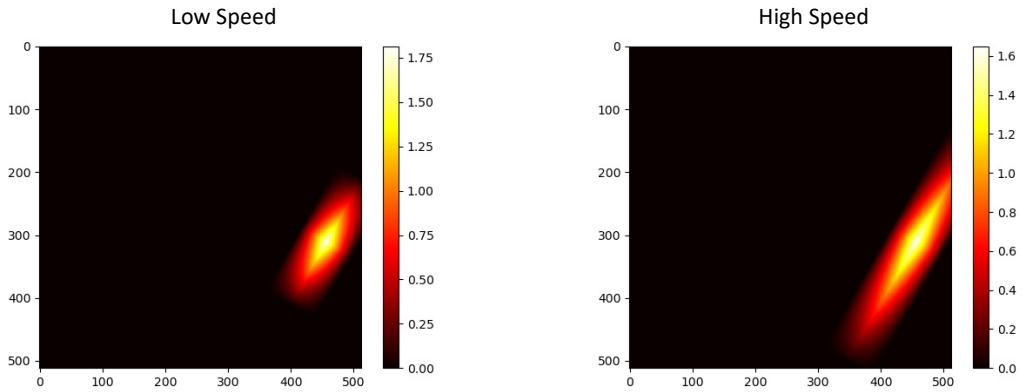

Figure 5.2: **The example heatmaps generated at different speeds.** We generate the heatmaps according to the (5.3), which defines a non-linear transform of the coordinate matrix $\mathbf{A}^{aligned}$. In the high-speed case, the heatmap has a larger area of impact, which can be interpreted as a larger look-ahead distance.

and rotation:

$$\mathbf{A}_{i,j}^{aligned} = \mathbf{R}(\psi) \left(\mathbf{A}_{i,j} - \begin{bmatrix} x \\ y \end{bmatrix} \right) \tag{5.2}$$

where x and y are the location of the ego vehicle, and ψ is the yaw angle of the ego vehicle. $\mathbf{R}(\psi)$ represents the rotation matrix that rotates the elements of the coordinate matrix $\mathbf{A}$ by ψ radians. Figure 5.1 illustrates $\mathbf{A}$ and $\mathbf{A}^{aligned}$.

Then, following the [Sobal et al., 2022] and [Henaff et al., 2019], we calculate the heatmap $\mathbf{H}^{ego}$ as follows:

$$\mathbf{H}_{i,j}^{ego} = \left[\left(\frac{d_x - \left| \mathbf{A}_{i,j,1}^{aligned} \right|}{d_x - l/2} \right)^{+} \times \min \left(\left(\frac{d_y - \left| \mathbf{A}_{i,j,2}^{aligned} \right|}{d_y - w/2} \right)^{+}, 1 \right) \right]^{\alpha} \tag{5.3}$$

where d_x and d_y are masks behaving as safety distances in our equations. Heuristically:

$$d_x = (k_{lat} w + b_{lat}) + \left(k_{lat}^{speed} v + b_{lat}^{speed} \right) \tag{5.4}$$

$$d_y = (k_{long} l + b_{long}) + \left(k_{long}^{speed} v + b_{long}^{speed} \right) \tag{5.5}$$

In (5.4) and (5.5), all $k_{[.]}$ and $b_{[.]}$ terms represent a scaler and an offset parameter that should be tuned. Moreover, w and l represent the width and length of the ego vehicle, respectively. Finally, α is a hyperparameter that controls the sharpness of the heatmap. Figure 5.2 illustrates the example heatmaps generated at different speeds.

We define a per class policy cost term $C^{policy}_{class_i}$ as follows:

$$C^{policy}_{class_i} = \langle \mathbf{H}^{ego}, \mathbf{M}^{class_i} \rangle \tag{5.6}$$

Finally, we can define the overall cost term C^{policy} as follows:

$$C^{policy} = \sum_{i=1}^{K} W_{class_i} C^{policy}_{channel_i} \tag{5.7}$$

where K is the number of semantic classes in the BEV representation and W_{class_i} is a weighting factor to for each semantic class. Depending on the situation, W_{class_i} can be a constant value, a normal distribution parameterized by mean μ and σ^2, or a function of the ego state $\mathbf{s}^{ego}$ and the world state $\mathbf{s}^{world}$.

5.2.2 Gradient-Based Model Predictive Control

In traditional model predictive control (MPC) formulation, we need to solve the finite-horizon optimization problem:

$$\underset{\mathbf{x}_{0:T-1} \in \mathcal{X}, \mathbf{a}_{0:T-1} \in \mathcal{A}}{\operatorname{argmin}} \sum_{t=0}^{T-1} c_t(\mathbf{x}_t, \mathbf{a}_t) \quad \text{subject to} \quad \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{a}_t), \quad \mathbf{x}_0 = x_{\text{init}}, \tag{5.8}$$

where $\mathbf{x}_t, \mathbf{a}_t$ are the state and control at time t, $\mathcal{X}$ and $\mathcal{A}$ are state and control constraints, $c_t : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is a generic cost function, $f : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$ is a dynamics model, and x_{init} is the initial state of the system. Under specific assumptions, we can solve the traditional MPC problem by using a variety of methods, such as quadratic programming [Maciejowski, 2002], nonlinear programming [Camacho and Alba, 2013], explicit MPC [Bemporad et al., 2002], and, iterative Linear Quadratic Regulator (iLQR) [Li and Todorov, 2004].

We aim to have a more versatile MPC formulation that can be utilized in various scenarios rather than adapting our problem to existing definitions. To this end, we construct our gradient-based planner (MPC) by following a similar strategy to that of [Henaff et al., 2018], [Srinivas et al., 2018], [Bharadhwaj et al., 2020].

Mathematical Formulation of Gradient-Based MPC

The main ingredients of the gradient-based MPC are the *differentiable* cost function $\mathbb{C}$ and the forward model f. We start with predicting the future states of ego and world by utilizing the forward models f_{ego} and f_{world}. In each time step, the ego forward model takes the current ego state $\mathbf{s}_t^{ego} \in \mathbb{R}^4$, and the action $\mathbf{a}_t \in \mathbb{R}^2$ as input and outputs the predicted ego state $\hat{\mathbf{s}}_t^{ego} \in \mathbb{R}^4$. Similarly, the world forward model takes the past N world states $\mathbf{s}_{t-N-1:t}^{world} \in \mathbb{R}^{N \times C \times H \times W}$ as input and outputs the predicted future world state $\hat{\mathbf{s}}_{t+1}^{world} \in \mathbb{R}^{1 \times C \times H \times W}$.

Then, according to our problem definition, the cost function C can be defined as follows:

$$C = \sum_{t=1}^{T} C_t \left(\hat{\mathbf{s}}_{t+1}^{ego}, \hat{\mathbf{s}}_{t+1}^{world}, \mathbf{s}_{t+1}^{target} \right) \tag{5.9}$$

$$C_t \left(\hat{\mathbf{s}}_{t+1}^{ego}, \hat{\mathbf{s}}_{t+1}^{world}, \mathbf{s}_{t+1}^{target} \right) = C_t^{policy} \left(\hat{\mathbf{s}}_{t+1}^{ego}, \hat{\mathbf{s}}_{t+1}^{world} \right) + \left|\left| \hat{\mathbf{s}^{ego}}_{t+1}, \mathbf{s}_{t+1}^{target} \right|\right|_1 \tag{5.10}$$

where C_t^{policy} represents the policy cost defined in 5.2.1 and the second term represents the $L1$ loss between ego state and the target state.

