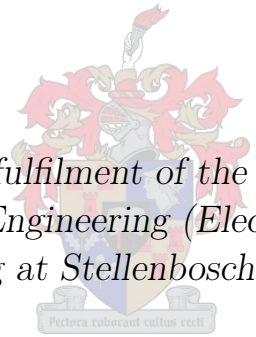
Partial end-to-end reinforcement learning for robustness towards model-mismatch in autonomous racing

by

Andrew Murdoch



Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. J.C. Schoeman
Co-supervisor: Dr. H.W. Jordaan

July 2023

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2023/07/01 .....

Copyright © 2023 Stellenbosch University
All rights reserved.

Abstract

The increasing popularity of self-driving cars has given rise to the emerging field of autonomous racing. In this domain, algorithms are tasked with processing sensor data to generate control commands (e.g., steering and throttle) that move a vehicle around a track safely and in the shortest possible time.

This study addresses the significant issue of practical *model-mismatch* in learning-based solutions, particularly in reinforcement learning (RL), for autonomous racing. Model-mismatch occurs when the vehicle dynamics model used for simulation does not accurately represent the real dynamics of the vehicle, leading to a decrease in algorithm performance. This is a common issue encountered when considering real-world deployments.

To address this challenge, we propose a partial end-to-end algorithm which decouples the planning and control tasks. Within this framework, a reinforcement learning (RL) agent generates a trajectory comprising a path and velocity, which is subsequently tracked using a pure pursuit steering controller and a proportional velocity controller, respectively. In contrast, many learning-based algorithms utilise an end-to-end approach, whereby a deep neural network directly maps from sensor data to control commands.

We extensively evaluate the partial end-to-end algorithm in a custom F1tenth simulation, under conditions where model-mismatches in vehicle mass, cornering stiffness coefficient, and road surface friction coefficient are present. In each of these scenarios, the performance of the partial end-to-end agents remained similar under both nominal and model-mismatch conditions, demonstrating an ability to reliably navigate complex tracks without crashing. Thus, by leveraging the robustness of a classical controller, our partial end-to-end driving algorithm exhibits better robustness towards model-mismatches than an end-to-end baseline algorithm.

Uittreksel

Die toenemende gewildheid van selfbesturende motors het aanleiding gegee tot die opkomende veld van outonome wedrenne. In hierdie domein, het algoritmes die taak om sensordata te verwerk om beheeropdragte (bv., stuur en versneller) te genereer wat 'n voertuig veilig en in die kortste moontlike tyd om 'n baan beweeg.

Hierdie studie spreek die beduidende kwessie van praktiese *model-wanverhouding* in leergebaseerde oplossings aan, veral in versterkingsleer (RL), vir outonome wedrenne. Model-wanpassing vind plaas wanneer die voertuigdinamika-model wat vir simulasie gebruik word nie die werklike dinamika van die voertuig akkuraat voorstel nie, wat lei tot 'n afname in algoritme-werkverrigting. Dit is 'n algemene probleem wat teegekom word wanneer werklike implementerings oorweeg word.

Om hierdie uitdaging aan te spreek, stel ons 'n gedeeltelike- 'end-to-end'-algoritme voor wat die beplanning- en beheertake ontkoppel. Binne hierdie raamwerk genereer 'n versterkingsleer (RL) agent 'n trajek wat 'n pad en snelheid bevat, wat vervolgens nagespoor word deur gebruik te maak van 'n suiwer agtervolgstuurbeheerder en 'n proporsionele snelheidsbeheerder, onderskeidelik. Daarteenoor gebruik baie leergebaseerde algoritmes 'n 'end-to-end'-benadering, waardeur 'n diep neurale netwerk direk (DNN) vanaf sensordata karteer om opdragte te beheer.

Ons evalueer die gedeeltelike- 'end-to-end'-algoritme breedvoerig in 'n pasgemaakte 'F1tenth'-simulasie, onder toestande waar model-wanverhoudings in voertuigmassa, draai styfheidskoeffisient en padoppervlakwrywingskoeffisient teenwoordig is. In elk van hierdie scenario's het die werkverrigting van die gedeeltelike- 'end-to-end'-agente dieselfde gebly onder beide nominale en model-wanpastoestande, wat 'n vermoe demonstreer om komplekse spore betroubaar te navigeer sonder om te verongeluk. Deur dus die robuustheid van 'n klassieke kontroleerder te benut, toon ons gedeeltelike- 'end-to-end'- bestuursalgoritme beter robuustheid teenoor model-wanpassings as 'n 'end-to-end'- basislynalgoritme.

Acknowledgements

This thesis appears in its current form due to the assistance and guidance of several people. I would therefore like to offer my sincere thanks to all of them.

I am thankful to God for granting me this opportunity to study. I praise Him for His strength, sustenance, and unwavering faithfulness.

I would like to express my sincere gratitude to my parents, Ross and Jeanne Murdoch. You have been a source of inspiration, and have fostered continual spiritual and emotional growth, as well as provided financial support throughout my studies.

To my supervisors, Dr. J.C. Schoeman and Dr. H.W. Jordaan, I would like to thank you for the guidance that you have provided, as well as the patience and kindness you have shown towards me during my degree. Thank you for the many meetings, comments, corrections, and encouragement.

Friends, thank you for your continual support, prayer, and encouragement throughout my studies.

And to my colleagues at the Electronic Systems Laboratory, thank you for making my studies a pleasant experience.

Contents

List of Figures

List of Algorithms

List of Tables

Nomenclature

Acronyms and abbreviations

LiDAR	light detection and ranging
IMU	inertial measurement unit
DNN	deep neural network
RL	reinforcement learning
MPC	model predictive control
DARPA	Defense Advanced Research Projects Agency
IL	imitation learning
CNN	convolutional neural network
BNN	bayesian neural networks
GTS	Gran Turismo Sport
TORCS	The Open Source Car Simulator
CAPS	conditioning for action policy smoothness
F1	Formula 1
ANN	artificial neural network
ReLU	rectified linear unit
FNN	feedforward neural network
Adam	adaptive moment estimation
MPD	Markov decision process
TD3	twin delay deep deterministic policy gradient
RC	remote controlled
DC	direct current
CoG	centre of gravity

Notation

x	Scalar
$\boldsymbol{x}$	Vector
$\boldsymbol{x}^\intercal$	Transpose of vector $\boldsymbol{x}$

Chapter 1

Introduction

Autonomous cars have the potential to revolutionise transportation by providing mobility to a broad range of people. These vehicles could (a) increase the independence of those who are incapable of driving, (b) reduce the number of road accidents caused by driver negligence, and (c) reduce both road congestion and pollution by optimising routes and driving style. These are just a few ways in which autonomous cars are expected to impact our daily lives [1].

There are numerous challenges to the large scale deployment of road-going autonomous cars. In particular, public roads are an unpredictable environment, and autonomous cars face a wide variety of scenarios which are difficult to program for. There are many edge cases which will require the vehicle to not only respond quickly, but also operate at its handling limits to ensure the safety of its occupants. An example of such a scenario is avoiding a collision [2].

The emergence of autonomous racing as a research field stems from the need to design autonomous driving solutions that address these edge cases. Racing leagues such as Formula Student Driverless [3], Indy Autonomous Challenge [4] and F1tenth [5] provide competitive environments for teams to develop autonomous algorithms that operate vehicles at the edge of their handling limits. In particular, F1tenth scaled racing cars, shown in Figure 1.1, are an ideal research platform due to their standardised hardware requirements and well developed simulators.

Figure 1.1: The F1tenth standard vehicle, built on the chassis of a miniature RC car [6].

1.1 Research motivation

Racing vehicles such as F1tenth cars are equipped with light detection and ranging (LiDAR) scanners, inertial measurement units (IMU) and rotational encoders. Autonomous racing algorithms must convert data from these sensors into steering and throttle actuator commands that safely move a vehicle around a track in the shortest possible time. Therefore, these algorithms most optimise for two objectives; (a) performance, i.e., operating the vehicle at the handling limits to achieve the fastest lap time, and (b) safety, which is to ensure that the vehicle does not collide with the track boundary or obstacles such as other cars. These two objectives are inherently in conflict with each other because operating the vehicle close to its handling limit increases the risk of losing control [7].

A recent breakthrough in autonomous racing was the introduction of learning based solutions, such as reinforcement learning (RL) for vehicle control. Reinforcement learning is a machine learning paradigm where an agent learns to make sequential decisions by interacting with an environment to maximize cumulative rewards. By setting up a reward signal that is maximised through fast and safe lap completion, RL agents can learn to race effectively. Interestingly, RL algorithms are commonly implemented within an end-to-end architecture, whereby a deep neural network (DNN) is trained to map sensor data directly to actuator commands [7]. These end-to-end RL approaches have achieved excellent results in scenarios that are considered challenging for classical approaches that separate the planning and control tasks. These scenarios include racing with low computational budgets [8; 9] and racing against multiple vehicles [10; 11].

Most RL agents undergo training in simulation before their deployment on physical vehicles [5; 12]. To ensure a consistent environment between training and deployment, simulators attempt to replicate the real world as closely as possible. This involves employing system identification techniques to create precise vehicle dynamics models [12]. However, estimating the parameters of the vehicle model is challenging due to the dynamic nature of the driving task. In fact, it is inevitable that these parameters undergo changes over time [13]. As a result, it is likely that the vehicle dynamics model employed during training does not align with the real-world vehicle dynamics. This phenomenon, known as model mismatch, leads to a decline in performance [14]. Since some level of model mismatch is always present, accurate system identification alone is insufficient to ensure the vehicle's safety. It is imperative that autonomous vehicles exhibit robustness towards modeling errors.

Research efforts into addressing this challenge in RL approaches have largely been limited to modifying the DNN training process. For example, *sim-to-real* best practices include randomising vehicle model parameters during training [15], or retraining the DNN after deployment [12]. Despite these efforts, the performance of learning-based methods are still negatively affected when there is model mismatch present [16].

1.2 Aims and objectives

The aim of this project is to develop a reinforcement learning autonomous racing algorithm that is robust to the vehicle modelling errors associated with real-world deployment. As such, our racing algorithm must minimise lap time and drive safely under conditions where model mismatch is present. The racing scenario that we consider is a single-vehicle time trial, whereby the vehicle must complete laps while being alone on the track. Furthermore,

our proposed solution should be compared to current RL solutions for autonomous racing. Therefore, our objectives are stated as:

1. Investigate the literature regarding methods for developing autonomous racing algorithms, with focus on methods that are robust to vehicle modelling error.
2. Identify and implement an appropriate baseline reinforcement learning autonomous racing algorithm.
3. Design a reinforcement learning autonomous racing algorithm that is robust to vehicle modelling errors.
4. Simulate the baseline and proposed racing algorithms under practical model mismatch conditions.

1.3 Document outline

Chapter 2 constitutes an overview of the existing approaches to solving the autonomous racing problem in literature. A variety of classical and learning-based solutions are discussed, with a focus on how these methods handle uncertainty in the vehicle model. We identify a suitable research avenue in methods that seek to unify ideas from classical and learning based approaches by utilising a DNN within a classical (i.e., decoupled) structure. Methods that utilise this approach are known as *partial end-to-end*.

The literature study is followed with the an overview of the necessary theory to understand reinforcement learning in Chapter 3. An essential component in training a reinforcement learning algorithm to complete a robotic task is a realsitic simulator. As such, Chapter 4 describes how the racing environment was modelled in a suitable custom F1tenth simulator.

Chapter 5 the baseline end-to-end RL solution, as well as the techniques for applying an RL algorithm to solve the racing problem. This end-to-end algorithm is extensiely evaluated in racing conditions where no model mismatch is present. Furthermore, the effectiveness of *sim-to-real* techniques for end-to-end methods in the context of autonomous racing is investigated.

Chapter 6 describes our partial end-to-end solution to the autonomous racing problem. This chapter compares the performance of our chosen partial end-to-end architecture against the end-to-end baseline algorithm, as well as the performance of several alternative partial end-to-end architectures against each other.

The experiments that were performed under model mismatch conditions are found in Chapter 7. This chapter considers practical model mismatch settings that could be encountered during real-world transfer, such as adding a dynamic mass to the vehicle, changing tire parameters, as well as a change in the road surface friction coefficient. The thesis is then concluded in Chapter 8 with a summary of the work completed, as well as recommendations for future work.

Chapter 2

Literature study

With the project objectives in mind, we conduct a study of the literature regarding autonomous racing. Figure 2.1 presents a taxonomy of approaches to the solving the autonomous racing problem. While the majority of efforts fall into the classical or end-to-end categories, a few research efforts have been made into combining classical and end-to-end ideas through a partial end-to-end driving architecture. Our focus is on examining these various approaches to developing autonomous driving algorithms address the model mismatch and sim-to-real problems. We end the chapter with a discussion of the research gaps, as well as a summary of the expected contributions of this project towards the literature.

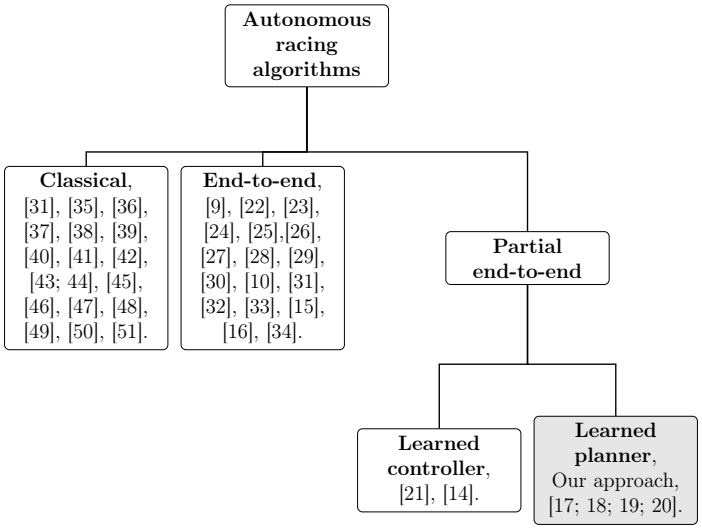


Figure 2.1: A taxonomy of approaches to solving the autonomous racing problem. Research efforts associated with each approach are listed.

2.1 Classical approaches

Classical approaches split the task of generating actuator commands from sensor data into three distinct phases, namely (a) perception, (b) planning and (c) control [7]. Perception is the task of processing sensor data into a format that can be used for planning. Generally, sensor fusion techniques are used to localise the vehicle within a map [52; 53]. Planners use the map and localisation data to compute a trajectory that is subject to the vehicle

dynamics constraints [44]. The controller then executes the plan on the physical hardware [35]. This generic framework is illustrated in Figure 2.2.

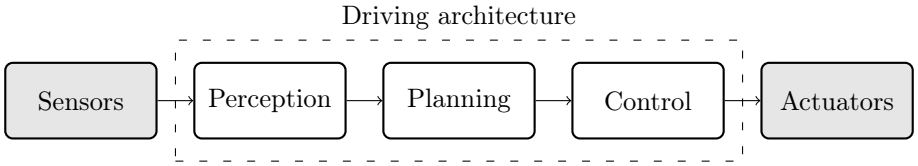


Figure 2.2: The classical autonomous driving software architecture. The driving software, which is enclosed in the dotted line, maps between sensor input and the commands sent to the actuators by performing perception, planning and control separately.

Classical approaches have successfully been used to control full scale racing cars [36; 37; 38] at high performance levels and near their handling limits. For instance, Heilmier et al. [39] and Betz et al. [40] achieved speeds of 150 and 270 km/h respectively. The success of classical approaches have also been showcased on scaled racing cars by Liniger et al. [43]. We now discuss methods for planning and control in more detail. Perception algorithms are not discussed because they are outside of the scope for this project.

2.1.1 Planning

Within classical planning approaches, trajectories are generated via optimisation methods that minimise a cost function. These trajectories trypically consist of a series of x and y coordinates and velocities that must be tracked using controllers.