In each optimization step, we minimize the unrolled optimization objective (5.9) with respect to the actions $\mathbf{a}_{t:t+T-1}$ over a fixed horizon T in a sliding window fashion:

$$\mathbf{a}_{t:t+T-1} = \underset{\mathbf{a}_{t:t+T-1}}{\operatorname{argmin}} C \tag{5.11}$$

We can solve the optimization problem (5.11) by utilizing a gradient descent method. That is, in each optimization iteration, we calculate the gradients of the

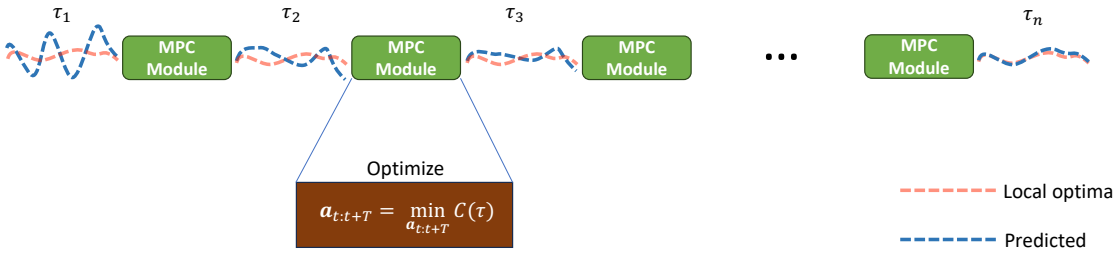

Figure 5.3: **Simple schematic demonstrating the working mechanism of the gradient-based model predictive control module.** In each optimization iteration, the proposed set of action $\mathbf{a}_{t:t+T}$ yields a trajectory τ. Here, we assume that the τ is a combination of both $\mathbf{s}^{ego}$ and $\mathbf{s}^{world}$. The objective is to optimize the actions to minimize the cost C defined in (5.10). Note that the optimization (τ) iteration is independent of the time (t) iteration in the sense that after doing n optimization iteration, we increase the time step by one to calculate the next action.

cost function C with respect to the actions $\mathbf{a}_{t:t+T-1}$. We can execute the gradient calculations by utilizing an automatic differentiation tool such as PyTorch [Paszke et al., 2019]. Then, we update the actions $\mathbf{a}_{t:t+T-1}$ according to the gradients:

$$\mathbf{a}_{t:t+T-1} \leftarrow \mathbf{a}_{t:t+T-1} - \alpha \nabla_{\mathbf{a}_{t:t+T-1}} C \tag{5.12}$$

We repeat the aforementioned action optimization procedure for n iterations. Finally, as in the case of traditional MPC algorithms, we only execute the first action $\mathbf{a}_t$ and discard the remaining actions. We repeat the optimization procedure for the next time step until the *high-level* objective is completed. We describe the detailed implementation procedure in Algorithm 5.2.2 and illustrate in Figure 5.3.

Algorithm 1 Gradient-Based Model Predictive Control

Input: $(f_{ego},\ f_{world},\ \mathbf{s}_t^{ego},\ \mathbf{s}_{t-N-1:t}^{world},\ \mathbf{s}_t^{target},\ C_t,\ T,\ n,\ \alpha)$
Output: Action $\mathbf{a}_t$, Cost C
procedure OPTIMIZE ▷ Optimize actions $\mathbf{a}_{t:t+T-1}$
Initialize actions $\mathbf{a}_{t:t+T-1}$ randomly over prediction horizon T
    for iteration in 1 to n **do**
        ▷ Predict future ego states
        $\hat{\mathbf{s}}_{t+1:t+T}^{ego} \leftarrow f_{ego}\left(\mathbf{s}_t^{ego}, \mathbf{a}_{t:t+T-1}\right)$
        ▷ Predict future world states
        $\hat{\mathbf{s}}_{t+1:t+T}^{world} \leftarrow f_{world}\left(\mathbf{s}_{t-N-1:t}^{world}, \mathbf{a}_{t:t+T-1}\right)$
        ▷ Calculate cost
        $C \leftarrow C_t\left(\hat{\mathbf{s}}_{t+1:t+T}^{ego}, \hat{\mathbf{s}}_{t+1:t+T}^{world}, \mathbf{s}_{t+1:t+T}^{target}\right)$
        ▷ Calculate gradients
        $\nabla_{\mathbf{a}_{t:t+T-1}} C \leftarrow \text{AUTOGRAD}(C)$
        ▷ Update actions
        $\mathbf{a}_{t:t+T-1} \leftarrow \mathbf{a}_{t:t+T-1} - \alpha \nabla_{\mathbf{a}_{t:t+T-1}} C$
    end for
    return $\mathbf{a}_t$, C
end procedure

5.2.3 Policy Network

We follow a relatively more straightforward approach while constructing the policy network, as our primary focus is to prove the effectiveness of the differentiable cost module proposed in Section 5.2.1. To this end, we construct a simple policy network π that takes the current ego state $\mathbf{s}_t^{ego}$, the target state $\mathbf{s}_t^{target}$, occupancy information obtained by utilizing a radar-based sensor module in CARLA, and future-aware world features obtained by the pretrained world forward model. Figure 5.5 illustrates the general diagram of the policy network π.

Occupancy Sensor. We design a hand-tailored radar-based occupancy sensor in CARLA to assess information related to the objects in the proximity of the ego

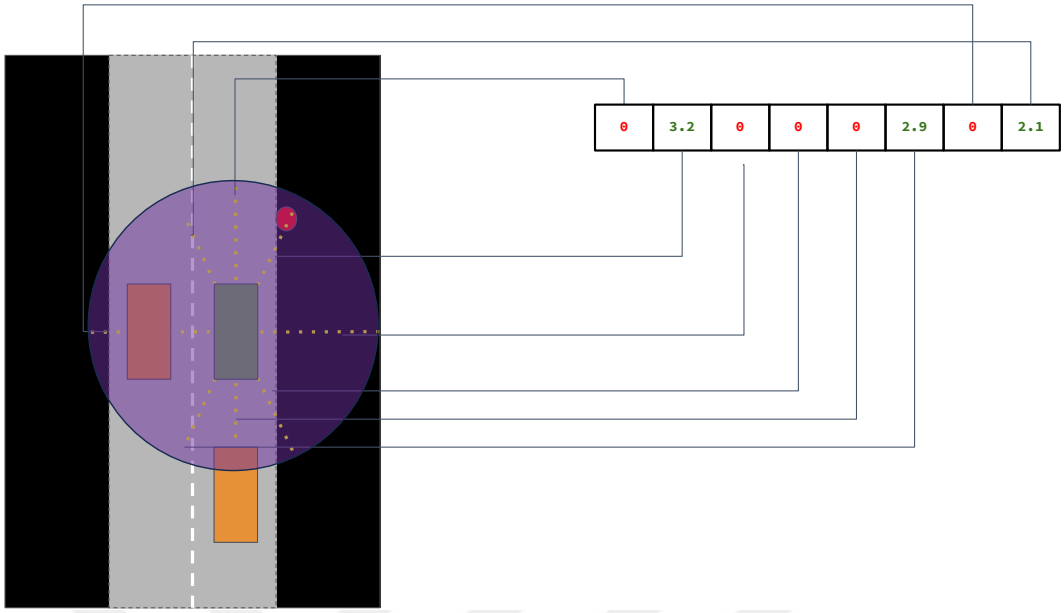

Figure 5.4: **An illustration of the occupancy sensor designed in CARLA.** The green rectangle represents the ego vehicle that the radars are mounted on. Orange rectangles represent the other vehicles that might be in the proximity of the ego vehicle. Yellow dotted lines represent the radar sensors. Finally, the big purple circle represents the working range of the radar sensors. If a radar does not detect any object in its working range, the corresponding sensor reading is set to 0. Otherwise, the sensor reading is set to the distance between the ego vehicle and the detected object.

vehicle. We place n radar sensors around the ego vehicle. Each radar sensor has 5 degrees of vertical and horizontal field of view. Each radar has a range of 10 meters. We only utilized the distance information from the radar sensors. Figure 5.4 visualizes the sensor reading $\mathbf{o} \in \mathbb{R}^n$.

Training and Inference Procedure

We first concatenate each component of the policy network to form a single vector $\mathbf{v} \in \mathbb{R}^D$. $\mathbf{v}$ consists of the current ego state $\mathbf{s}_t^{ego} \in \mathbb{R}^4$, the target state $\mathbf{s}_t^{target} \in \mathbb{R}^2$, occupancy information $\mathbf{o}_t \in \mathbb{R}^n$, and future-aware world features $\mathbf{f}_t^{world} \in \mathbb{R}^{D_f}$. Ego state $\mathbf{s}_t^{ego}$ is a vector including the location x and y coordinates, speed v, and heading ψ of the ego vehicle. Moreover, target state $\mathbf{s}_t^{target}$ is a vector whose elements are the location x and y coordinates of the target location. Occupancy information $\mathbf{o}_t$

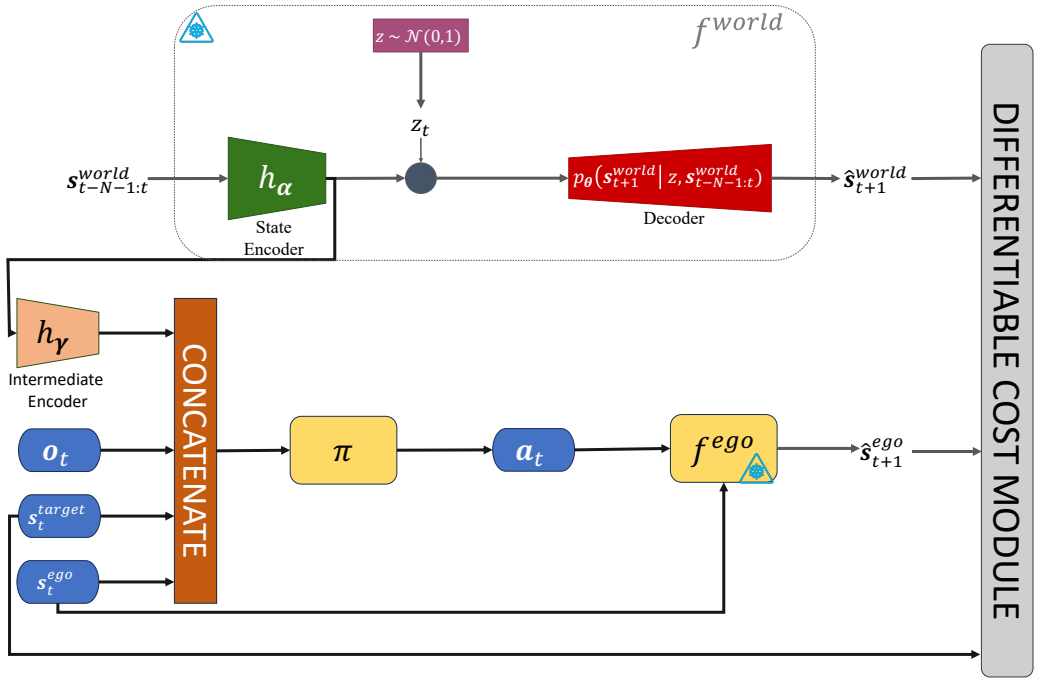

Figure 5.5: **A schematic of the policy network.** The policy network π takes the current ego state $\mathbf{s}_t^{ego}$, the target state $\mathbf{s}_t^{target}$, occupancy information obtained by utilizing a radar-based sensor module in CARLA, and future-aware world features obtained by the pretrained world forward model. Then, the policy network π outputs the action $\mathbf{a}_t$. We train the policy network π to minimize the cost C without changing the parameters of the forward models.

is a vector that represents the objects in the proximity of the ego vehicle. Finally, future-aware world features $\mathbf{f}_t^{world}$ is the output of the state encoder of the world forward model. We process the concatenated vector $\mathbf{v} = \left[\mathbf{s}_t^{ego}, \mathbf{s}_t^{target}, \mathbf{o}_t, \mathbf{f}_t^{world}\right]$ by a multi-layer perceptron (MLP) to obtain the predicted action $\hat{\mathbf{a}}_t \in \mathbb{R}^2$:

$$\hat{\mathbf{a}}_t = \pi(\mathbf{v}) \tag{5.13}$$

In turn, we use the predicted action to predict the next future ego state $\hat{\mathbf{s}}_{t+1}^{ego}$ with the help of the ego forward model f^{ego}. Then, the policy network π uses the predicted future ego state to estimate action in the next time step. We repeat this procedure T times to obtain the predicted trajectory τ:

$$\tau = \{\hat{\mathbf{a}}_t, \hat{\mathbf{a}}_{t+1}, \ldots, \hat{\mathbf{a}}_{t+T-1}\} \tag{5.14}$$

Since we decouple the predicted future world state from the ego state, we can predict the future world states independently by incorporating the world forward model f^{world}. Once we have the unrolled trajectory τ (i.e., predicted the future world and ego states for T time steps), we can calculate the cost C as follows:

$$C = \sum_{t=1}^{T} C_t \left(\hat{\mathbf{s}}_{t+1}^{ego}, \hat{\mathbf{s}}_{t+1}^{world}, \mathbf{s}_{t+1}^{target}\right) \tag{5.15}$$

Optionally, (5.15) might include a behavior cloning (BC) objective such that it becomes:

$$C_{BC} = \sum_{t=1}^{T} C_t \left(\hat{\mathbf{s}}_{t+1}^{ego}, \hat{\mathbf{s}}_{t+1}^{world}, \mathbf{s}_{t+1}^{target}\right) + ||\mathbf{a}_t - \hat{\mathbf{a}}_t||_2 \tag{5.16}$$

where the second term in the summation corresponds to $L2$ loss between ground truth and predicted actions.

Finally, we train the policy network π to minimize the cost C or C_{BC}:

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \left[C \textit{ or } C_{BC}\right] \tag{5.17}$$

Note that the ego forward model f^{ego} and the world forward model f^{world} are frozen in this policy network training pipeline. In other words, we train the policy network π to minimize the cost C without changing the parameters of the forward models.