Several approaches [39; 41; 42] shift the computational burden of generating a trajectory to offline, before the race starts. These approaches are categorised by their cost function. While Heilmeier et al. [39] optimise the geometry of the path to achieve minimum curvature, Kelly et al. [41] considers a time-based optimisation to construct a minimum-time path. These offline planners create robustness towards the modelling errors by constructing conservative paths and taking vehicle model constraints into account. For instance, Heilmeier et al. [39] constrain the curvature of the path so that the vehicle adheres to a maximum lateral acceleration constraint.

While offline planning is useful, a need exists to construct trajectories that actively avoid collisions with the environment or other vehicles in an online manner. Although it is an optimal control technique, model predictive control (MPC) is commonly used for generating fixed-horizon trajectories online [43; 44; 54; 55]. MPCs sample dynamically feasible trajectories by forward simulating the vehicle dynamics using multiple actuator input sequences. A cost is assigned to each trajectory, after which the trajectory with the minimum cost is executed [7]. While standard MPCs consider linear vehicle dynamics, Liniger et al. [43] proposed a non-linear MPC to more accurately represent the vehicle dynamics.

Another common approach to constructing a trajectory online is graph based planning, whereby a hierarchical tree of parameterised paths are spanned across the drivable area of the track [45; 46; 47]. These approaches calculate the cost of each path based on its geometry, and select the path associated with the least cost. Since the effect of model-mismatch can be exagurated when extreme control actions are taken, the graph based approach by Stahl et al. [45] ensured continuous and smooth paths by parameterising them as cubic polynomials in a curvilinear coordinate frame attached to the track

centerline, known as the Frenet frame. Their method successfully controlled the Devbot 2.0 Roborace development vehicle shown in Figure 2.3 at speeds of up to 212 km/h.

Figure 2.3: The DevBot 2.0 vehicle used by Stahl et al. [45] in the Roborace.

2.1.2 Control

Controllers allow a trajectory to be executed on hardware by computing control commands that minimise the error between the vehicle's current and desired state. The use of feedback controllers to follow the trajectory directly addresses the need for robustness towards vehicle model error. At this level of abstraction, typical control commands are steering angle and longitudinal acceleration, which are sent to low-level controllers to actuate the motors and brakes [7].

Classical controllers separate steering and longitudinal acceleration control. While longitudinal control is typically performed using a PID controller, methods to perform steering control to track the path vary. Pure pursuit is a simple steering controller presented by Coulter [48] that steers the vehicle towards a target point on the path that is always some distance ahead of the it. The steering angles are based on the geometric properties of the vehicle. Although the method is effective at low speeds, it does not take into account a dynamic model of the vehicle, which leads to sub-optimal performance at higher speeds.

The approaches by Becker et al. [49] and Hoffman and Montemerlo [50] also steer the vehicle towards a target point on the path ahead of the vehicle, but compute steering angles based on a dynamic model of the vehicle. Becker et al. [49] reported a four-fold improvement in path tracking error over the pure pursuit algorithm on an F1tenth race car, while the controller by Hoffman and Montemerlo [50] was implemented on the winning vehicle of the DARPA Grand Challenge in 2005. The vehicle used by Hoffmann et al. [50] is shown in Figure 2.4.

Model predictive control (MPC) is a state of the art classical control technique [9; 51; 56; 57; 58; 59]. When used as a path tracking controller, the objective of the MPC is to minimise the error between the vehicle's actual and planned trajectory [58]. However, many MPC approaches use linearised vehicle models [51] because the computation requirements for an MPC that utilises a non-linear vehicle model is too demanding for hardware onboard a scaled vehicle to execute [9]. Furthermore, the MPC cost function is considered too inflexible for complex manoeuvres such as racing with multiple vehicles [30]. Research efforts into improving the performance of MPCs under model-mismatch

conditions are focused on learning a vehicle model online, such that the error between the vehicle model and real vehicle dynamics is minimised [9; 59].

Figure 2.4: The controller by Hoffmann et al. [50] was implemented on a modified Volkswagen Touareg called Stanley. Hoffmann et al.'s vehicle won the DARPA grand challenge in 2005.

2.2 End-to-end approaches

The limitations of optimisation techniques from classical methods has led to research in learning-based systems that use data to formulate a decision-making policy to control the vehicle [30]. Many learning-based approaches use an end-to-end architecture, whereby a decision making agent, whose policy is typically represented by a deep neural network (DNN), predicts actuator commands directly from sensor data, thus performing the task of the entire classical architecture. This end-to-end approach is illustrated in Figure 2.5. The two machine learning paradigms by which the neural networks in end-to-end systems are trained are imitation learning (IL) and reinforcement learning (RL).

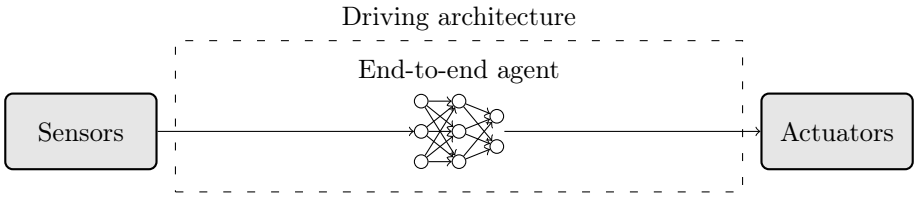


Figure 2.5: The end-to-end autonomous driving architecture, whereby an agent maps directly from sensor data to actuator commands.

2.2.1 Imitation learning

Imitation learning is a supervised learning technique that aims to learn a mapping from sensor data to control actions, by using examples generated from an expert as training data [9; 24; 25; 60]. Imitation learning approaches that solve the autonomous racing problem typically rely on an MPC to provide expert training data.

Tatulea-Codrean et al. [9] used an IL algorithm to train a DNN that mimics a non-linear MPC expert. Sampling an action from the neural network was less computationally expensive than sampling from the non-linear MPC. As such, the use of a DNN made

physical deployment tractable given the computational constraints of the F1tenth vehicle that was used as the hardware platform.

The IL approach by Pan et al. [24] demonstrated a convolutional neural network (CNN) that is capable of learning a mapping between images taken by a camera mounted to the front of the vehicle, and control actions. Their approach mimics an MPC expert that has access to a more expensive set of sensors which included a LiDAR. By using only camera images as input to the CNN, Pan et al. [24] were able to circumvent the need for expensive LiDAR sensors. Furthermore, the IL algorithm performed high speed manoeuvres in a difficult off-road setting where vehicle dynamics modelling inaccuracies are likely to occur due to the difficulty of modelling the track surface.

Lee et al. [25] used IL to to train an ensemble of Bayesian neural networks (BNNs) to create a policy that was robust to sensor failure, which was not possible with previous classical approaches. An MPC expert was used to generate training data. The vehicle and track used by both Lee et al. [25] and Pan et al. [24] is shown in Figure 2.6.

(a) **(b)**

Figure 2.6: The (a) 1/5 scale vehicle and (b) track used by both Lee et al. [25]. and Pan et al. [24].

The use of end-to-end IL methods necessitates the involvement of an expert MPC to generate the necessary training data [61]. Consequently, the application of IL approaches is restricted to scenarios in which MPCs already demonstrate a high degree of proficiency. As such, IL methods do not enhance the sim-to-real capabilities of classical approaches with respect to modeling mismatch. Instead, they increase the feasibility of deploying MPC policies on actual vehicles that are subject to computational limitations and potential sensor malfunctions [25; 61].

2.2.2 Reinforcement learning

Reinforcement learning is a method that aims to train decision-making agents to maximize a scalar reward signal through direct interaction with their environment [62]. In the context of training agents to control racing cars, RL approaches typically use DNNs to represent the decision-making policy [7]. DNNs offer several advantages when used within an RL framework, including the ability to map complex inputs such as camera images to control outputs [16; 26; 28; 29], as well as the ability to sample actions with low computational cost compared to other methods such as MPC [14]. Unlike imitation learning (IL) methods, RL algorithms do not require expert training data and can find optimal strategies with minimal human intervention, making them suitable for challenging scenarios such as multi-vehicle racing [10; 11]. Interestingly, many research efforts into RL applied to racing are focused at solving video games.

Reinforcement learning applied to racing video games

Jaritz et al. [28] and Perot et al. [29] present research efforts that use model-free RL to train an agent to race in the video game World Rally Championship 6. Model-free RL algorithms enable agents to learn to make decisions based solely on trial and error experience, without using a model of the vehicle or environment. Both studies focus on RL agents that learn a mapping between the game screen and control output by representing the policy as a CNN with long-short-term memory (LSTM) layers. Despite the agents' ability to drive for some distance along the track, Jaritz et al. [28] reported that the vehicle collided with the track boundary an average of 5.44 times per kilometer.

The video game Gran Turismo Sport (GTS), shown in Figure 2.7, has been the subject of several research efforts that showcase model-free RL agents can outperform humans [10; 11; 30] in some race settings. Fuchs et al. [30] use a DNN to map a hand-crafted set of features to control outputs in a racing scenario where there is a single vehicle on the track. This agent outperforms even the best competitive GTS players. Fuchs et al. [30] also considers the effect of model-mismatch on the agent by varying the road-surface friction coefficient after training. There is significant uncertainty associated with the road-surface friction value due to its dependence on the weather and the difficulty of measuring it at every point along the track [63]. As such, robustness to road-surface friction is a pertinent sim-to-real issue. The agent by Fuchs et al. [30] makes contact with the track boundary when model mismatched in friction are considered.

Wurman et al. [11] and Song et al. [10] also consider GTS racing scenarios, but with multiple vehicles. Both approaches achieve better than human performance using a similar set of input features and DNN design as Fuchs et al. [30]. This demonstrates the ability of RL agents to learn policies that solve complex tasks such as overtaking. However, all three approaches that solve GTS allow the vehicle to scrape against the boundary of the track or collide with other vehicles, which is an unrealistic assumption to make for RL agents that are deployed onto physical cars.

Figure 2.7: A screenshot of Song et al.'s agent overtaking multiple vehicles in the video game GTS [10].

Cai et al. [64] present a model-free RL approach to controlling a race car during high-slip drifting manoeuvres in the Speed Dreams simulator. The agent controls the vehicle beyond the friction limit of the tires. While their approach demonstrates that RL agents can learn to control vehicles with complex non-linear dynamics, their DNN maps an unrealistic set of input features (e.g., the derivative of the angle between the vehicle's heading and forward velocity vector) to control actions.

These RL techniques have shown excellent performance when applied to racing games. However, they make unrealistic assumptions that render them infeasible for real-world driving scenarios. The most significant challenges to applying these approaches on physical vehicles are the lack of safety considerations and the availability of unrealistic input features sets.

Reinforcement learning applied to realistic racing scenarios

A number of RL related research efforts do consider more realistic driving scenarios, as well as techniques that improve the sim-to-real capabilities of RL agents. For instance, Niu et al. [31] address the issue that DNNs do not have perfect prediction accuracy and can select unsafe actions that lead to crashes, even after deployment. Their approach uses a model-based safety controller that acts as a safeguard mechanism to prevent the agent from selecting unsafe actions on a vehicle simulated with the open racing simulator (TORCS). Their model-free RL agent did not crash during training or testing with the safeguard mechanism in place. Niu et al.'s [31] approach may alleviate the sim-to-real gap by allowing an RL agent to train directly on the physical hardware without risking a crash. However, their approach is not validated on a physical vehicle.

The effectiveness of model-based RL in real-world applications is demonstrated by Brunbauer et al. [34], who deploy an agent which learns to control the physical F1tenth vehicle shown in Figure 2.8. Their agent learns to race using the Dreamer algorithm [65], whereby a learned observation model is used to predict agent-track interactions. The agent can then learn a policy based purely on 'imagined' sequences using the observation model, without interacting with the track.

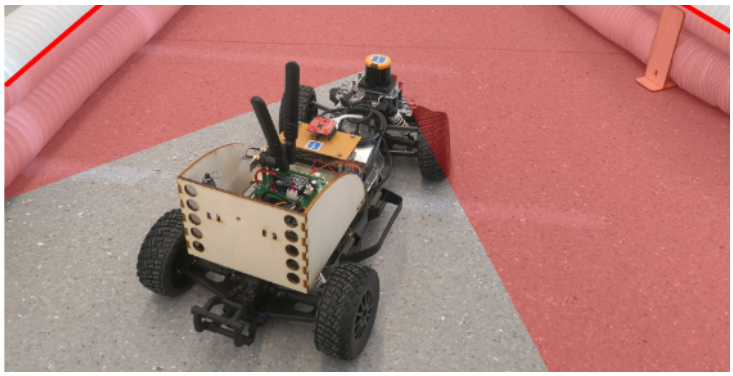


Figure 2.8: The F1tenth vehicle used by Brunbauer et al. [34]. The red area indicates the field of view of the LiDAR scanner on the vehicle.

Hsu et al. [16] deployed a model-free RL agent that takes camera images as input onto a physical F1tenth vehicle. They apply conditioning for action policy smoothness (CAPS) [66] as a policy output regularisation strategy. Policy output regularisation aims to prevent jerky and extreme vehicle control actions by adding the difference between actions selected on consecutive time steps to the cost function of the policy update. Extreme steering and acceleration control actions causes the vehicle to operate closer to the friction limits of the tires than with smooth control actions, and can lead to uncontrollable and dangerous behaviour. This issue becomes more pertinent when considering scenarios in which model mismatches are present [33].

Hsu et al. [16] also employed domain randomisation by adding noise to the input camera images. Domain randomisation involves varying the simulation environment slightly

during training. This prevents the agent from forming a policy that overfits to a single simulation environment. Moreover, it is more likely that the data encountered by the agent during training includes the distribution of data encountered after deployment when domain randomisation is used [12]. The standard end-to-end agent's average lap completion rate, along with agents trained using solely policy output regularisation or solely domain randomisation was 0%. However, average lap completion rate increased to 42% when both domain randomisation and policy output regularisation techniques were used.

Ivanov et al. [15] trained a model-free RL agent to steer the F1tenth vehicle shown in Figure 2.9 around a corner using only a LiDAR sensor as input. They identified their perception model (i.e., the simulated LiDAR sensor) as a major source of uncertainty because the track reflectivity was unknown. Although they applied domain randomisation techniques by randomising LiDAR measurement noise, they achieved an 83% success rate rounding the corner using their best method.

Figure 2.9: Ivanov et al.'s F1tenth vehicle [15].

Chisari et al. [33] also utilized domain randomisation and action smoothing techniques to enable a model-free RL agent to control 1:43 scale cars in their study. Rather than adding noise to the sensor readings, they apply domain randomisation by adding noise to the vehicle dynamics model parameters. Without domain randomisation, the agent was unable to complete even a single lap on the physical car. When they compared their baseline reinforcement learning agent which incorporated only domain randomisation to an MPC, the MPC outperformed their agent in terms of both lap time and track boundary violations. However, their agent that used domain randomisation and output policy regularization demonstrated a 30% reduction in track boundary violations compared to the MPC method, although it had an 8.1% slower lap time.

Table 2.1 summarizes the end-to-end RL approaches reviewed. While these end-to-end RL algorithms have delivered impressive results in video games, practical implementations have been limited to small-scale vehicles, primarily using the F1tenth racing platform. Furthermore, results have shown that end-to-end agents often fail under model-mismatch conditions, even when sim-to-real practices such as domain randomisation are implemented.