During inference, we concatenate the current ego state $\mathbf{s}_t^{ego}$, the target state $\mathbf{s}_t^{target}$, occupancy information (see Section 5.2.3) obtained by utilizing a radar-based sensor module in CARLA, and future-aware world features obtained by the pretrained world forward model. Then, the policy network π processes the concatenated vector to calculate the action $\mathbf{a}_t$.

5.2.4 Evaluation Procedure

We test the efficacy of the proposed gradient-based MPC module and policy network module with a custom environment created in the CARLA simulator. In each trial, we sample a start and an end point to generate a downsampled route, using the A^* algorithm, for the ego vehicle. In this context, a route is a set of intermediate waypoint and high-level navigational command tuples, which corresponds to the target state $\mathbf{s}_t^{target}$ delineated in Section 3.1. Then, we place the ego vehicle at the start point of the route and start the simulation. In each time step, we calculate the action $\mathbf{a}_t$ by utilizing either policy network π or gradient-based MPC. Then, we feed the action $\mathbf{a}_t$ to the agent in the CARLA simulator. We repeat this procedure until the ego vehicle reaches the end point of the route. We halt the trial if the ego vehicle collides with any other static or dynamic object in the environment.

For MPC, we use the prediction horizon $T = 10$ and the optimization steps $n = 40$. We use the Adam optimizer [Kingma and Ba, 2017] with a learning rate of $\alpha = 0.01$. For the policy network, we use the behavior cloning loss C_{BC}.

In our preliminary investigations, we find that the gradient-based MPC module works reasonably well without using any world forward model and only by repeating the last frame as T times. However, it works under the assumption that the environment runs at a high sampling rate (e.g., 20 Hz) and the rate of change between frames is relatively low. The advantage of not using the world forward model is that we reduce the computational budget of the gradient-based MPC module, which is already high due to the iterative nature of the optimization procedure. Therefore, in *most* of our investigations, we disable the world forward model in the gradient-based MPC module for the sake of computational efficiency.

In the next section, we present our qualitative findings that we predicate on the manual inspections throughout the trials.

5.3 Results and Discussion

In this section, we present our observations regarding the utilization of the gradient-based MPC and the policy network in various scenarios.

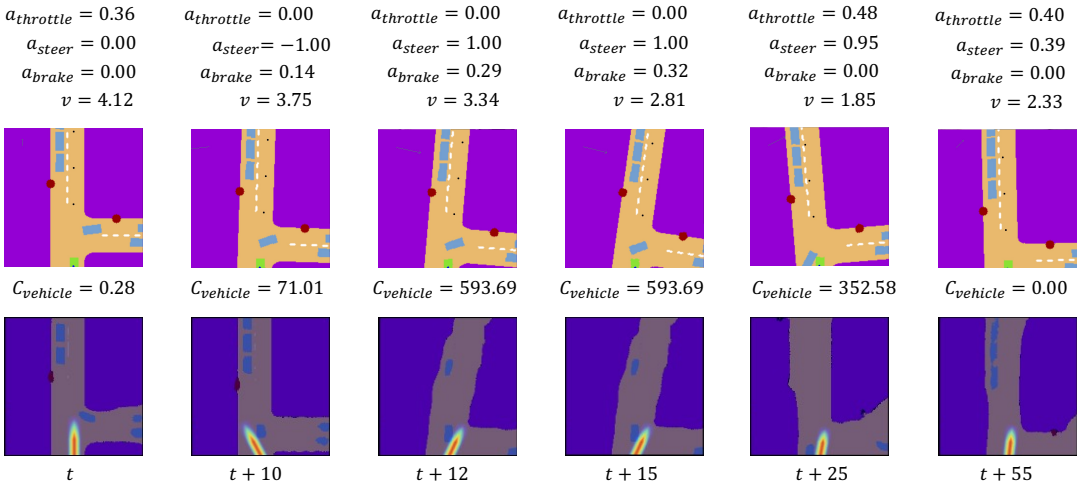

Figure 5.6: **A timelapse from a scene where the ego vehicle makes a sudden maneuver to avoid a collision. World forward model in action.** We represent the ego vehicle with a green rectangle at the bottom of the image. On the bottom row, we show the generated heatmaps used in the differentiable cost calculation. In this policy network example, we utilize the world forward model, so the predicted future world states are visible under the heatmap. The indices at the bottom show the amount of time step increment since the first frame, where the time difference between time steps is $\Delta t = 0.05$ *sec*. The agent adjusts its steering based on the vehicle cost and future world state predictions, initially steering left before correcting to the right to avoid collisions and then proceeding forward when safe.

5.3.1 Gradient-Based MPC

In general, the proposed gradient-based MPC agent can follow and complete a route and avoid collisions with other traffic participants. Moreover, it can act based on the future predictions that the forward models provide.

Figure 5.6 represents a timelapse from a scene where the ego vehicle makes a sudden maneuver to avoid collision with another vehicle, with the help of the forward models. Note that, in Figure 5.6, the bottom row displays the heatmaps generated by the differentiable cost module and the future world state predictions of the world forward model. Therefore, we can observe that the bottom row represents a future world state, hence scenes at the top and bottom rows are not identical. In the first column of Figure 5.6, the agent continues to move forward since the vehicle cost is considerably small. In the second column, the agent applies a sudden steer command to the left since it thinks that it will cost less based on the current situation. Then,

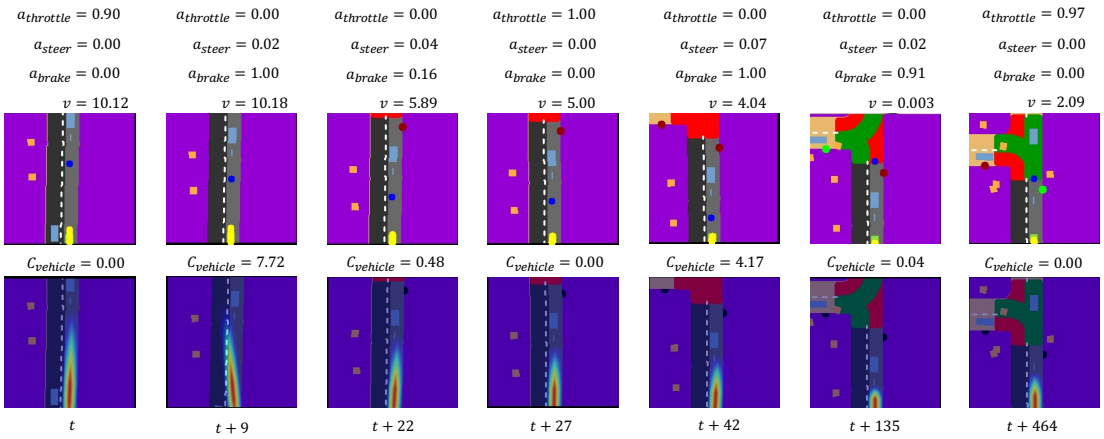

Figure 5.7: **A timelapse from a scene where the ego vehicle slows down to avoid a collision with another vehicle.** We represent the ego vehicle with a green rectangle at the bottom of the image, which is partially occluded by yellow dots that represent the predicted future locations of the ego vehicle. On the bottom row, we show the generated heatmaps used in the differentiable cost calculation. The indices at the bottom show the amount of time step increment since the first frame, where the time difference between time steps is $\Delta t = 0.05$ *sec*. The agent modulates its speed based on the vehicle cost, braking when proximity to other vehicles increases, and accelerating once it is safe to proceed.

from the third to the fifth column, the world forward model starts to predict the other vehicles more accurately, so the agent decides to apply a steering command to the right to avoid collision. Finally, in the sixth column, the agent continues to move forward as the vehicle cost is zero.

Moreover, Figure 5.7 demonstrates a similar scenario where the agent needs to slow down to avoid collision with other vehicles. In this example, the agent can slow down and maintain a safe distance from other vehicles by utilizing the vehicle cost $C_{vehicle}$. In the first column, the agent applies a throttle command to increase speed, as the vehicle cost $C_{vehicle}$ is zero, indicating that the road is safe to move forward without any intervention. In the second column, the vehicle cost increases as the distance between the ego vehicle and other vehicles decreases. As a result, the agent decides to apply a brake command. From the third to the sixth column, the agent continues applying the brake command to reduce speed and lower the vehicle cost. Finally, in the seventh column, it becomes apparent that the agent decides to apply

the throttle command once the distance between itself and other vehicles increases to a level where the vehicle cost is near zero. Note that we can change the spread of safe distances by tuning the parameters of the (5.4) and (5.5). Especially, in Figure 5.7, we can easily observe that the agent can apply a throttle command as long as it thinks that it does not collide with other vehicles and it does not exceed the safe distance.

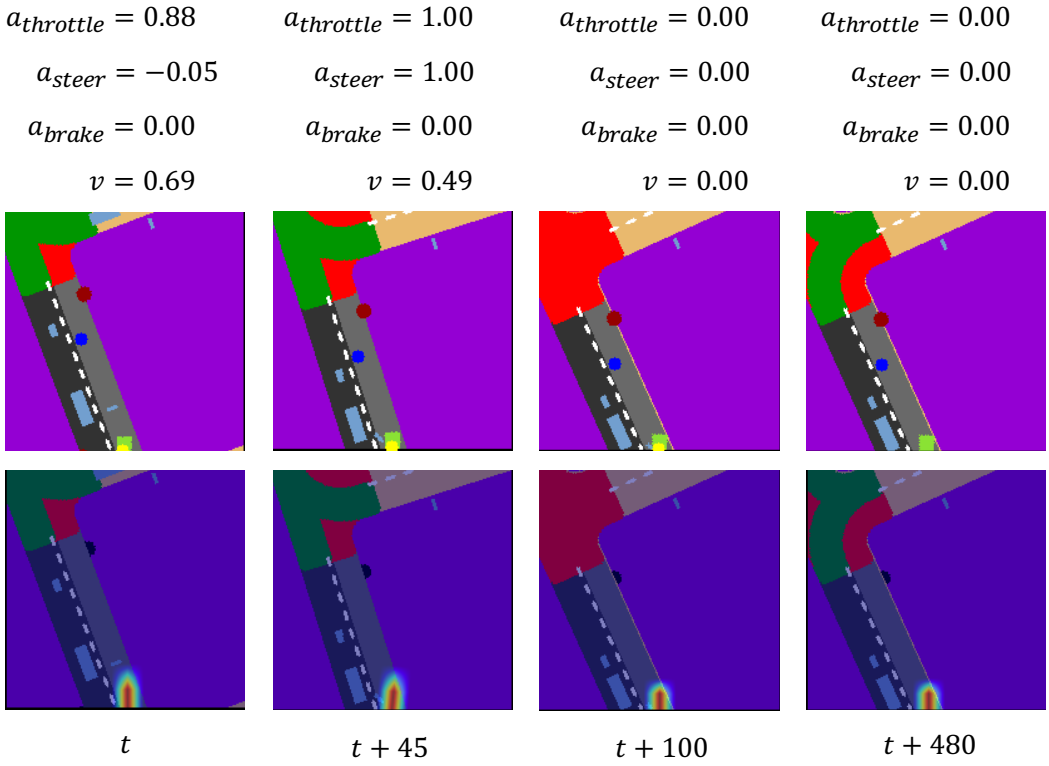

Figure 5.8: **A timelapse from a scene where the ego vehicle gets stuck at local optima.** We represent the ego vehicle with a green rectangle at the bottom of the image, which is partially occluded by yellow dots that represent the predicted future locations of the ego vehicle. On the bottom row, we show the generated heatmaps used in the differentiable cost calculation. The indices at the bottom show the amount of time step increment since the first frame, where the time difference between time steps is $\Delta t = 0.05$ *sec*. The agent gets trapped in a local optimum due to conflicting cost evaluations from the MPC module, resulting in repetitive braking despite initial intentions to progress forward.

The Local Optima Problem. Although the gradient-based MPC agent can perform a *decent* driving in most of the randomly generated scenarios, and it drives extremely carefully around other traffic elements, it shows a highly undesirable behavior in certain situations. It can get stuck at local optima, which is already known to be a common problem in gradient-based planning [Bharadhwaj et al., 2020]. Figure 5.8 shows an example timelapse where the agent gets stuck at the local optima. In this specific example, the agent cannot escape from the local optima since every action predicted by the gradient-based MPC module creates both negative and positive costs that cancel out each other. For instance, even though the agent issues a throttle and a steering command to move a little bit forward toward the target waypoint, the differentiable cost module indicates that it will increase the offroad cost. Therefore, it decides to apply a brake command. Then, the same procedure repeats itself, causing the agent to get stuck at the local optima.

5.3.2 Policy Network

The proposed policy network agent can perform actions to follow a lane and avoid collisions with other dynamic and static elements in the traffic. Additionally, the future-aware features provided by the world forward model can help the agent to make more informed decisions, as it can contribute as a look-ahead mechanism to the policy network.

Figure 5.9 demonstrates a timelapse from a scene where the ego vehicle turns around a corner. In this example, we observe that the agent can maintain a certain speed while turning a corner and avoid crossing the opposite lane. Moreover, we demonstrate that the agent can maintain its radius of curvature by preserving the steering command in a certain range while turning. Finally, world forward model predictions at the bottom row indicate that the agent predicts the future world states accurately.

Dependence on Expert Supervision. Although, initially, we hypothesize that the policy network agent can learn how to drive without any expert supervision, our experiments show that the policy network agent needs a certain amount of expert