2.3 Partial end-to-end approaches

Approaches to designing autonomous driving algorithms that synthesise the classic and end-to-end techniques are now considered. In these approaches, the modular structure of classic approaches are utilised. However, rather than implementing techniques associated with classic approaches in each of the driving algorithm components, these components may be combined with or replaced by a DNN. Two popular partial end-to-end design

Author(s)	Year	Physical vehicle	Sim2real research contribution
Jaritz et al. [28]	2018		-
Perot et al. [29]	2017		-
Fuchs et al. [30]	2021		Test effects of model inaccuracy in simulation.
Song et al. [30]	2021		-
Wurman et al. [11]	2021		-
Cai et al. [64]	2021		-
Niu et al. [31]	2020		Safety module based on learned vehicle model prevents agent from selecting unsafe actions.
Brunnbauer et al. [34]	2021	F1tenth	-
Hsu et al. [16]	2022	F1tenth	Domain randomisation while training. Control action smoothing.
Ivanov et al. [15]	2020	F1tenth	Domain randomisation while training.
Chisari et al. [33]	2021	1 : 43 scale cars	Domain randomisation while training, policy refinement after deployment. Control action smoothing.

Table 2.1: A summary of end-to-end reinforcement learning approaches for autonomous racing.

philosophies are to use a DNN to perform the task of the planner [17; 18; 19; 23], or the task of the controller [14; 21]. The resulting driving software architectures are shown in Figure 2.10.

2.3.1 Learned controller

In the partial end-to-end configuration with a learned controller, the autonomous driving system leverages classical perception and planning algorithms. In this setup, a DNN serves as the controller by learning mapping between the vehicle's current state and desired state to issue actuator commands [7].

Evans et al. [21], used an RL agent to modify the control output of a pure pursuit path follower that tracks a global path, with the goal of avoiding previously unseen obstacles on the track. Their experiments were performed on a simulated F1tenth setup. The proposed partial end-to-end agent is able to avoid 94% of unseen obstacles without maintaining an obstacle map, while maintaining a smoother path than the baseline end-to-end agent.

In a study by Ghignone et al. [14], a simulated F1tenth vehicle's path tracking controller was replaced with a model-free RL agent. During training, domain randomisation was applied by varying the tire friction coefficient. The results showed that the approach

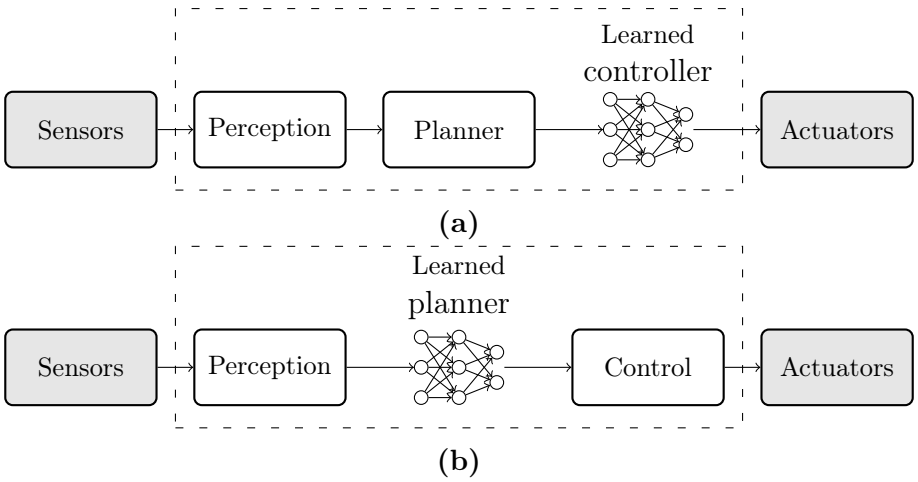


Figure 2.10: The two configurations of the partial end-to-end pipeline are (a) a classic planner in conjunction with a learned controller, and (b) a learned planner in conjunction with a classic controller. Both configurations require a perception module to localise the vehicle.

led to 6.7-fold and 2.7-fold improvements in the number of track collisions under model-mismatch conditions compared to an end-to-end agent and a classical MPC approach, respectively. Figure 2.11 showcases the performance difference between the end-to-end and partial end-to-end agents by displaying the paths taken by both agents rounding a corner multiple times. The partial end-to-end agent visibly swerves less and collides with the track boundary fewer times than the end-to-end agents. These findings demonstrate the benefits of decoupling the planning and control tasks for learning-based systems. Specifically, decoupling planning and control in learning-based systems can enhance their robustness towards model mismatch.

Figure 2.11: Distributions of paths taken by (a) an end-to-end and (b) a partial end-to-end agent to round a corner multiple times during an experiment by Ghignone et al. [14].

2.3.2 Learned trajectory planner

In a partial end-to-end configuration with a learned planner, the autonomous driving system uses classical perception and control algorithms in conjunction with a DNN planner. The DNN planner is trained to map perception data (i.e., vehicle state and track information) to trajectories. These trajectories are then tracked by classical controllers, enabling the vehicle to race autonomously [7].

Weiss and Behl [18; 20] conducted an extensive examination of partial end-to-end systems in which a deep neural network (DNN) replaces the planner. Specifically, they

utilized an IL algorithm to train a DNN to generate trajectories that were tracked using a pure pursuit controller. They applied this approach to a realistic Formula One (F1) video game, which allowed them to use human experts playing the video game as training data. Through experimentation, they discovered that generating trajectories using Bézier curves resulted in high performance. Their proposed partial end-to-end method significantly outperformed the end-to-end baseline agent. In particular, the end-to-end agent was unable to finish a lap without crashing, while their approach successfully completed laps.

Capo et al. [17] employed a model-free RL agent to learn the task of planning in the TORCS simulator. Specifically, the agent was trained to produce a single point in front of the vehicle, given a hybrid input consisting of a bird's-eye view of the car as well as the vehicle's state. This approach exhibited significant improvement over end-to-end agents. Capo et al. [17] explain that this outcome is unsurprising since the partial end-to-end system has embedded driving knowledge via the low-level controller.

Table 2.2 presents a summary of the partial end-to-end methods reviewed. While partial end-to-end systems have shown superior performance compared to end-to-end systems in simulation, their performance on physical cars is yet to be evaluated. Furthermore, there are relatively few research efforts into partial end-to-end systems compared to end-to-end approaches.

Author(s)	**Year**	**Learning method**	**Learned component**	**sim2real approach**
Weiss and Behl [18; 19; 20]	2020	IL	Planner	-
Capo et al. [17]	2022	RL	Planner	-
Evans et al. [21]	2021	RL	Controller	-
Ghignone et al. [14]	2022	RL	Controller	Randomise vehicle model parameters while training

Table 2.2: A summary of partial end-to-end approaches for autonomous racing.

2.4 Evaluation of existing approaches

Classical approaches lead learning-based approaches in terms of real-world driving capabilities. The fact that several classical approaches achieve high speed control of full-size physical vehicles indicates that they are robust towards model mismatches and the sim-to-real transfer [45; 67]. This robustness is achieved through the decoupling of planning and control tasks. Furthermore, both planners and controllers are designed to account for uncertainty in the vehicle dynamics. Planners are designed to generate trajectories that conform to the physical constraints of the vehicle [39; 41; 45]. Meanwhile, controllers are responsible for ensuring robustness to model-mismatch by guiding the vehicle towards the trajectory determined by the planner using feedback loops [48; 49; 50]. Additionally, a recent research trend is improving the vehicle dynamics model online, so that the modelling error is eliminated [9; 59].

In contrast, many learning-based and RL systems are implemented in an end-to-end manner [10; 30]. Furthermore, a significant portion of research in the area of end-to-end RL has focused on model-free techniques for training the agent [16; 33], which excludes

the possibility of enhancing the policy by learning an accurate vehicle model online like classical approaches. Instead, the general approach taken by end-to-end RL methods to increase robustness towards model mismatch is to introduce slight variations to the simulation used during the training process using domain randomisation [15; 16; 33].

A popular strategy for domain randomisation is to introduce noise to the vehicle parameters [33]. However, the optimal policy is highly sensitive to the vehicle model parameters, especially those pertaining to the road surface friction. As such, the policy discovered through domain randomisation is likely to be suboptimal for most scenarios. Domain randomisation is also commonly used in conjunction with policy output regularisation to smooth the control actions [16; 33]. However, approaches that employ these techniques [15; 16; 33] still violate track boundaries, indicating that they do not guarantee the safety of the vehicle.

Partial end-to-end systems present a promising solution for increasing robustness towards model-mismatch by utilising the decoupled structure of classic approaches For instance, Ghigone et al. [14] showed that their learning-based system exhibited excellent robustness to model-mismatch by training an RL agent to learn the task of the controller. However, the advantages of learning-based systems are their ability to learn complex behavior, and relegating the RL agent to the task of path following largely negates this benefit. Furthermore, classical controllers can reliably achieve path following [48; 49; 50].

Several partial end-to-end approaches [17; 18; 19; 20] have utilized a partial end-to-end structure whereby an RL agent is used for planning in conjunction with a classic controller for path tracking. These systems benefit from the agent's heuristic while constructing the plan, while also leveraging the reliability of classical controllers to follow the path [7]. These systems have consistently outperformed end-to-end systems by a significant margin in simulation studies. However, their results have not been validated under conditions where model mismatches are present. Thus, a research gap exists to determine whether a partial end-to-end system that includes a learned planner and a classic controller can offer better performance under model-mismatch conditions than current end-to-end learning-based systems.

To determine whether the performance of current common RL (i.e., end-to-end) systems can be improved by combining an RL planner and classic controller in a classic algorithm structure, this thesis will cover the theoretical foundations of RL agents, the development and implementation of a partial end-to-end system, as well as an end-to-end system. Furthermore, our work will compare the performance of these two systems under practical model-mismatch conditions. This research seeks to contribute to the existing literature by providing insight into the potential benefits of using partial end-to-end systems over current end-to-end systems.

Chapter 3

Reinforcement learning

Reinforcement learning (RL) is a paradigm of machine learning concerned with training an agent to take sequential decisions in an environment in order to maximise a reward signal. This chapter begins with an introduction to Markov Decision Processes (MDPs), which provide the formal framework for modelling the sequential decision-making problem that reinforcement learning aims to solve. A comparison of different reinforcment learning methods is then given. After careful consideration, the twin delay deep deterministic policy gradient (TD3) RL method is chosen for implementation. TD3 is particularly reliant on deep neural networks (DNNs). Consequently, the theoretical backgorund of DNNs are discussed, followed by an in-depth explanation of the TD3 method itself.

3.1 Markov decision processes

The Markov Decision process is a popular mathematical framework for modelling discrete-time decision-making processes where the outcomes are partly random and partly under the control of the *agent* (i.e., the decision maker). MDPs are useful for studying optimisation problems whereby the goal is for the agent to learn a sequence of actions that maximise a *reward* signal. Such optimisation problems are commonly found in the fields of robotics, automated control, and manufacturing [68]. The ability of MDPs to model sequential problems is useful for the autonomous racing problem, where the outcome of the race is determined by a sequence of control actions, and the exact vehicle dynamics model is often uncertain.

Within an MDP, the learner and decision maker is called an agent. Everything outside of the agent constitutes the environment. The agent and environment interact in a sequence of discrete time steps, $t = 0, 1, 2, 3, \cdots$, which is represented in Figure 3.1. At every time step t, the agent receives a representation of the environment known as the state (denoted as S_t), on which basis it selects an action A_t. Furthermore, the environment represents a set of state transition probabilities that are dependent only on the previous state and action. At the next time step $t + 1$, the agent receives a scalar reward R_{t+1}, along with a new state S_{t+1}, which is sampled from the environment [69]. This agent-environment interaction is repeated, giving rise to a trajectory

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots. \tag{3.1}$$

Within the MDP framework, the objective of the agent is described in terms of the reward signal: it must formulate a policy $\pi(a|s)$ that maximises the sum of discounted

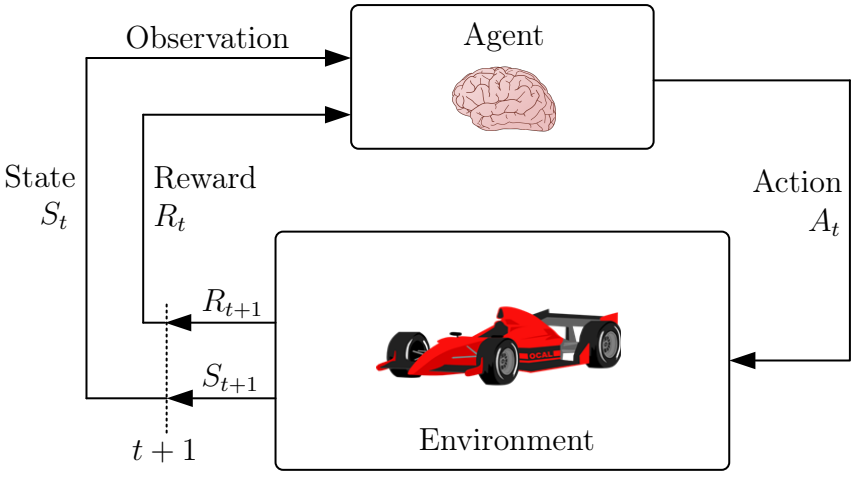


Figure 3.1: The agent environment interaction within an MDP. Adapted from Sutton and Barto [69].

rewards over the course of its entire trajectory. This sum of discounted rewards is called the *return*, and is denoted G_t,

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{3.2}$$

where T denotes the final time step in the trajectory, and γ is a discount factor that determines the weighting of future rewards [69].

3.2 Taxonomy of reinforcement learning

A taxonomy of RL algorithms is shown in Figure 3.2. Broadly speaking, there are two classes of RL techniques; model-based and model free. Model-free RL methods aim to learn a policy solely through direct trial-and-error interactions with the environment, without utilising a representation of the environment dynamics. On the other hand, model-based reinforcement learning involves learning a model of the environment dynamics, which the agent then uses to plan ahead before taking an action [70]. Due to the complexity involved in learning the dynamics of a race-car, we opted to limit the scope of the project to encompass only model-free methods.

Within model-free techniques, a relevant distinction that can be made is that of value-based and policy gradient methods. Value-based RL methods focus on estimating the action-value function, which represents the expected return of being in a particular state taking a specific action. In these methods, the policy is derived from the action-value function. For instance, a greedy policy always selects the action associated with the highest action-value. However, a limitation of value-based methods is that they require the action-space to be discretised [69]. In the context of autonomous racing, where precise control actions are necessary for maintaining vehicle safety, discretized action-spaces fall short in providing the required level of fidelity for control.

Meanwhile, policy gradient techniques form a class of RL methods that explicitly represent the agent's policy as a function. These methods directly optimise the policy,

which serves as a mapping between states and actions, in order to maximise performance. Representing the policy as a differentiable function such as a DNN allows for the representation of continuous state and action spaces [71]. We have therefore chosen to employ policy gradient methods in this study. Specifically, we use a state of the art policy gradient method known as twin delay deep deterministic policy gradient (TD3) [72]. Due to this algorithm's reliance on DNNs to represent the policy, we briefly discuss DNNs before detailing the algorithm itself.

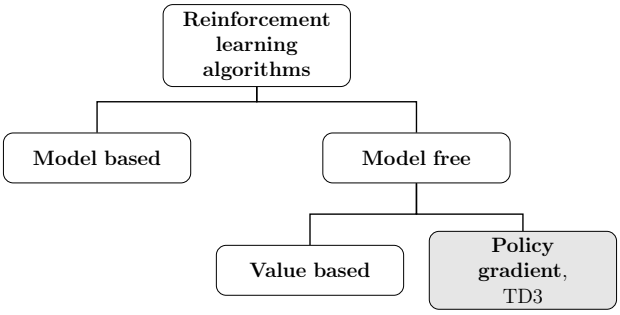


Figure 3.2: A taxonomy of reinforcement learning algorithms.

3.3 Deep neural networks

Deep neural networks (DNNs) are computational models which can represent complex relationships in data. We present the theory surrounding DNNs in the context of supervised learning. In supervised learning, the goal is to approximate a functional mapping, which is denoted as $y = f(\boldsymbol{x})$. However, the exact function $f(\boldsymbol{x})$ is unknown, and instead, we have a data set

$$\mathcal{D} = \{(\boldsymbol{x}^{[1]}, y^{[1]}), (\boldsymbol{x}^{[2]}, y^{[2]}), \cdots, (\boldsymbol{x}^{[n]}, y^{[n]})\} \tag{3.3}$$

consisting of examples where $f(\boldsymbol{x})$ was evaluated at inputs $\boldsymbol{x}^{[i]}$ to produce corresponding outputs $y^{[i]}$. The goal of an DNN is to define an mapping $\hat{y} = \hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$ which approximates $f(\boldsymbol{x})$ with parameters $\boldsymbol{\theta}$. The values of $\boldsymbol{\theta}$ that result in the best approximation are learned using the provided dataset examples [73]. Furthermore, in the context of applying DNNs to the TD3 algorithm, we are particularly interested in a subset of supervised learning known as regression, whereby both the input and output variables are continuous.

Taking inspiration from neural networks in the human brain, the basic component comprising DNNs are *artificial neurons*, which are analogous to their biological counterpart. These artificial neurons apply a non-linear function their inputs, and are organised in layers. We focus our discussion on feedforward neural networks (FNNs), which are a type of DNN whereby information is passed through the layers of the DNN in sequential order. Furthermore, we limit the discussion to *fully connected layers*, in which each of the inputs of neurons in a given layer are connected to he outputs of each of the neurons through a weight, denoted with w. An example of a feedforward DNN with fully connected layers is shown in Figure 3.3. According to Ng et al. [74], the weights connecting each neuron in the kth layer to the preceding layer is given by the matrix

$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \\ w_{m1} & & w_{md} \end{bmatrix}, \tag{3.4}$$

where m is the number of neurons in the kth layer, and d is the number inputs to each neuron (i.e., the number of neurons in the preceding layer). Furthermore, each neuron takes a bias, denoted with b, as input. The biases of neurons in the kth layer are represented as a vector

$$\mathbf{b}^{(k)} = [b_1 \dots b_m]^\intercal, \tag{3.5}$$

The weights and biases of all the layers in the network make up its parameters, and are denoted with θ.

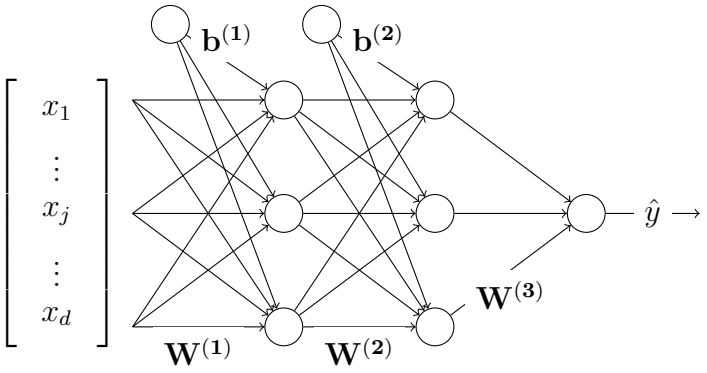


Figure 3.3: A three layer feedforward DNN with three fully connected layers: an input and a hidden layer of three units each, and an output layer of one unit. The weight and bias connections of the kth layer are denoted $\mathbf{W}^{(k)}$ and $\mathbf{b}^{(k)}$ respectively. Furthermore, the DNN receives an input vector $\boldsymbol{x} = [x_1, \dots, x_d]$, and output $\hat{y}$.

The inputs to the kth layer of a DNN are processed in two steps to produce an output. First, a *pre-activation* function $\boldsymbol{a}^{(k)}(\boldsymbol{x})$ adds the biases and product of the weight matrix and the vector output of the preceding layer (denoted as $\mathbf{h}^{(k-1)}$):

$$\boldsymbol{a}(\boldsymbol{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\boldsymbol{x}), \tag{3.6}$$

The output of the kth layer is then calculated by applying an *activation* function, denoted as $g^k(a)$, element-wise to the result of the pre-activation from Equation 3.6:

$$\mathbf{h}^{(k)}(\boldsymbol{x}) = \boldsymbol{g}^{(k)}(\boldsymbol{a}^{(k)}(\boldsymbol{x})). \tag{3.7}$$

Popular choices for activation functions include a linear function whose output is identical to its input

$$g(a) = a, \tag{3.8}$$

the rectified linear unit (ReLU), which takes the maximum between zero and the weighted sum of its inputs

$$g(a) = \max(0, a), \tag{3.9}$$

as well as the hyperbolic tangent

$$g(a) = \tanh(a). \tag{3.10}$$

The process of evaluating an input $\boldsymbol{x}$ to generate an output $\hat{y} = \hat{f}(\boldsymbol{x}, \theta)$ is known as forward propagation. During forward propagation, Equations 3.6 and 3.7 are applied to every layer sequentially from first to last [73], as shown in Algorithm 1. In this algorithm, the input to the DNN is set as the 0th layer of the network, and the output of the DNN corresponds to the output of the final layer.

Algorithm 1: Evaluating a single example input $\boldsymbol{x}$ via forward propagation. Adapted from Goodfellow et al. [73].

Input: An input $\boldsymbol{x}$, DNN with parameters $\{\mathbf{W}, \mathbf{b}\} \in \boldsymbol{\theta}$
Output: DNN output $\hat{y}$

$\mathbf{h}^{(0)} \leftarrow \boldsymbol{x}$
for $k = 1, \cdots, K$ **do**
  $\mathbf{a}^{(k)} \leftarrow \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$
  $\mathbf{h}^{(k)} \leftarrow \mathbf{g}^{(k)}(\mathbf{a}^{(k)})$
end
$\hat{y} \leftarrow \mathbf{h}^{(K)}$

3.3.1 Gradient-based learning

The process through which the optimal set of parameters are determined such that $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$ approximates the mapping between input and output data as accurately as possible is called training. Training is accomplished by incrementally adjusting the parameters of the DNN, such that a *cost function*, denoted as J, is minimised.

The cost function measures the error between the DNN predicted output $\hat{y} = \hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$ and target examples of the true function $y = f(\boldsymbol{x})$. A popular cost function that we consider in this study is the mean squared error (MSE) over a batch of data samples,

$$J(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{f}(\boldsymbol{x}^i, \boldsymbol{\theta}))^2, \tag{3.11}$$

where $\boldsymbol{y} = [y^1, y^2, \ldots, y^N]$ is a vector of training data (often referred to as the *target*), and $\hat{\boldsymbol{x}} = [x^1, x^2, \ldots, x^N]$ is a vector of input values. When this cost function is minimised over all of the available training data, the neural network becomes a better approximator of the true function $f(\boldsymbol{x})$.

The iterative process of adjusting the DNN parameters such that the cost function decreases is known as gradient descent. In gradient descent, the partial derivative of the cost function with respect to the DNN parameters are computed at every time step. The parameters are then adjusted slightly so that the cost decreases, such that the parameters at the next time step are

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha\nabla_{\boldsymbol{\theta}_t}J(\boldsymbol{\theta}_t), \tag{3.12}$$

where α is a scalar controlling the magnitude of the updates, and is known as the learning rate.

A popular gradient descent method that we consider in this study, which is based on Equation 3.12, is adaptive moment estimation (Adam), introduced by Kingma and Ba [75]. Adam computes adaptive learning rates for each parameter in the DNN by keeping track of a decaying average of past gradients, referred to as the momentum.

3.4 Twin delay deep deterministic policy gradient

The twin delay deep deterministic policy gradient (TD3) algorithm by Fujimoto et al. [72], is a popular RL technique applied to solve robotics problems due to its capability to handle continuous action-spaces. In TD3, the policy, which is denoted as $\pi_{\boldsymbol{\phi}}(s)$ and

commonly known as the actor, is represented as a DNN with parameters $\boldsymbol{\phi}$. To update the actor parameters such that the policy improves, gradient ascent is performed with respect to a performance measure $J(\boldsymbol{\phi})$,

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t + \alpha \nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}_t). \tag{3.13}$$

The performance measure J is defined in terms of an *action-value function* (denoted $Q_{\boldsymbol{\theta}}(s, a)$), which is the expected return $\mathbb{E}[G_t]$ from the current state to the end of the episode, given that a specific action is selected in the current state, after which the policy $\pi_{\boldsymbol{\phi}}$ is followed. This action-value function is known as a critic, and is represented by a DNN with parameters $\boldsymbol{\theta}$. In TD3, the idea is to set the performance measure equal to the action-value function

$$J(\boldsymbol{\phi}) = Q_{\boldsymbol{\theta}}(s, a)|_{a=\pi_{\boldsymbol{\phi}}(s)}. \tag{3.14}$$

Updating the actor parameters as in Equation 3.13 will then result in a policy that maximises the expected return, thus solving the MDP. TD3 is presented in Algorithm 2.

In the first two lines of this algorithm, the actor and two critic DNNs are initialised as $\pi_{\boldsymbol{\phi}}$ and $Q_{\boldsymbol{\theta}_q}$, respectively. The actor is parameterised by $\boldsymbol{\phi}$, and the critics by $\boldsymbol{\theta}_q$, where the subscript q indicates the critic number. Two critics are utilised to combat a phenomenon known as *overestimation bias*, whereby states with a low real return are evaluated as having a high expected return.

Subsequently, *target networks* are initialised for each critic and the actor. These target networks are identical in structure to their corresponding counterparts, and are indicated with a prime symbol. Therefore, the target actor and critics are denoted as $\pi'_{\boldsymbol{\phi}}$ and $Q'_{\boldsymbol{\theta}_q}$, respectively. These target networks are updated at a slower rate than their counterparts. According to Mnih et al. [76], utilising these target networks to update the action-value function increases training stability.

Next, a replay buffer, denoted by $\mathcal{B}$, is initialised as an empty array to store the agents experiences. These experiences are stored as a tuple

$$\boldsymbol{e}_t = (s_t, a_t, r_t, s_{t+1}), \tag{3.15}$$

where t is the time step of the experience. By maintaining a history of experiences, each experience can be sampled multiple times to update the parameters of the actor and critics. This approach also helps to break correlations between samples, preventing the DNN parameters from converging to a sub-optimal local minima [76].

After initialising the actor, critics, and replay buffer, the TD3 algorithm executes for M number of of episodes. Each of these episodes represents a trajectory of state, action and reward sequences, as shown in Equation 3.1. At each step of the episode, the algorithm samples an action from the agent, after which the environment provides the next state and reward. Subsequently, the critics are updated to improve the accuracy of action-value estimation. Additionally, the policy and target networks are updated every d time steps. These processes are now explained in more detail.

In line 6, an action is sampled from the actor using

$$a_t = \pi_{\boldsymbol{\phi}}(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma), \tag{3.16}$$

where $\pi_{\boldsymbol{\phi}}(s_t)$ is obtained by forward passing the state through the actor DNN. To encourage exploration, Gaussian noise with a mean of 0 and a standard deviation of σ is added to each action. After sampling the new state , denoted as s_{t+1} and reward, denoted as r_t

Algorithm 2: The twin delay deep deterministic policy gradient (TD3) algorithm, adapted from Fujimoto et al. [72].

1 Initialise critic networks $Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}$ and actor network $\pi_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ and $\boldsymbol{\phi}$
2 Initialise target networks $Q_{\boldsymbol{\theta}'_1}$, $Q_{\boldsymbol{\theta}'_2}$ and $\pi_{\boldsymbol{\phi}'}$ with weights $\boldsymbol{\theta}'_1 \leftarrow \boldsymbol{\theta}_1$, $\boldsymbol{\theta}'_2 \leftarrow \boldsymbol{\theta}_2$, and $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}$
3 Initialise experience replay buffer $\mathcal{B}$

4 **for** episode = *0, M* **do**
5 **for** *t=0, T* **do**
6 Select an action with random exploration noise $a_t \sim \pi_{\boldsymbol{\phi}}(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$
7 Execute action a_t, observe reward r and new state s_{t+1}
8 Store transition tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$
9 Sample mini-batch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{B}$
10 Select target actions for every sample: $\tilde{a} \leftarrow \pi_{\boldsymbol{\phi}'}(s_{i+1}) + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
11 Set TD target: $y \leftarrow r_i + \gamma \min_{q=1,2} Q_{\boldsymbol{\theta}'_q}(s_{i+1}, \tilde{a})$
12 Update critics by minimising the loss: $\boldsymbol{\theta}_q \leftarrow \text{argmin}_{\boldsymbol{\theta}_q}(\frac{1}{N}\sum_{i=0}^{N}(y_i - Q_{\boldsymbol{\theta}_q}(s_i, a_i))^2)$
13 **if** t mod d **then**
14 Update $\boldsymbol{\phi}$ by the deterministic policy gradient: $J(\boldsymbol{\phi}) = \frac{1}{N}\sum Q_{\boldsymbol{\theta}_1}(s_i, a)|_{a=\pi_{\boldsymbol{\phi}}(s_i)}$
15 Update target networks:
16 $\boldsymbol{\theta}'_q \leftarrow \tau\boldsymbol{\theta}_q + (1-\tau)\boldsymbol{\theta}'_q$
17 $\boldsymbol{\phi}' \leftarrow \tau\boldsymbol{\phi} + (1-\tau)\boldsymbol{\theta}'$
18 **end**
19 **end**
20 **end**

from the environment in line 7, the agent's experience tuple is stored in the replay buffer in line 8.

The process of updating the critics occurs in lines 9 to 12. Firstly, a batch of N experiences is sampled from the replay buffer. Each experience is represented as a tuple comprised of a state s_i, action a_i, reward r_i and subsequent state s_{i+1}. For each of these experiences, the target actor is used to select *target actions* associated with the subsequent state,

$$\tilde{a} \leftarrow \pi_{\boldsymbol{\phi}'}(s_{i+1}) + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c), \tag{3.17}$$

after which Guassian noise is added to the target action with upper and lower magnitude bounds of c. The critic DNNs are then updated towards a *TD target* vector, denoted as y, in line 12. Each element in the TD target vector corresponds to a sampled experience i, and is computed by

$$y_i \leftarrow r_i + \gamma \min_{q=1,2} Q_{\boldsymbol{\theta}'_q}(s_{i+1}, \tilde{a}). \tag{3.18}$$

The cost to be utilised in the gradient descent step for the critics, which we denote as $J(\boldsymbol{\theta})$, is computed as the MSE between the TD target and the action-values of the sampled experiences, as evaluated by the critics themselves:

$$J(\boldsymbol{\phi}) = \frac{1}{N}\sum_{i=0}^{N}(y_i - Q_{\boldsymbol{\theta}_q}(s_i, a_i))^2. \tag{3.19}$$

Note that the cost is computed twice by taking the MSE between the target and both critics. The parameters of both critics are then updated using gradient descent with the objective of minimizing the cost function represented in line 12. Utilising two critics in this manner aims to overcome the issue of overestimation bias of action-values.

The updates to the actor and target DNNs occur every d time steps to reduce the variance of their updates. The update to the actor parameters is performed by gradient ascent with respect to the performance measure $J(\boldsymbol{\phi})$. This performance measure is defined as the the average of the first critics evaluation of the sampled states, where the action is given by the current policy $\pi_{\boldsymbol{\theta}}$, over the N experiences sampled from the replay buffer:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum Q_{\boldsymbol{\theta}_1}(s_i, a)|_{a=\pi_{\boldsymbol{\phi}}(s_i)}. \tag{3.20}$$

Therefore, updating the policy maximizes the performance measure, as well as the estimated return for each action.