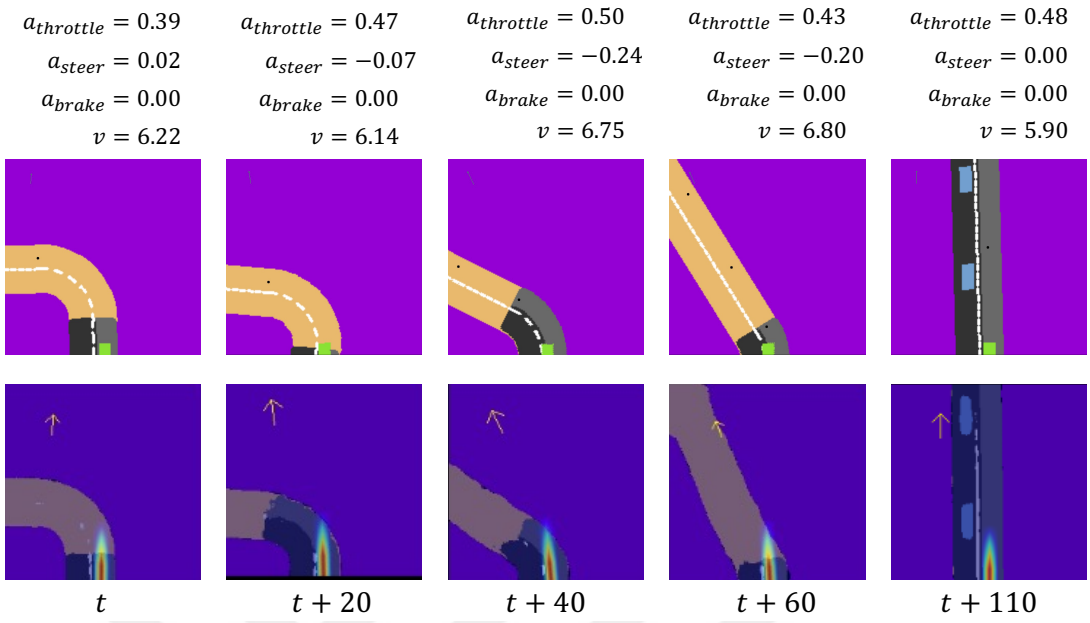

Figure 5.9: **A timelapse from a scene where the ego vehicle turns around a corner.** We represent the ego vehicle with a green rectangle at the bottom of the image. At the bottom row, we present the generated heatmaps used in the differentiable cost calculation. In this policy network example, we utilize the world forward model, so the predicted future world states are visible under the heatmap. The indices at the bottom show the amount of time step increment since the first frame, where the time difference between time steps is $\Delta t = 0.05$ *sec*. The agent adeptly maintains speed and steering during turns, avoids crossing into the opposite lane, and accurately predicts future world states based on the forward model.

supervision to learn how to drive. In other words, the policy network agent cannot learn how to drive without any expert supervision. We observe that the agent only learns or converges to one *action* when the expert action supervision is missing. We argue that it is a direct outcome of our optimization scheme in the context of policy learning. In other words, as opposed to the multi-step-refinement nature of the gradient-based MPC, we claim that the one-step-refinement nature of the policy network is not enough for an agent to acquire driving skills, provided that the only optimization objective is the differentiable cost function proposed in (5.15).

5.4 Conclusion

All in all, we propose a differentiable policy cost module that we can use in conjunction with the gradient-based MPC and policy network to equip an agent necessary driving skills. We show that both methods can learn to perform basic tasks in driving such as following a lane, turning around a corner, and avoiding collisions with other traffic participants.

Limitations and Future Work. We observe that the gradient-based MPC agent can get stuck at local optima in certain situations. Even though it is a known drawback of gradient-based planners [Bharadhwaj et al., 2020], we argue that it is a direct outcome of the set of weighting factors $W = (W_{pedestrian}, W_{vehicle}, \ldots, W_{offroad})$ that we use with differentiable policy cost module. We observe that the particular selection of weighting factors affects the agent's *stuck* behavior differently. For instance, while a specific set of weighting factors can lead to a local optima problem in the scenario depicted in Figure 5.8, another set of weighting factors can cause a local optima problem in a different scenario. This observation suggests that we should assign different weighting factors to different scenarios since it is not feasible to find a single set of weighting factors that work well in all scenarios. Therefore, we argue that we should construct a mechanism to assign different weighting factors to different scenarios. In this context, we plan to design a new system where we can learn the weighting factors based on the state of the ego and the world in the future.

On the other hand, we show that the policy network agent cannot acquire the necessary driving ability without expert supervision. As noted in Section 5.3, we think that the policy network cannot learn a sufficient driving policy with only one-step-refinement, under the naive optimization objective that we use in (5.15). Therefore, we plan to investigate the possibility of utilizing a multi-step-refinement scheme in the policy network training pipeline. In other words, as a future work, we want to explore an architecture where we can combine gradient-based MPC with the policy network to train the policy in a multi-step-refinement fashion [Hansen et al., 2022]. In this case, in each step, we can optimize the local objective (i.e., avoid

collision, follow a path, etc.) with a gradient-based MPC module, and optimize the global objective (i.e., reach the target location) with the policy network.

Chapter 6

CONCLUSION

Urban autonomous driving continues to remain at the front line of technological challenges, given the difficulties existing in navigating progressive and uncertain environments. In this thesis, we provide insight into potential advancements in forward models and their usage in differentiable policy cost design, offering a roadmap for the further enhancement of self-driving systems in urban landscapes.

Chapter 4 highlights the crucial role of future prediction in autonomous driving. Our proposed ego and world forward models show promise to accurately predict future states, with the ego model achieving an error of less than 10 cm and the world model showcasing commendable mIoU scores. Nevertheless, these models are not without limitations. The simplicity of the ego forward model, based on the kinematic bicycle model, might not be versatile enough for all the edge cases. Similarly, the world forward model's fixed-size constraint on context frames implies the need for potential improvements, such as integrating more flexible RNN architectures. Moreover, the world forward model shows a degradation in performance when the ego vehicle begins to turn. This is a clear sign that we should construct a more balanced dataset that includes both rectilinear and curvilinear motions to a similar degree.

In Chapter 5, we reveal the potential of the differentiable policy cost module. Integrated with both the gradient-based MPC and policy network, it empowers the agent with essential driving skills, helping in tasks like lane following, making turns, and avoiding collision with other traffic participants. However, the gradient-based MPC's tendency to get trapped in local optima due to variations in weighting factors demonstrates the challenge of optimization in the urban driving context. The solution may lie in a dynamically adaptive set of weighting factors, which could be tailored based on the vehicle's current environment. Additionally, the dependency of

the policy network on expert guidance points out the necessity for a more elaborate training strategy, possibly delving into a multi-step refinement process.

In conclusion, the interplay between imitation learning (IL) and reinforcement learning (RL) seems to offer innovative solutions. Merging the strengths of both could provide a balanced framework for autonomous driving agents. By integrating the gradient-based MPC with the policy network in a multi-step approach, agents could be better equipped to navigate specific challenges while adhering to broader driving objectives.

BIBLIOGRAPHY

[Amos et al., 2019] Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B., and Kolter, J. Z. (2019). Differentiable mpc for end-to-end planning and control.

[Babaeizadeh et al., 2018] Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. (2018). Stochastic variational video prediction.

[Bemporad et al., 2002] Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20.

[Bertozzi et al., 2000] Bertozzi, M., Broggi, A., and Fascioli, A. (2000). Vision-based intelligent vehicles: State of the art and perspectives. *Robotics and Autonomous Systems*, 32(1):1–16.

[Bharadhwaj et al., 2020] Bharadhwaj, H., Xie, K., and Shkurti, F. (2020). Model-predictive control via cross-entropy and gradient-based optimization.

[Bojarski et al., 2016] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to end learning for self-driving cars.

[Bronstein et al., 2022] Bronstein, E., Palatucci, M., Notz, D., White, B., Kuefler, A., Lu, Y., Paul, S., Nikdel, P., Mougin, P., Chen, H., Fu, J., Abrams, A., Shah, P., Racah, E., Frenkel, B., Whiteson, S., and Anguelov, D. (2022). Hierarchical model-based imitation learning for planning in autonomous driving.

[Camacho and Alba, 2013] Camacho, E. and Alba, C. (2013). *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London.

[Chen et al., 2021] Chen, D., Koltun, V., and Krähenbühl, P. (2021). Learning to drive from a world on rails.