Lastly, in lines 15 to 17, the target network parameters $\boldsymbol{\phi}'$ and $\boldsymbol{\theta}'$ are updated towards $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ using a *soft update* rule,

$$\begin{aligned} \boldsymbol{\theta}' &\leftarrow \tau\boldsymbol{\theta}' + (1-\tau)\boldsymbol{\theta}, \\ \boldsymbol{\phi}' &\leftarrow \tau\boldsymbol{\phi} + (1-\tau)\boldsymbol{\phi}', \end{aligned} \tag{3.21}$$

where $\tau \ll 1$ is the soft update rate that prevents the target DNN parameters from changing too quickly, which in turn stabilises learning.

3.5 Summary

This chapter presented the theoretical foundation of the Markov Decision Process (MDP), which serves as the formal mathematical framework for the problem that Reinforcement Learning (RL) aims to solve. We also explored the field of RL, with particular emphasis on the TD3 algorirthm.

Within an MDP, the agent interacts with an environment that is defined by a set of state transition probabilities. However, there are many cases (such as autonomous racing) where the state transition probabilities are difficult to represent explicitly. In these cases, simulators are used to represent them implicitly by generating the state transition according to the current state and selected action. In the next chapter, we discuss how F1tenth autonomous racing is modelled as a simulator environment, such that TD3 can be applied to train RL agents to learn how to race.

Chapter 4

Modelling the autonomous racing problem

For the autonomous racing problem to be solved using reinforcement learning, it needs to be modelled such that it can be represented as an environment within the MDP framework. To this end, we implement a simulator that generates a state transition according to a model of the vehicle dynamics, taking the agent's selected action as input. We begin this chapter with discussion of the requirements and rules for F1tenth autonomous racing. An overview of the vehicle dynamics is then given, after which our autonomous racing simulator is detailed.

4.1 Requirements of F1tenth autonomous racing

We have chosen to model an F1tenth race setting because it is a well researched platform with a standardised vehicle model and hardware requirements. F1tenth racing utilises one-tenth scale remote control (RC) car chassis as a vehicle platform. An example of an F1tenth vehicle is shown in Figure 4.1. These vehicles are equipped with all the elements that are necessary for autonomous navigation, including a front facing LiDAR sensor, an Nvidea Jetson module for on-board computing, a servo motor for controlling the front wheels, and a DC motor for controlling the rear wheels. The front wheels are only used for steering, while the rear-wheels are driven. However, the standardised F1tenth simulator is incompatible with fully end-to-end systems due to its inclusion of a velocity controller. We therefore chose to develop an in-house simulator that excludes the velocity controller, which allows us to compare our system to a fully end-to-end system.

The race rules that we consider in our simulator are as follows: a single vehicle competes to complete a single lap of a racetrack in a time trial setting. The vehicle may be started anywhere along the length of the track. The start and finish line then coincides with the starting position of the vehicle. If the vehicle makes contact with the track boundary, which is assumed to be detectable by the LiDAR scanner, the lap is considered failed and the lap time is not counted. We also operate under the assumption that the track is known.

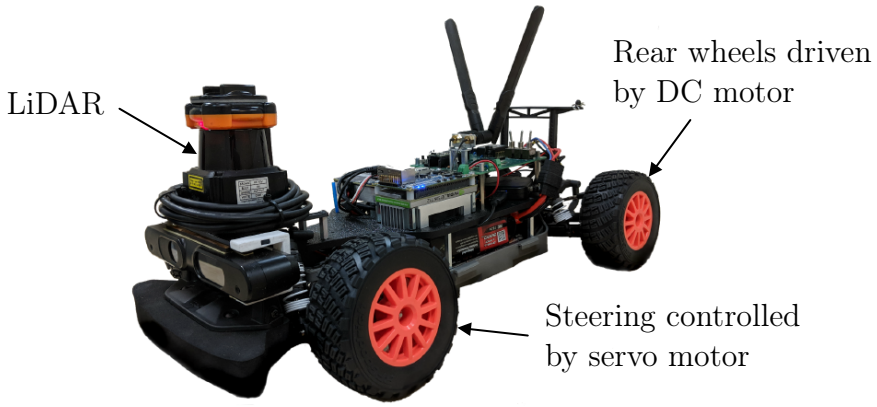


Figure 4.1: An example of an F1tenth vehicle [6], showing the position of the sensors and motors.

4.2 Vehicle dynamics

We start the discussion of our simulator by detailing the vehicle dynamics model that was utilised to accurately represent the motion of the vehicle. An F1tenth vehicle was modelled using the single-track bicycle model by Altoff and Würshing [77]. This model takes vehicle slipping into account, and is accurate even when the vehicle is driven close to its handling limits, thus making it suitable for the racing context. The basic assumptions of the model are that the car is a rigid body, with the two front wheels consolidated into a central wheel. Similarly, the rear wheels are also consolidated into a single center wheel.

Furthermore, the model discretises time, such that each time step k is Δt secnods long:

$$t = k \cdot \Delta t, \tag{4.1}$$

where the time is denoted as t. This discretisation is useful in the context of MDPs, since MDPs themselves are discrete-time. The inputs to the single-track bicycle model at each time step are the current state, denoted as $\mathbf{x}$, and control actions, which are denoted $\mathbf{u}$. The output of the single-track bicycle model is then the time derivative of state, and is denoted as $\dot{\mathbf{x}}$. At every time step k, Euler's method was applied to integrate $\dot{\mathbf{x}}$, thereby determining the state at the next time step.

The state $\mathbf{x}$ is represented as a 7-dimensional vector

$$\mathbf{x} = [s_x,\ \ s_y,\ \ \delta,\ \ v,\ \ \psi,\ \ \dot{\psi},\ \ \beta]^\intercal. \tag{4.2}$$

A description of each variable within the state vector, along with its unit and reference direction is given in Table 4.1. Additionally, the control inputs to the state space model are represented by the vector $\mathbf{u}$. These inputs are a steering rate, denoted as $\dot{\delta}$ (measured in rad/s), and a longitudinal acceleration, denoted as a_{long} (measured in m/s):

$$\mathbf{u} = [\dot{\delta}, a_{\text{long}}]^\intercal. \tag{4.3}$$

The aforementioned state variables and control inputs in illustrated in Figure 4.2.

Altoff and Würshing [77] assume that the vehicle is equipped with motors capable of achieving the desired motion specified by the control inputs, so long as the control

Description	Symbol	Unit	Reference direction
x-Coordinate	s_x	m	Right, from bottom left corner to vehicle CoG
y-Coordinate	s_y	m	Up, from bottom left corner to vehicle CoG
Steering angle	δ	rad	Anti-clockwise, from vehicle heading to steering wheel direction
Longitudinal velocity	v	m/s	Direction of heading
Heading	ψ	rad	Anti-clockwise, from x-axis to vehicle longitudinal axis
Heading rate	$\dot{\psi}$	rad/s	Anti-clockwise
Slip angle	β	rad	Anti-clockwise, from heading to direction of motion

Table 4.1: A description of each state space variable, along with its symbol, unit and reference direction. A depiction of each of these state variables on a diagram of the vehicle is shown in Figure 4.2.

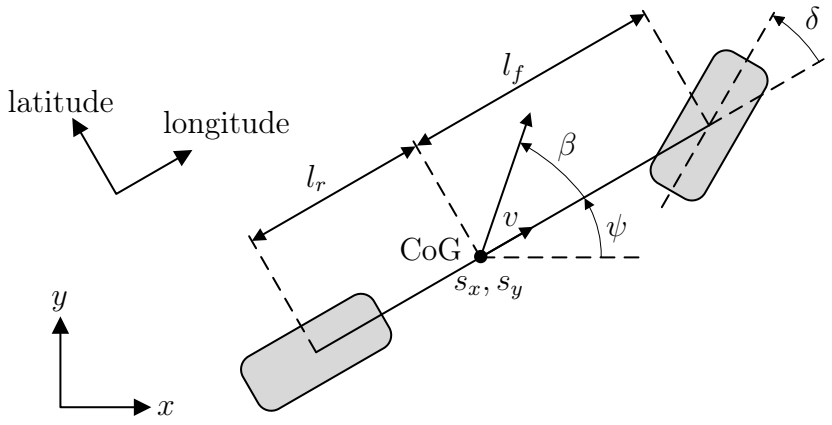


Figure 4.2: The single-track vehicle dynamics model with state variables. The centre of gravity (CoG) is depicted as being l_f m away from the front axle, and l_r m from the rear axle. Furthermore, the longitude and latitude axes are parallel and perpendicular to the forward direction.

inputs are within the physical limitations of the actuators. The following constraints are therefore applied to the steering angle, steering rate, and velocity:

$$\dot{\delta} \in [\underline{\dot{\delta}}, \overline{\dot{\delta}}], \quad \delta \in [\underline{\delta}, \overline{\delta}], \quad v \in [\underline{v}, \overline{v}]. \tag{4.4}$$

Here, an underline denotes the minimum allowable value, while an overline denotes the maximum allowable value.

The vehicle's limited engine power and braking capability impose a constraint on longitudinal acceleration. Additionally, the maximum achievable acceleration and deceleration is influenced by wheel spin. To model this acceleration constraint, the switch velocity, denoted as v_S, is introduced. This velocity represents the threshold above which the engine power is insufficient to induce wheel spin. After taking the switch velocity into account, the acceleration constraint is expressed as

$$a_{\text{long}} \in [\underline{a}, \overline{a}(v)], \quad \overline{a}(v) = \begin{cases} a_{\max} \frac{v_S}{v} & \text{for } v > v_S \\ a_{\max} & \text{otherwise,} \end{cases} \tag{4.5}$$

where $a_{\max}$ is the absolute maximum acceleration that can be achieved without wheel slip, $\underline{a}$ is the maximum deceleration, and $\overline{a}$ is the maximum acceleration that the vehicle can achieve, taking wheel slip into account. A comprehensive description of each constraint, along with its corresponding units, is provided in Table 4.2.

Name	Symbol	Unit	Value
Minimum steering angle	$\underline{\delta}$	rad	-0.4189
Maximum steering angle	$\overline{\delta}$	rad	0.4189
Minimum steering rate	$\underline{\dot{\delta}}$	rad/s	-3.2
Maximum steering rate	$\overline{\dot{\delta}}$	rad/s	3.2
Minimum velocity	$\underline{v}$	m/s	-5
Maximum velocity	$\overline{v}$	m/s	20
Maximum acceleration	a_{max}	m/s^2	9.51
Switching velocity	v_S	m/s	7.319

Table 4.2: Constraint parameters of a standard F1tenth vehicle.

Practically, the constraints from Equations 4.4 and 4.5 are achieved by applying the case statements

$$\dot{\delta} = \begin{cases} 0 & \text{for } (\delta \leq \underline{\delta} \text{ AND } \dot{\delta} \leq 0) \text{ OR } (\delta \geq \overline{\delta} \text{ AND } \dot{\delta} \geq 0) \text{ (condition } C1), \\ \underline{\dot{\delta}} & \text{for NOT } C1 \text{ AND } \dot{\delta} \leq \underline{\dot{\delta}}, \\ \overline{\dot{\delta}} & \text{for NOT } C1 \text{ AND } \dot{\delta} \geq \overline{\dot{\delta}}, \\ \dot{\delta} & \text{otherwise,} \end{cases}$$

$$a_{\text{long}} = \begin{cases} 0 & \text{for } (v \leq \underline{v} \text{ AND } a_{\text{long}} \leq 0) \text{ OR } (v \geq \overline{v} \text{ AND } a_{\text{long}} \geq 0) \\ & \text{(condition } C2), \\ \underline{a} & \text{for NOT } C2 \text{ AND } a_{\text{long},d} \leq \underline{a}, \\ \overline{a(v)} & \text{for NOT } C2 \text{ AND } a_{\text{long},d} \geq \overline{a}(v), \\ a_{\text{long}} & \text{otherwise,} \end{cases} \tag{4.6}$$

to the input vector $\mathbf{u}$, before applying them as input to the state space equations.

After applying these constraints to the input vector $\mathbf{u}$, non-linear state-space equations with inputs $\mathbf{u}$ and $\mathbf{x}$ are applied. Altoff and Würshing [77] define the state-space model as a piece-wise function that is dependent on the velocity. For large velocities, a dynamic bicycle model denoted by $f_1(\mathbf{x}, \mathbf{u})$ is used, as it accounts for tire slip and accurately represents the motion close to the physical limits of the vehicle. However, this dynamic bicycle model becomes singular for small velocities. Therefore, a kinematic bicycle model $f_2(\mathbf{x}, \mathbf{u})$ that does not take tire slip into account is used for velocities slower than 0.1 m/s. The piece-wise defined dynamics model is then

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{cases} f_1(\mathbf{x}, \mathbf{u}) & \text{if } |v| \geq 0.1 \text{ m/s}, \\ f_2(\mathbf{x}, \mathbf{u}) & \text{else.} \end{cases} \tag{4.7}$$

The dynamic bicycle model $f_1(\mathbf{x}, \mathbf{u})$ is described by the set of equations

$$\begin{aligned}
\dot{s_x} &= \upsilon \cos(\psi + \beta), \\
\dot{s_y} &= \upsilon \sin(\psi + \beta), \\
\dot{\delta} &= \dot{\delta}, \\
\dot{\upsilon} &= a_{\text{long}}, \\
\dot{\psi} &= \dot{\psi}, \\
\ddot{\psi} &= \frac{\mu m}{I_z(l_r + l_f)}(l_f C_{S,f}(g l_r - a_{\text{long}} h_{cg})\delta + (l_r C_{S,r}(g l_f + a_{\text{long}} h_{cg}) \\
&\quad - l_f C_{S,f}(g l_r - a_{\text{long}} h_{cg}))\beta - (l_f^2 C_{S,f}(g l_r - a_{\text{long}} h_{cg}) + l_r^2 C_{S,r}(g l_f + a_{\text{long}} h_{cg}))\frac{\dot{\psi}}{\upsilon}), \\
\dot{\beta} &= \frac{\mu}{\upsilon(l_r + l_f)}(C_{s,f}(g l_r - a_{\text{long}} h_{cg})\delta - (C_{S,r}(g l_f + a_{\text{long}} h_{cg}) + C_{S,f} \\
&\quad (g l_r - a_{\text{long}} h_{cg}))\beta + (C_{S,r}(g l_f + a_{\text{long}} h_{cg}) l_r - C_{S,f}(g l_r - a_{\text{long}} h_{cg}) l_f)\frac{\dot{\psi}}{\upsilon}) - \dot{\psi},
\end{aligned} \tag{4.8}$$

and the kinetic bicycle model $f_2(\mathbf{x}, \mathbf{u})$ by

$$\begin{aligned}
\dot{s_x} &= \upsilon \cos(\psi + \beta), \\
\dot{s_y} &= \upsilon \sin(\psi + \beta), \\
\dot{\delta} &= \dot{\delta}, \\
\dot{\upsilon} &= a_{\text{long}}, \\
\dot{\psi} &= \frac{\upsilon \cos(\beta)}{l_{wb}} \tan(\delta), \\
\ddot{\psi} &= \frac{1}{l_{wb}}(a_{\text{long}} \cos(x_7) \tan(\delta) - x_4 \sin(\beta) \tan(\delta)\dot{\beta} + \frac{\upsilon \cos(\beta)}{\cos^2(\delta)}\dot{\delta}), \\
\dot{\beta} &= \frac{1}{1 + (\tan(\delta)\frac{l_r}{l_{wb}})^2} \cdot \frac{l_r}{l_{wb} \cos^2(\delta)}\dot{\delta},
\end{aligned} \tag{4.9}$$

where m is the vehicle mass, I_z is the moment of inertia about the z axis of the vehicle (i.e., a vertical axis that passes through the CoG), l_f is the distance from the CoG to the front axle, l_r is the distance from the CoG to the rear axle, and h_{cg} is the height of CoG. Furthermore, the parameters $C_{S,f}$ and $C_{S,r}$ are the cornering stiffness coefficients of the the front and rear tires, respectively. The tire cornering stiffness coefficient is the ratio between the lateral force acting on the tire, and its slip angle, which is the angle between the direction the wheel is pointing, and its direction of travel. Additionally, the road surface friction is denoted μ. The values for these vehicle parameters were identified for a standard F1tenth vehicle [6], and are summarised in Table 4.3.

The state space Equations 4.8 and 4.9 yield the derivative of the state with respect to time. Consequently, to determine the state at the next time step, we employ a numerical integration

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \dot{\mathbf{x}}\Delta t, \tag{4.10}$$

known as Euler's method. This method allows us to incrementally update the state by adding the product of the state derivative and the time step, which is sufficiently accurate when Δt is small. We chose Δt as 0.01 seconds.

Symbol	Description	Unit	Value
m	Mass	kg	3.74
I_z	Moment of inertia about z axis	kg	0.04712
l_f	Distance from CoG to front axle	m	0.1587
l_r	Distance from CoG to rear axle	m	0.17145
h_{cg}	Height of CoG	m	0.074
$C_{S,f}$	Cornering stiffness coefficient (front)	1/rad	4.718
$C_{S,r}$	Cornering stiffness coefficient (rear)	1/rad	5.4562
μ	Road surface friction coefficient	-	1.0489

Table 4.3: Vehicle model parameters for the single track model. The parameters were identified for a standard F1tenth vehicle.

4.3 Simulation environment

The vehicle single track vehicle dynamics model discussed in Section 4.2 was utilised to create a custom F1tenth simulator that adheres to the rules specified in Section 4.1. The simulator runs an initialisation procedure at the start of each episode, after which it is sampled at every time step. Algorithm 3 details the procedure that the simulator executes at the beginning of an episode. This procedure takes as input a bird's-eye image of the racetrack, as well as the coordinates, velocity and heading at which to initialise the vehicle.

In line 1 of the episode start procedure, the simulator generates an occupancy grid from the provided image. The occupancy grid is represented as a binary array, with each element corresponding to a specific coordinate on the track. This occupancy grid is used to detect whether the vehicle has collided with the track boundary during a lap.

Algorithm 3: The simulator initialisation procedure.

Input: An image of a track, starting position, velocity and heading: $[s_x, s_y, \upsilon, \psi]$

1 Generate occupancy grid
2 Find centerline
3 Start vehicle at random point along centerline
4 Set start/ finish line

Once the occupancy grid is generated, the simulator proceeds to determine the centerline of the track. The centerline refers to a line that stretches around the track, and maintains an equal distance from either side of the track boundaries. To represent the centerline, a cubic spline [78] is employed.

Subsequently, the vehicle is initialized with an x and y coordinate, velocity, as well as heading. Typically, this involves selecting a random point along the length of the centerline and applying a small offset in the x and y directions. The remaining state variables (i.e., the steering angle, heading rate and slip angle) are initialised with a zero value. Following this, the start and finish line are defined as perpendicular to the track's centerline, intersecting with the initial coordinates of the vehicle.

Once initialised, the simulator is sampled at discrete time steps to forward simulate the vehicle dynamics according to the control inputs. This is done using the single-track bicycle model from Section 4.2. The simulator receives an input action a_k at time step k and produces outputs at time step $k+1$. The output of the simulator is an observation,

denoted o_{k+1}, and reward, denoted as r_{k+1}, which is consistent with the MDP environment definition. The interaction between the autonomous racing algorithm and simulator is shown in Figure 4.3. Additionally, the procedure that the algorithm executes at every time step is detailed in Algorithm 4.

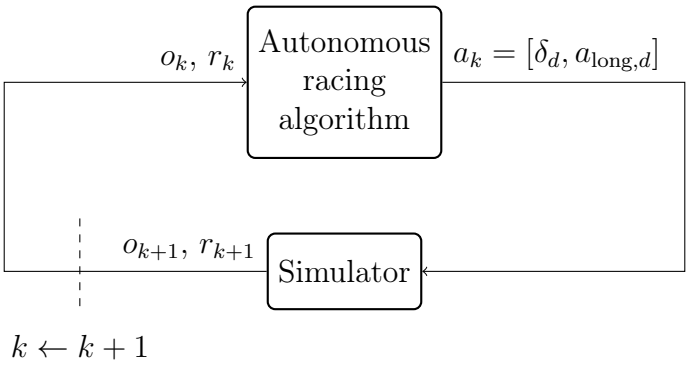


Figure 4.3: The interaction between the racing algorithm and simulator at every time step is analogous to that of the agent and MDP.

The input to the simulator at every time step is a desired steering angle δ_d, and a desired longitudinal velocity, $a_{\text{long},d}$, where the subscript d indicates the desired value provided by the autonomous racing algorithm. The *observation* that it outputs encompasses a LiDAR scan with L equispaced beams spanning a 180-degree field of view, as well as the vehicle's pose. The LiDAR scan is a vector whose elements correspond to the range measurement of one of the LiDAR beams. The vehicle's pose comprises several state variables. Specifically the x-coordinate s_x, the y-coordinate s_y, longitudinal velocity v and heading ψ. The observation is therefore denoted as a vector

$$o_k = [\underbrace{s_x, s_y, \delta, v, \psi}_{\text{Pose}}, \underbrace{l_0, l_1, l_2, \cdots L}_{\text{LiDAR scan}}]. \tag{4.11}$$

Before outputting the observation, a small amount of noise with a zero mean can be added to each element in the observation vector. The standard deviation of noise added is discussed in the following chapters.

Algorithm 4: The simulator execution at every time step.

Input: Control actions $\dot{\delta}_d$ and a_{long_d} at time step k.
Output: An observation o_{k+1}, reward r_{k+1} at time step $k+1$.

1 Simulate servo motor: $\dot{\delta} = \frac{\delta_d - \delta}{|\delta_d - \delta|}\bar{\dot{\delta}}$,
2 Update state: $\mathbf{x}_{k+1} = \mathbf{x}_k + f(\mathbf{x}_k, \mathbf{u}_k)\Delta t$
3 Generate LiDAR scan: $[l_0, l_1, \cdots L]$
4 Sample the observation: $o_{k+1} = [s_x, s_y, \delta, v, \psi, l_0, l_1, l_2, \cdots L]$
5 Check for collisions
6 Get distance travelled: $\Delta D = D_{k+1} - D_k$
7 Generate reward: $r_{k+1} = r(s_k, a_k)$

The vehicle dynamics are simulated in line 1 of Algorithm 4. When simulating these dynamics, we abstract the longitudinal acceleration controller and assume that the vehicle

can accelerate at the desired rate, so long as the desired rate is within the bounds of the upper and lower acceleration limits specified by Equation 4.5. The desired acceleration $a_{\text{long},d}$ then serves as the control input a_{long} to the state space model. However, the steering dynamics are not abstracted. Instead, the rate of change of steering angle by steering servo motor calculated as

$$\dot{\delta} = \frac{\delta_d - \delta}{|\delta_d - \delta|}\bar{\dot{\delta}}. \tag{4.12}$$

The constraints from Equation 4.6 are applied to the longitudinal acceleration and steering rate, after which Equations 4.7 and 4.10 are used to calculate the new state in line 2.

After updating the state using the state space model, we generate the LiDAR scan and sample from the state variables to create the observation. The simulator then checks the vehicle position against the occupancy grid to see if a collision has occurred. If the vehicle has collided with the track boundary, a terminal state is reached and the episode ends. Additionally, the episode also ends if the vehicle has completed one lap. To ascertain whether a lap is complete, we calculate the distance travelled from the start along the centerline between the subsequent previous time step as

$$\Delta D = D_{k+1} - D_k, \tag{4.13}$$

where distance D_k represents the point on the centerline closest to the vehicle at time step k. The ΔD values at each time step are accumulated, and if the sum is equal to or greater than the track length, the vehicle has completed one lap. An illustration of the quantity ΔD is shown in Figure 4.4. The distance travelled along the centerline is also useful in calculating the reward signal

$$r_{k+1} = r(\mathbf{x}_k, a_k), \tag{4.14}$$

which appears in line 7 of Algorithm 4, and whose details are discussed in the next chapter.

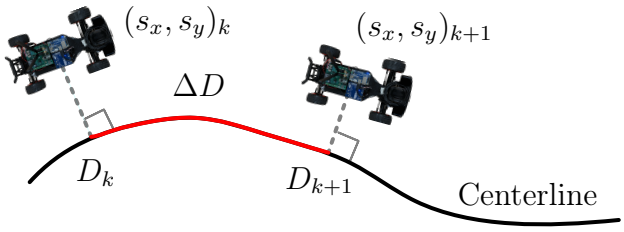


Figure 4.4: An F1tenth vehicle moving along the track centerline. The red section of the centerline indicates ΔD.

4.4 Summary

This chapter introduced the racing scenario, vehicle modelling, as well as simulator. By allowing F1tenth vehicles to race alone on the track, the setting that we consider provides

an opportunity for autonomous racing algorithms to compete in a time-trial style environment. Furthermore, the F1tenth vehicle is modelled using single-track bicycle dynamics by Altoff and Würshing [77], which accurately predicts the motion of the vehicle close to the handling limits. The dynamics model is incorporated into a simulator that encompasses both the race car dynamics and the track environment. By discretizing time, the simulator conforms to the time discrete framework of the MDP. At each time step, the simulation receives a control input from the autonomous racing algorithm and generates an observation corresponding to the updated state of the environment.

Approaches to solving the autonomous racing problem that utilise an end-to-end design have an agent whose inputs and outputs are directly passed from and given to the simulator. In the next chapter, we utilise our racing simulation within an MDP environment to train such an end-to-end RL agent to race.

Chapter 5

End-to-end autonomous racing

Having introduced our simulation environment, we formulate an end-to-end solution method in which a reinforcement learning (RL) agent directly predicts controller commands based on observation information. This end-to-end agent is employed as a baseline to compare our partial end-to-end algorithm against, as similar end-to-end approaches are commonly used to solve the racing problem [10; 15; 16; 26; 28; 29; 30; 31; 32; 33; 34].

We begin this chapter by discussing the design of the end-to-end racing algorithm. Subsequently, we show how the TD3 RL algorithm is used to train an end-to-end agent, followed by a detailed exposition of evaluation procedures. We then experimentally determine the optimal values for each hyper-parameter for an agent racing on a relatively simple race track, before presenting agents capable of driving on more complex race tracks. The performance of end-to-end agents under conditions where vehicle modelling errors are present is also investigated, along with the effectiveness of domain randomisation as a technique to improve performance under these conditions.

5.1 End-to-end racing algorithm

Our end-to-end autonomous racing algorithm is composed of an RL agent and a velocity constraint. The agent maps an observation sampled from the simulator to desired longitudinal acceleration ($a_{\text{long},d}$) and steering angle (δ_d) control commands. The velocity constraint then modifies the acceleration commands to ensure that the vehicle remains within safe velocity bounds. The steering angle from the agent and acceleration from the velocity constraint component are passed to the simulator described in Chapter 4. This end-to-end framework is depicted in Figure 5.1.

Importantly, the simulator and velocity constraint components are grouped together in the environment. This is because the definition of the MDP given in Section 3.1 solely encompasses an agent and an environment. In fact, to ensure conformity between the end-to-end algorithm and the MDP definition, all of the racing algorithm components apart from the agent are considered as part of the environment, and executed in unison with the rest of the environment. Furthermore, due to the simulator's time step being chosen as 0.01 seconds, the environment components are sampled at a frequency of 100 Hz. The agent is sampled at a slower rate of f_{agent} Hz.

The end-to-end agent, which comprises a neural network, is shown in Figure 5.2. To ensure uniformity across all observation vector elements, each element in the input vector is normalized to the range $[0, 1]$. The neural network's design consists of three fully connected layers, with m_1, m_2, and 2 neurons in the input, hidden, and output layers,

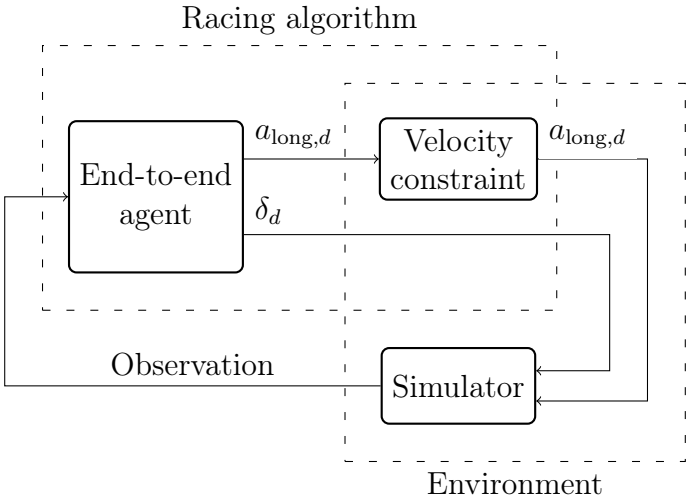


Figure 5.1: The end-to-end racing algorithm, which is comprised of an RL agent which outputs control actions, and a velocity constraint. The velocity constraint and simulator are both considered part of the environment.

respectively. The first two layers are ReLU-activated, while the output layer is activated by a hyperbolic tangent function to normalize the neural network output to the range $(-1, 1)$. While the number of neurons in the first two layers are determined empirically, the two neurons in the output layer correspond to the steering and acceleration actions. Scaling factors are applied to their outputs so that the selected steering and acceleration actions fall within the range $(\underline{\delta}, \overline{\delta})$ and $(\underline{a}, \overline{a})$ from Table 4.2, respectively.

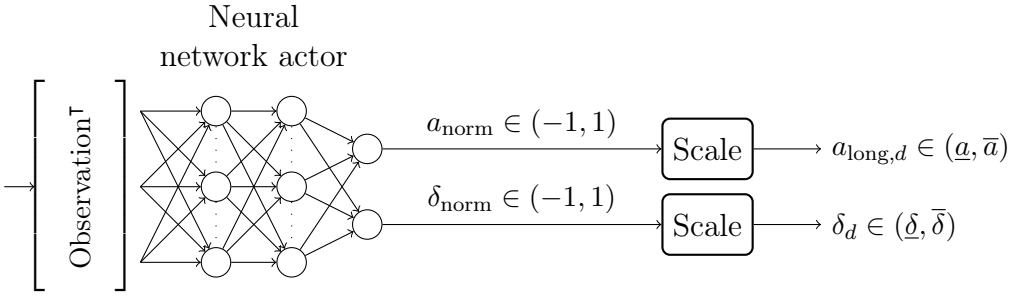


Figure 5.2: The end-to-end agent. The outputs of the neural network are scaled to the ranges of a_{long} and δ in Table 4.2.

While the steering angle is passed directly to the simulator, the longitudinal action is first modified by the velocity constraint component to ensure that the velocity of the vehicle remains within safe bounds,

$$a_{\text{long},d} \leftarrow \begin{cases} 0 & \text{for } v \geq v_{\max}, \\ 0 & \text{for } v \leq v_{\min}, \\ a_{\text{long},d} & \text{otherwise}, \end{cases} \tag{5.1}$$

before being passed to the simulator. In Equation 5.1 $v_{\max}$ and $v_{\min}$ are the imposed maximum and minimum allowable velocities.

5.2 Applying TD3 to end-to-end autonomous racing

We applied the TD3 RL algorithm from Section 3.4 to train the end-to-end agent. Several adaptations to the original TD3 algorithm were made to ensure its compatibility with the end-to-end racing algorithm. The adapted TD3 is shown in Algorithm 5.