[Chen and Krähenbühl, 2022] Chen, D. and Krähenbühl, P. (2022). Learning from all vehicles.

[Chen et al., 2019] Chen, D., Zhou, B., Koltun, V., and Krähenbühl, P. (2019). Learning by cheating.

[Chen et al., 2023] Chen, L., Wu, P., Chitta, K., Jaeger, B., Geiger, A., and Li, H. (2023). End-to-end autonomous driving: Challenges and frontiers.

[Chitta et al., 2021] Chitta, K., Prakash, A., and Geiger, A. (2021). Neat: Neural attention fields for end-to-end autonomous driving.

[Chitta et al., 2022] Chitta, K., Prakash, A., Jaeger, B., Yu, Z., Renz, K., and Geiger, A. (2022). Transfuser: Imitation with transformer-based sensor fusion for autonomous driving.

[Codevilla et al., 2018] Codevilla, F., Müller, M., López, A., Koltun, V., and Dosovitskiy, A. (2018). End-to-end driving via conditional imitation learning.

[Codevilla et al., 2019] Codevilla, F., Santana, E., López, A. M., and Gaidon, A. (2019). Exploring the limitations of behavior cloning for autonomous driving.

[Denton and Fergus, 2018] Denton, E. and Fergus, R. (2018). Stochastic video generation with a learned prior.

[Dosovitskiy et al., 2017] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator.

[Epic Games,] Epic Games. Unreal engine.

[Fagnant and Kockelman, 2015] Fagnant, D. and Kockelman, K. (2015). Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77.

[Franceschi et al., 2020] Franceschi, J.-Y., Delasalles, E., Chen, M., Lamprier, S., and Gallinari, P. (2020). Stochastic latent residual video prediction.

[García et al., 1989] García, C. E., Prett, D. M., and Morari, M. (1989). Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348.

[Ha and Schmidhuber, 2018] Ha, D. and Schmidhuber, J. (2018). World Models. arXiv:1803.10122 [cs, stat].

[Hafner et al., 2020] Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.

[Hafner et al., 2019] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels.

[Hafner et al., 2021] Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. (2021). Mastering atari with discrete world models. In *International Conference on Learning Representations*.

[Hansen et al., 2022] Hansen, N., Wang, X., and Su, H. (2022). Temporal difference learning for model predictive control.

[Henaff et al., 2019] Henaff, M., Canziani, A., and LeCun, Y. (2019). Model-predictive policy learning with uncertainty regularization for driving in dense traffic.

[Henaff et al., 2018] Henaff, M., Whitney, W. F., and LeCun, Y. (2018). Model-based planning with discrete and continuous actions.

[Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning.

[Hu et al., 2022] Hu, A., Corrado, G., Griffiths, N., Murez, Z., Gurau, C., Yeo, H., Kendall, A., Cipolla, R., and Shotton, J. (2022). Model-based imitation learning for urban driving.

[Hussein et al., 2017] Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2).

[Inc., 2022] Inc., T. M. (2022). Matlab version: 9.13.0 (r2022b).

[Johansson et al., 2004] Johansson, R., Williams, D., Berglund, A., and Nugues, P. (2004). Carsim: A system to visualize written road accident reports as animated 3d scenes. In *Proceedings of the 2nd Workshop on Text Meaning and Interpretation*, TextMean '04, page 57–64, USA. Association for Computational Linguistics.

[Kaur et al., 2021] Kaur, P., Taghavi, S., Tian, Z., and Shi, W. (2021). A survey on simulators for testing self-driving cars.

[Kendall et al., 2018] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2018). Learning to drive in a day.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Kingma and Welling, 2022] Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes.

[Lang et al., 2019] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds.

[Lecun et al., 2004] Lecun, Y., Cosatto, E., Ben, J., Muller, U., and Flepp, B. (2004). Dave: Autonomous off-road vehicle control using end-to-end learning. *DARPA-IPTO Final Report*, page 36.

[Lee et al., 2020] Lee, G., Kim, D., Oh, W., Lee, K., and Oh, S. (2020). Mix-gail: Autonomous driving using demonstrations with mixed qualities. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5425–5430.

[Li and Todorov, 2004] Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation and Robotics*.

[Li et al., 2017] Li, Y., Song, J., and Ermon, S. (2017). Infogail: Interpretable imitation learning from visual demonstrations.

[Lillicrap et al., 2019] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning.

[Maciejowski, 2002] Maciejowski, J. M. (2002). Predictive control : with constraints.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

[Pomerleau, 1988] Pomerleau, D. A. (1988). ALVINN: An Autonomous Land Vehicle in a Neural Network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann.

[Prakash et al., 2021] Prakash, A., Chitta, K., and Geiger, A. (2021). Multi-modal fusion transformer for end-to-end autonomous driving.

[Rajamani, 2012] Rajamani, R. (2012). *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer US, Boston, MA.

[Romero et al., 2023] Romero, A., Song, Y., and Scaramuzza, D. (2023). Actor-critic model predictive control.

[Rong et al., 2020] Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., et al. (2020). Lgsvl simulator: A high fidelity simulator for autonomous driving. *arXiv preprint arXiv:2005.03778*.

[Ross et al., 2011] Ross, S., Gordon, G. J., and Bagnell, J. A. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. Number: arXiv:1011.0686 arXiv:1011.0686 [cs, stat].

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge.

[Russell, 1998] Russell, S. (1998). Learning agents for uncertain environments. page 101 – 103. Cited by: 223.

[Sauer et al., 2018] Sauer, A., Savinov, N., and Geiger, A. (2018). Conditional affordance learning for driving in urban environments.

[Schmidhuber, 2015] Schmidhuber, J. (2015). On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[Siemens PLM Software, 2023] Siemens PLM Software (2023). Safety testing. `https://plm.sw.siemens.com/en-US/simcenter/autonomous-vehicle-solutions/prescan/`. Accessed: 2023-08-29.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Sobal et al., 2022] Sobal, V., Canziani, A., Carion, N., Cho, K., and LeCun, Y. (2022). Separating the world and ego models for self-driving.

[Sohn et al., 2015] Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

[Srinivas et al., 2018] Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. (2018). Universal planning networks.

[Sutton, 1991] Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163.

[Sutton et al., 2018] Sutton, R. S., Bach, F., and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press Ltd.

[Toromanoff et al., 2019] Toromanoff, M., Wirbel, É., and Moutarde, F. (2019). End-to-end model-free reinforcement learning for urban driving using implicit af-

fordances. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7151–7160.

[Toromanoff et al., 2020] Toromanoff, M., Wirbel, E., and Moutarde, F. (2020). End-to-end model-free reinforcement learning for urban driving using implicit affordances.

[U.S. Department of Transportation Federal Highway Administration, 2016] U.S. Department of Transportation Federal Highway Administration (2016). Next generation simulation (ngsim) program i-80 videos. [Dataset]. Provided by ITS DataHub through Data.transportation.gov. Accessed 2023-08-24.

[Vaswani et al., 2023] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

[Wu et al., 2022] Wu, P., Jia, X., Chen, L., Yan, J., Li, H., and Qiao, Y. (2022). Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline.

[Zeng et al., 2021] Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., and Urtasun, R. (2021). End-to-end interpretable neural motion planner.

[Zhang et al., 2023] Zhang, J., Huang, Z., and Ohn-Bar, E. (2023). Coaching a teachable student.