Algorithm 5: Twin delay deep deterministic policy gradient

Input: Table 5.1 parameters
Output: Trained actor DNN $\pi_{\boldsymbol{\phi}}$

Initialise critic networks $Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}$ and actor network $\pi_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ and $\boldsymbol{\phi}$
Initialise target networks $Q_{\boldsymbol{\theta}'_1}$, $Q_{\boldsymbol{\theta}'_2}$ and $\pi_{\boldsymbol{\phi}'}$ with weights $\boldsymbol{\theta}'_1 \leftarrow \boldsymbol{\theta}_1$, $\boldsymbol{\theta}'_2 \leftarrow \boldsymbol{\theta}_2$, and $\boldsymbol{\phi}' \leftarrow \boldsymbol{\phi}$
Initialise experience replay buffer $\mathcal{B}$
while MDP time steps $< M$ **do**
  Reset simulator (start new episode)
  for *t=0, T* **do**
    Sample action with exploration noise from end-to-end agent, $a_t = [a_{\text{long},d}, \delta_d] \leftarrow \text{scale}(\pi_{\boldsymbol{\phi}}(o_t) + \epsilon),\ \ \epsilon \sim \mathcal{N}(0, \sigma_{\text{action}})$
    for *n=0, N* **do**
      Modify $a_{\text{long},d}$ according to Equation 5.1 to limit velocity
      Simulator executes action a_t
      Observe environment step reward $r_{t,n}$
      Update MDP one step reward: $r_t = r_t + r_{t,n}$
      Sample observation with noise, $o_t \leftarrow o_t + \epsilon,\ \ \epsilon \sim \mathcal{N}(0, \sigma_{\text{observation}})$
    end
    Store transition tuple (o_t, a_t, r_t, o_{t+1}) in $\mathcal{B}$
    Sample mini-batch of B transitions (o_i, a_i, r_i, o_{i+1}) from $\mathcal{B}$
    Select target actions: $\tilde{a} \leftarrow \pi_{\boldsymbol{\phi}'}(o_{t+1}) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    Set TD target: $y_i \leftarrow r_i + \gamma \min_{q=1,2} Q_{\boldsymbol{\theta}'_q}(o_{i+1}, \tilde{a})$
    Update critics by minimising the loss: $J_{\boldsymbol{\theta}_q} \leftarrow \frac{1}{N}\sum_i^N (y_i - Q_{\boldsymbol{\theta}_q}(o_i, a_i))^2$
    if $t \mod d$ **then**
      Update $\boldsymbol{\phi}$ by the deterministic policy gradient: $\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) = \frac{1}{N}\sum_i^N \nabla_a Q_{\boldsymbol{\theta}_1}(o_i, a)|_{a=\pi_{\boldsymbol{\phi}}(o_i)} \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(o_i)$
      Update target networks:
      $\boldsymbol{\theta}'_q \leftarrow \tau\boldsymbol{\theta}_q + (1-\tau)\boldsymbol{\theta}'_q$
      $\boldsymbol{\phi}' \leftarrow \tau\boldsymbol{\phi} + (1-\tau)\boldsymbol{\theta}'$
    end
  end
end

Note that in the context of racing, agents receive only partial information about the

state of the environment. This is because the pose and LiDAR scan do not fully capture the environment state. Hence, the racing environment is only partially observable. As such, the input to a racing agent is therefore an observation, denoted as o, rather than the complete environment state s. This notation conforms to the notation used for the output of the simulator in Chapter 4. However, we use the same notation for time steps as in Chapter 3, denoting a time step as t, rather than k.

In line 1 of Algorithm 5, the actor ($\pi_{\boldsymbol{\phi}}$) and critics ($Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$) are initialised. The end-to-end agent shown in Figure 5.2 is the actor $\pi_{\boldsymbol{\phi}}$. For simplicity, the critics have an identical structure which is analogous to that of the actor. We therefore describe the details of only one critic, which is depicted in Figure 5.3. The critic DNN receives a vector input comprised of observation and control actions normalised to the range $(-1, 1)$. It comprises three fully connected layers. The input and hidden layers are identical to the actor, having m_1 and m_2 neurons with ReLU activation functions, respectively. The output layer comprises a single neuron with a linear activation function. The output of this layer is the action-value, and is denoted $Q_{\boldsymbol{\theta}}(o, a)$. Additionally, the target networks are initialised identically to their counterparts.

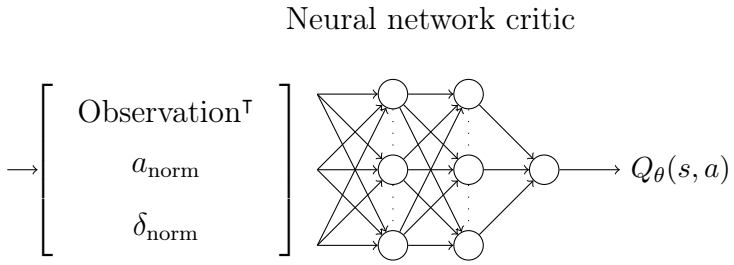


Figure 5.3: The critic DNN. Its input is a vector containing a normalised observation and action pair, and its output is an action-value.

After initializing the replay buffer in line 3, the TD3 algorithm enters a while loop which executes a number of episodes (lines 5-22). However, rather than limiting the number of episodes, we set a limit on the number of MDP time steps, denoted as M, as it is a more accurate indicator of the training time and the number of actor and critic updates. Each episode starts by resetting the simulator, and ends when the simulator indicates that the vehicle has crashed or finished.

In line 7 of Algorithm 5, an action is sampled from the end-to-end agent by forward passing the observation through the actor DNN. Gaussian noise with a zero mean and a standard deviation of σ_{action} is added to the normalised output (a_{norm} and δ_{norm}), which is then scaled to generate a longitudinal acceleration and a steering angle.

The action sampled from the agent in line 7 is executed by repeatedly sampling the environment (lines 8-13). Our implementation for sampling the environment differs from the standard implementation of TD3 given in Algorithm 2. This is because the sample rate of the components inside the environment is higher than the agent, whereas the definition of the MDP given in Section 3.1 requires that the agent and environment is sampled at the same rate. As such, the environment components will be sampled multiple times in-between agent sampling periods. We therefore define an MDP step as N environment samples, where

$$N = \frac{100}{f_{\text{agent}}}. \tag{5.2}$$

In Equation 5.2, N is a whole number. The environment is sampled by applying the velocity constraint from Equation 5.1 to limit the agent's selected longitudinal action, and then executing one simulator step. This is followed by sampling the reward signal from the simulator in line 11.

The reward signal is designed to closely approximate the objective of minimizing lap time for high reward discount rates. Specifically, the is rewarded agent for the distance it travels along the centerline between the current and previous time step, while being penalised by a small amount on every time step, as described by Fuchs et al. [30]. In addition, a large penalty is imposed on the agent if it collides with the track boundary. As a result, the reward signal is expressed as the piece-wise function

$$r(s_t, a_t) = \begin{cases} r_{\text{collision}} & \text{if collision occurred} \\ r_{\text{dist}}(D_t - D_{t-1}) + r_{\text{time}} & \text{otherwise.} \end{cases} \tag{5.3}$$

Here, $r_{\text{collision}}$, r_{dist}, and r_t represent the penalty for collisions, the reward for distance traveled, and the penalty for each time step, respectively. Notably, this reward signal is similar to those used in numerous prior works [10; 15; 29].

The reward signal is accumulated over the sequence of N steps during which the environment is sampled in line 12. In line 15, the transition tuple is stored, which consists of the observation before sampling the environment N times, as well as the observation and accumulated reward after sampling the environment N times.

The remaining steps of Algorithm 5 are identical to the standard implementation of TD3 described in Algorithm 2. Specifically, a mini-batch of B transitions is sampled from the replay buffer in line 16. Next, the target actor network is employed to select actions for each observed sample, which in turn are used to update the critics. To ensure the stability of the learning process, the actor and the target networks are updated every d steps from lines 20 to 23. Furthermore, the target networks are updated via a soft update which is controlled by the target update rate parameter τ.

After the training procedure is completed, Algorithm 6 is utilised to evaluate the trained agents. Under evaluation conditions, training is halted and the weights of the DNNs are not updated. Furthermore, no exploration noise is added to the agents selected actions. However, Gaussian noise is added to the observation vector to mimic practical

Algorithm 6: Evaluating the end-to-end algorithm without exploration noise, and with observation noise.

Input: Trained actor DNN $\pi_{\boldsymbol{\phi}}$
Output: Lap times, collisions over 100 laps

```
1  Initialise actor DNN π_φ with weights φ from training
2  for episode = 1, 100 do
3  |  for t=0, T do
4  |  |  Select control action a_t = [a_long,d, δ_d] ~ scale(π_φ(o_t + ε))
5  |  |  for n=0, N do
6  |  |  |  Modify a_long,d according to Equation 5.1 to limit velocity
7  |  |  |  Simulator executes action a_t
8  |  |  end
9  |  end
10 end
```

sensor data in simulation. This Gaussian noise has standard deviations of 0.025 m for x and y coordinates, 0.05 rads for heading, 0.1 m/s for velocity, and 0.01 m for LiDAR scan. Each agent completed 100 laps under these evaluation conditions.

5.3 Empirical design and hyper-parameter values

The optimal values for the hyper-parameters introduced in Sections 5.1 and 5.2 cannot be derived, and require experimentation to be determined empirically. Furthermore, hyper-parameters are sensitive to the track. The following five sections of this chapter detail the experiments that were undertaken to determine a locally optimal set of hyper-parameters for agents racing on a relatively simply track named Porto. The selected hyper-parameters are listed in Table 5.1. Additionally, the average learning curve for 10 agents racing on this track using this set of hyper-parameter is shown in Figure 5.4.

Hyper-parameter	Symbol	Value
Algorithm		
Maximum time steps	M	$1.5 \cdot 10^5$
Target update rate	τ	$5 \cdot 10^{-3}$
Replay buffer size	$\mathcal{B}$	$5 \cdot 10^5$
Replay batch size	B	400
Exploration noise standard deviation	σ_{action}	0.1
Reward discount rate	γ	0.99
Agent samples between network updates	d	2
Agent sample rate	f_{agent}	5 Hz
Target action noise standard deviation	$\tilde{\sigma}$	0.2
Target action noise clip	c	0.5
Reward signal		
Distance reward	r_{dist}	0.25
Time step penalty	r_{time}	0.01
Collision penalty	$r_{\text{collision}}$	-10
Observation		
Number of LiDAR beams	L	20
Neural network		
Learning rate	α	10^{-3}
Neurons in input layer	m_1	400
Neurons in hidden layer	m_2	300
Velocity constraints		
Minimum velocity	v_{min}	3 m/s
Maximum velocity	v_{max}	5 m/s

Table 5.1: Selected values of hyper-parameters for the end-to-end racing algorithm on the Porto track.

To select each hyper-parameter value, we repeatedly trained agents using Algorithm 5 with various values of the hyper-parameter under consideration while keeping all other hyper-parameters fixed at the values listed in Table 5.1. When evaluating agents, we are particularly interested in the rate at which they successfully complete laps, as well as their

lap time during and after training. Furthermore, to ensure consistency in the results, we trained and evaluated multiple agents for each set of hyper-parameters. Specifically, we chose to train three agents for each hyper-parameter set due to time constraints.

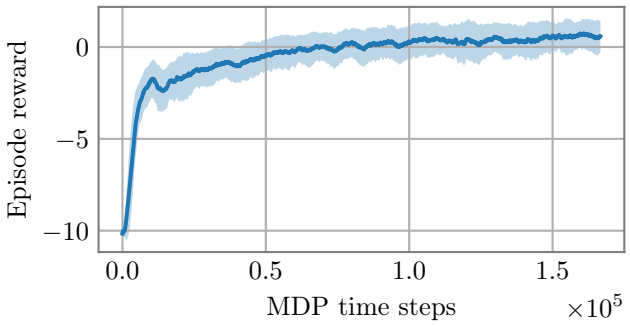


Figure 5.4: Average learning curve for 10 end-to-end agents trained on the simple Porto track.

5.4 TD3 hyper-parameters

We performed experiments to determine values for the TD3 algorithm hyper-parameters that result in good performance for the Porto track. These hyper-parameters were the number of MDP time steps M, agent sample rate f_{agent}, target update rate τ, replay buffer size $\mathcal{B}$, replay batch size B, standard deviation of exploration noise σ_{action}, reward discount rate γ, and agent samples in-between DNN updates d.

First, an appropriate number of time steps to train the agent for was determined. The objective for determining the length of the training time was to ensure that the agent demonstrates satisfactory performance under evaluation conditions by racing quickly and consistently avoiding crashes. Ending training too soon may result in poor agent performance, while training for too many time steps long may result in unnecessarily prolonged training times. To achieve this, a set of three agents with the hyper-parameters listed in Table 5.1 were trained. These agents were evaluated using Algorithm 6 at 100 episode intervals during training. The percentage of failed laps and lap time under evaluation conditions are depicted as a function of training time in Figure 5.5.

We observe that during the early stages of training, both lap time and success rate improved rapidly. However, it takes a considerable amount of time before the agent consistently completes all of its laps under evaluation conditions. Considering these results, we determined that $1.5 \cdot 10^5$ MDP time steps is an appropriate length for training an end-to-end agent.

The optimal value for the rate at which actions are sampled from the agent, denoted as f_{agent}, was then determined. We investigated agent sample rates in the range of 3 Hz to 50 Hz. For each sample rate investigated, three agents were trained with the remaining hyper-parameters set equal to those listed in Table 5.1. Figure 5.6 shows the average failed laps and lap time of agents racing under evaluation conditions as a function of f_{agent}. From this figure, we observe that agents trained with sampling rates higher than 5 Hz tend to crash, as well race slowly. This outcome may be attributed to the fact that when a higher sampling rate is used, the agent needs to learn longer action sequences to complete a lap, leading to a more complex learning problem. The value of f_{agent} was

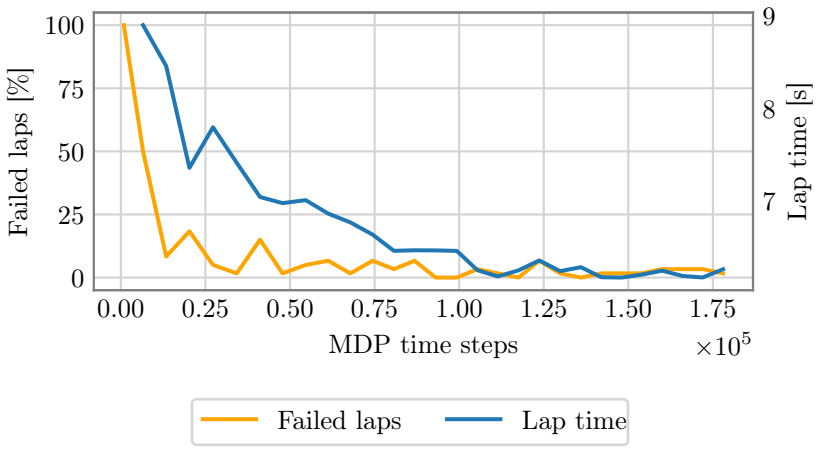


Figure 5.5: Percentage failed laps (left vertical axis) and lap time (right vertical axis) of three agents tested under evaluation conditions at 100 episode intervals during training.

set to 5 Hz as it resulted in the minimum number of failed laps during evaluation. It is notable that 5 Hz is a relatively slow sampling rate compared to classical controllers. For instance, Li et al. [79] develop path tracking controllers with sampling rates up to 100 Hz.