[Zhang and Ohn-Bar, 2021] Zhang, J. and Ohn-Bar, E. (2021). Learning by Watching. Number: arXiv:2106.05966 arXiv:2106.05966 [cs].

[Zhang et al., 2021a] Zhang, Z., Liniger, A., Dai, D., Yu, F., and Gool, L. V. (2021a). End-to-end urban driving by imitating a reinforcement learning coach.

[Zhang et al., 2021b] Zhang, Z., Liniger, A., Dai, D., Yu, F., and Van Gool, L. (2021b). End-to-End Urban Driving by Imitating a Reinforcement Learning Coach. Number: arXiv:2108.08265 arXiv:2108.08265 [cs].

Appendix A

ADDITIONAL RESULTS

A.1 World Forward Model

Table A.1: Simple Case

Context Horizon (N)	Prediction Horizon (T)	Semantic Class	IoU	Precision	Recall	F1 Score
20	30	Roads	0.983	0.992	0.991	0.991
		Lane markings	0.446	0.777	0.511	0.616
		Vehicles	0.737	0.878	0.821	0.849
		Green light	0.000	0.000	0.000	0.000
		Red or yellow light	0.747	0.889	0.824	0.855
		Offroad	0.992	0.996	0.996	0.996
20	10	Roads	0.982	0.991	0.991	0.991
		Lane markings	0.401	0.641	0.518	0.573
		Vehicles	0.707	0.838	0.820	0.829
		Green light	0.330	0.845	0.351	0.496
		Red or yellow light	0.685	0.896	0.745	0.813
		Offroad	0.992	0.996	0.996	0.996
10	10	Roads	0.983	0.992	0.991	0.991
		Lane markings	0.384	0.670	0.474	0.555
		Vehicles	0.624	0.720	0.824	0.768
		Green light	0.493	0.833	0.547	0.661
		Red or yellow light	0.673	0.877	0.743	0.805
		Offroad	0.992	0.996	0.996	0.996
5	10	Roads	0.979	0.990	0.990	0.990
		Lane markings	0.345	0.624	0.418	0.500
		Vehicles	0.539	0.678	0.819	0.742
		Green light	0.529	0.856	0.429	0.571
		Red or yellow light	0.627	0.848	0.702	0.768
		Offroad	0.992	0.996	0.996	0.996

Table A.2: Hard Case

Context Horizon (N)	Prediction Horizon (T)	Semantic Class	IoU	Precision	Recall	F1 Score
20	10	Roads	0.673	0.977	0.967	0.972
		Lanes agent can drive	0.940	0.966	0.973	0.969
		Lanes agent cannot drive	0.953	0.972	0.980	0.976
		Lanes affected by red light	0.888	0.937	0.945	0.941
		Lanes affected by green light	0.729	0.850	0.837	0.844
		Lane markings	0.533	0.587	0.851	0.695
		Vehicles	0.435	0.491	0.792	0.606
		Offroad	0.974	0.984	0.990	0.987
10	10	Roads	0.940	0.978	0.960	0.969
		Lanes agent can drive	0.938	0.969	0.967	0.968
		Lanes agent cannot drive	0.951	0.975	0.975	0.975
		Lanes affected by red light	0.881	0.943	0.931	0.937
		Lanes affected by green light	0.704	0.833	0.820	0.826
		Lane markings	0.530	0.597	0.825	0.693
		Vehicles	0.440	0.511	0.761	0.611
		Offroad	0.971	0.980	0.991	0.986
20	30	Roads	0.940	0.973	0.965	0.969
		Lanes agent can drive	0.926	0.968	0.955	0.961
		Lanes agent cannot drive	0.940	0.974	0.965	0.969
		Lanes affected by red light	0.885	0.941	0.937	0.939
		Lanes affected by green light	0.729	0.846	0.840	0.843
		Lane markings	0.424	0.721	0.507	0.595
		Vehicles	0.369	0.747	0.421	0.539
		Offroad	0.971	0.982	0.988	0.985
5	10	Roads	0.941	0.977	0.962	0.969
		Lanes agent can drive	0.940	0.971	0.967	0.969
		Lanes agent cannot drive	0.952	0.977	0.974	0.975
		Lanes affected by red light	0.877	0.930	0.939	0.935
		Lanes affected by green light	0.686	0.848	0.782	0.814
		Lane markings	0.403	0.693	0.491	0.575
		Vehicles	0.353	0.759	0.397	0.521
		Offroad	0.971	0.981	0.990	0.985

A.2 Policy Network

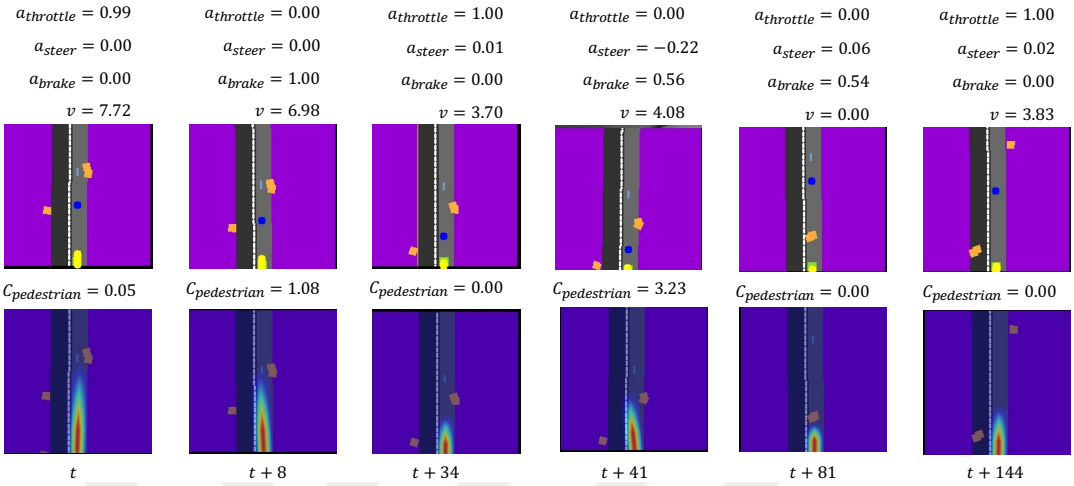

Figure A.1: **A timelapse from a scene where the ego vehicle slows down to avoid a collision with a pedestrian.** We represent the ego vehicle with a green rectangle at the bottom of the image, which is partially occluded by yellow dots that represent the predicted future locations of the ego vehicle. On the bottom row, we show the generated heatmaps used in the differentiable cost calculation. The indices at the bottom shows the amount of time step increment since the first frame, where the time difference between time steps is $\Delta t = 0.05$ *sec*. In the first column, the agent applies a throttle command since the pedestrian cost $C_{pedestrian}$ is small, indicating that the road is safe to move forward without any intervention. Then, in the second column, the agent applies a brake command to slow down since the pedestrian cost is relatively high. The action taken in the second column makes the pedestrian cost zero. Therefore, in the third column, the agent applies a throttle command once more. This fluctuating behavior ends in the fourth column where the agent decides to apply a brake command. In the fourth and fifth columns, a pedestrian is visible walking across the street, so the increase in the pedestrian cost prevents the agent from moving forward. Finally, in the sixth column, the agent decides to apply a throttle command to move forward since the pedestrian cost is zero, and the agent can proceed.