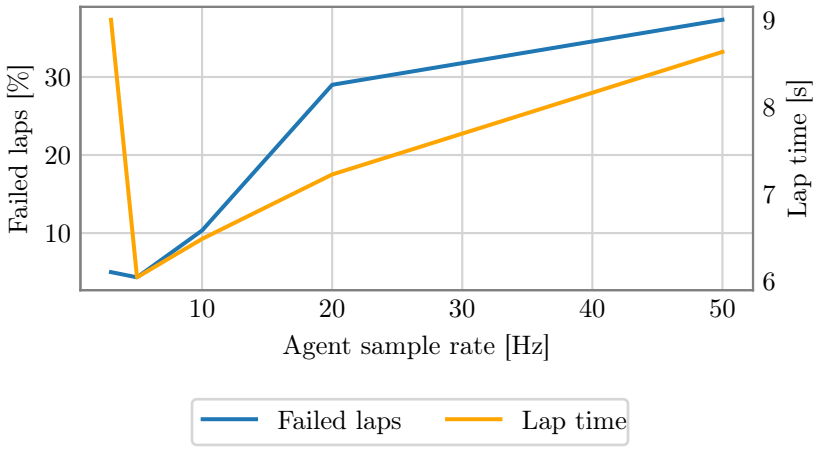


Figure 5.6: Training time (left vertical axis) and percentage failed laps (right vertical axis) of three trained end-to-end agents racing under evalution conditions on the Porto track, with sampling rates ranging from 3 Hz to 50 Hz.

The optimal value for the batch size B was determined by training and evaluating agents with batch sizes of 50, 100, 150, 200, 400, 600 and 1000 samples. Three agents were trained for every batch size, while holding the remaining hyper-parameters constant at the values listed in Table 5.1. The average lap time and percentage of failed laps of agents under evaluation conditions are shown as a function of batch size in Figure 5.7. From this figure, we observe that lap time and failed laps under evaluation conditions are minimised when the batch size is set to 400. Based on these results, we selected a batch size of 400 samples for our agents.

An experimental analysis was then conducted to select the reward discount rate, denoted γ. To determine the value for γ, we assessed the performance of agents with reward

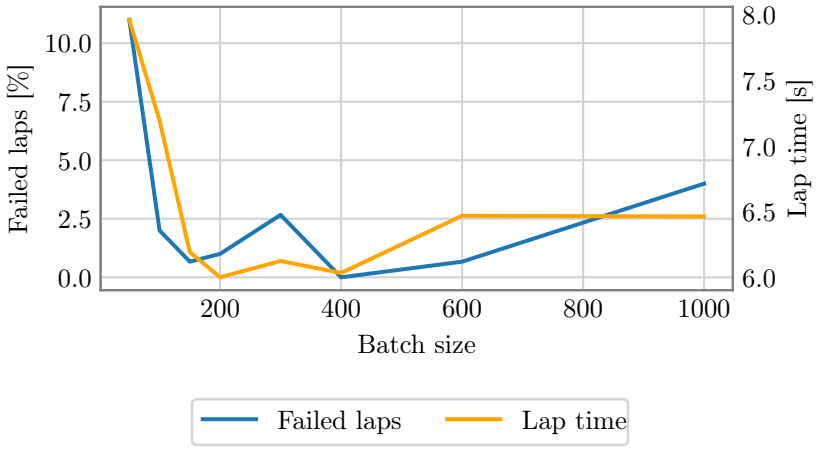


Figure 5.7: Percentage failed laps and lap time under evaluation conditions of end-to-end agents with batch sizes from 50 to 1000. The percentage failed laps is mapped onto the left vertical axis, while the lap time is mapped onto the right vertical axis.

discount rates of 0.95, 0.98, 0.99 and 1 during training. For each of these reward discount rate values, three agents were trained with their remaining hyper-parameters set equal to those listed in Table 5.1. The percentage failed laps and lap times during training, as well as the learning curves for these agents are shown in Figure 5.8.

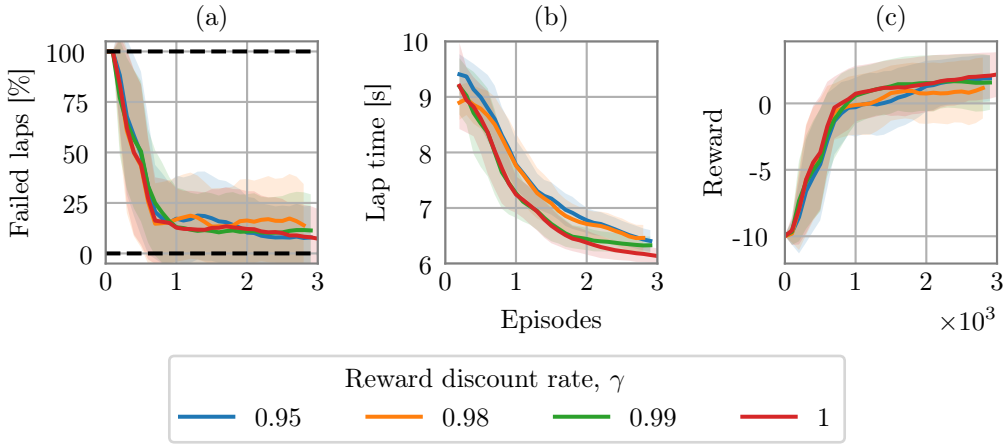


Figure 5.8: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of three end-to-end agents with reward discount rates ranging from 0.9 to 1.

When viewing the performance during training, these agents appear to be perform similarly. However, the TD3 algorithm has no mechanism for decreasing the exploration noise added to every action with training time. Figure 5.8 is therefore an indicator of the performance of each agent with exploration noise added to every action. As such, we also considered the performance of each agent under evaluation conditions where no exploration noise is present. The percentage failed laps and lap times for agents trained with each learning rate is shown in Table 5.2. The table show that a discount rate of 0.99 yields agents that successfully complete all of their laps. Based on this finding, a discount rate of 0.99 was selected.

Reward discount rate (γ)	Failed laps [%]	Average lap time [s]	Standard deviation of lap time [s]
0.95	2.33	6.12	0.28
0.98	0.33	6.51	0.28
0.99	0.00	6.07	0.20
1	0.33	5.94	0.11

Table 5.2: Percentage failed laps and average lap times under evaluation conditions for end-to-end agents trained with reward discount rates ranging from 0.9 to 1.

Values for the hyper-parameters τ, σ_{action}, and d were determined by repeating the tuning procedure used for γ. That is, one hyper-parameter was varied while holding the others constant. For each hyper-parameter set, three agents were trained, and the selected was based on the agents' average performance during training and evaluation. For conciseness, the experimental results for these hyper-parameters are presented in Appendix A. Values of $5 \cdot 10^{-3}$ for τ, 0.1 for σ_{action}, and 2 for d yielded agents with the best performance.

After determining locally optimal hyper-parameters for TD3, we compared the performance of our implementation of the TD3 algorithm to a standard implementation of the popular Deep Deterministic Policy Gradient (DDPG) algorithm [15; 17; 31]. The percentage failed laps, lap time and learning curves of agents trained using both algorithms are depicted in Figure 5.9. The results reveal that TD3 outperforms DDPG by a substantial margin in terms of both crashes and lap time. Moreover, we have observed that the training stability of TD3 is superior to that of DDPG, as evidenced by the smoother learning curve of TD3 in contrast to the more erratic curve of DDPG.

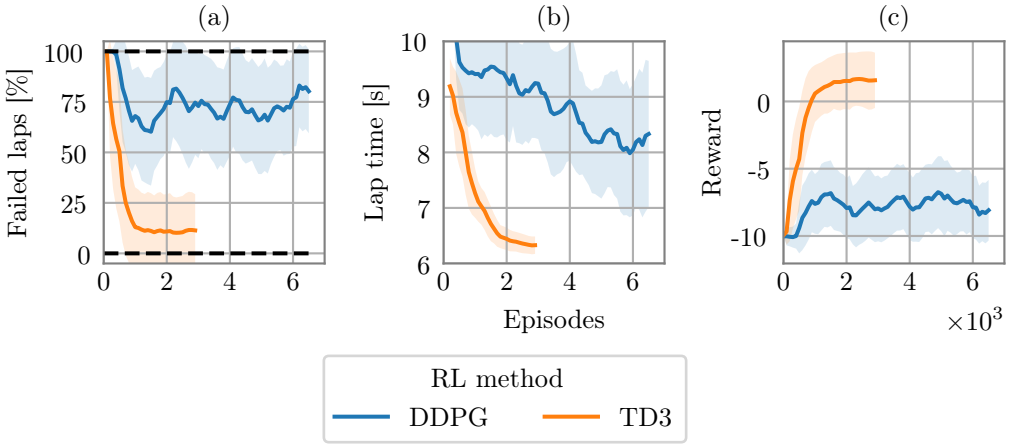


Figure 5.9: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of three end-to-end agents that were trained using TD3 and DDPG.

5.5 Reward signal

Having experimentally determined locally optimal values for the TD3 hyper-parameters, we investigated the reward signal. Our objectives were to choose reward signal parameter values that yielded agents that (a) race safety while (b) minimising lap time. This was a challenging task, considering that these two objectives are in conflict with each other. Further complicating the task is that the lap time alone is too sparse a signal to allow the agent to learn effectively [29; 28]. The reward signal from Equation 5.3 was therefore designed to approximate a signal that minimises lap time, while also providing continuous rewards to the agent. Specifically, the reward signal described in Equation 5.3 rewards the agent for the distance it travelled along the centerline between the current and previous time step, and penalises the agent a small amount on every time step [30]. Additionally, the agent receives a large penalty for colliding with the track boundary.

To motivate the use of a time step penalty r_{time} and to prove that our reward signal approximates a minimisation of lap time, we examine the total reward accumulated over a successful episode,

$$R_{\text{total}} = \sum_{t=1}^{T} \left(r_{\text{dist}}(D_t - D_{t-1}) + r_{\text{time}} \right), \tag{5.4}$$

which is the quantity that the agent learns to maximize when no reward discounting is assumed. In this equation, the subscript t indicates a time step, T is the final time step of the episode, and D_t is the distance travelled along the centerline at time t. Expanding the summation from Equation 5.4 yields

$$\begin{aligned} R_{\text{total}} &= r_{\text{dist}} \left((D_1 - 0) + \ldots + (D_T - D_{T-1}) \right) + \sum_{t=1}^{T} r_{\text{time}} \\ &= r_{\text{dist}} D_T + T r_{\text{time}}. \end{aligned} \tag{5.5}$$

To simplify the expression for total reward, r_{time} was set equal to $-\Delta t$, or -0.01. Additionally, T was substituted as

$$T = \frac{\text{lap time}}{\Delta t}. \tag{5.6}$$

By substituting Equation 5.6 into Equation 5.5, we get R_{total} as

$$R_{\text{total}} = r_{\text{dist}} D_T - \text{lap time}. \tag{5.7}$$

From this equation, we can see that in order to maximise the cumulative reward, the agent must minimise lap time. This is because D_t and r_{dist} are constants. Therefore, the reward signal from Equation 5.3 approximates a signal that minimises lap time for sufficiently large reward discount factors. Interestingly, if no time step penalty is applied, then every successful lap yields the same reward regardless of lap time.

To experimintally confirm the result from Equation 5.7, we trained and evaluated three agents with r_{time} set to -0.01, then repeated the training procedure for three agents without the time step penalty. For each of these agents, the remaining reward signal components and hyper-parameters were set equal to the values listed in Table 5.1. Setting r_{time} to -0.01 improves the average evaluation lap time of agents from 9.26 seconds to 6.07 seconds, compared to agents that did not receive the penalty. Furthermore, we tuned the other reward signal terms are relative to the r_{time} value of -0.01.

We now present the tuning procedure for the distance reward r_{dist}, as well as the collision penalty $r_{\text{collision}}$. We initially determined a plausible range of r_{dist} values to train

our agents with. Intuitively, a lower bound for r_{dist} exists that results in a policy that completes laps. If r_{dist} is set beneath this lower bound, the agent can only accumulate negative reward by continuing to race, and the optimal action is to crash immediately. We estimated this lower bound by considering that the agent should be able to achieve positive reward at every time step, such that

$$r_{\text{dist}}(D_t - D_{t-1}) + r_{\text{time}} > 0. \tag{5.8}$$

Solving the inequality in Equation 5.8 for r_{dist} gives

$$r_{\text{dist}} > \frac{-r_{\text{time}}}{(D_t - D_{t-1})}, \tag{5.9}$$

where D_t and D_{t-1} are unknown. To obtain the smallest value for r_{dist}, the largest value possible for $(D_t - D_{t-1})$ is estimated by considering a case whereby the vehicle travels at maximum speed parallel to the centerline, such that

$$(D_t - D_{t-1}) = v_{\text{max}} \Delta t. \tag{5.10}$$

After substituting the expression from Equation 5.10 into Equation 5.9 and setting r_{time} equal to $-\Delta t$, the minimum value for r_{distance} is found to be

$$r_{\text{dist}} > \frac{1}{v_{\text{max}}}. \tag{5.11}$$

Substituting the value for v_{max} as 5 m/s (see Table 5.1, as well as Section 5.8) into Equation 5.11 yields an estimated minimum r_{dist} of 0.2.

Using this value as a guide for the region in which to search for r_{dist}, we trained agents with r_{dist} values of 0.1, 0.25, 0.3 and 1. For each r_{dist} value, three agents were trained with their remaining hyper-parameters equal to those listed in Table 5.1. The percentage failed laps and average lap time of completed laps during training for these agents are shown in Figure 5.10. Unsurprisingly, the agent with r_{dist} set to 0.1 (i.e., less than the estimated minimum) learns that terminating the episode immediately is the optimal behaviour, as its failure rate remains at 100 percent.

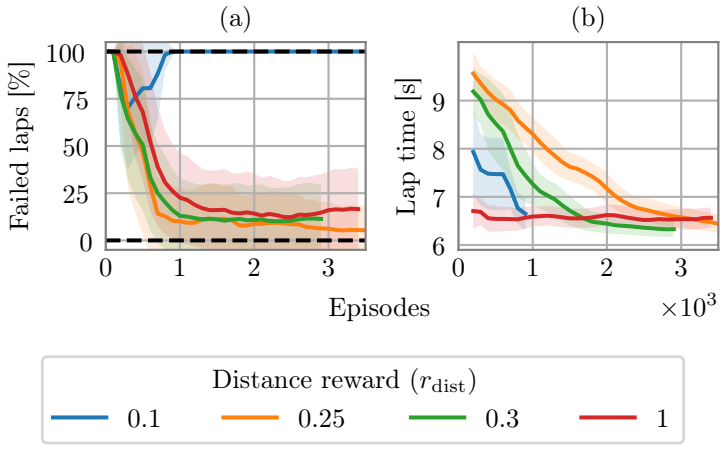


Figure 5.10: (a) The percentage failed laps and (b) lap times of completed laps during training of end-to-end agents with r_{dist} values ranging from 0.1 to 1.

Figure 5.10 also reveals that larger values of r_{dist} result in worse performance in terms of failed laps and lap time. When r_{dist} is set to a larger value, the time step penalty becomes less significant. As such, the agent is less incentivised to minimise lap time. Conversely, when r_{dist} is set close to the estimated minimum value, the time step penalty becomes significant, and the agent must optimise lap time to receive positive rewards. The value for r_{dist} was chosen as 0.25, as agents that were trained with this value had the lowest crash rate while also achieving competitive lap times.

After setting the value for r_{dist}, the penalty imposed on the agent when it collides with the track boundary was fine-tuned. Initially, we investigated whether the agent could acquire the racing skills without facing any penalties for collisions. However, agents trained with such a reward signal crashed on 4% of their laps during evaluation. Consequently, further experiments were conducted that considered negative $r_{\text{collision}}$ values.

To identify a suitable range within which we could conduct experimental searches for an optimal value, we operated on the premise that $r_{\text{collision}}$ should be substantial compared to the positive reward an agent can receive in an episode. As shown in Figure 5.4, agents attain an average reward value of 2 in episodes where crashes do not occur. Consequently, we trained agents with collision penalties ranging from -2 to -10. As before, three agents with hyper-parameter set equal to those listed in Table 5.1 were trained for each $r_{\text{collision}}$ value. The percentage failed laps and average lap times under evaluation conditions for agents trained with these values for $r_{\text{collision}}$ are presented in Table 5.3. We selected $r_{\text{collision}}$ as -10, as it is the only penalty that results in no failed laps.

$\mathbf{r_{collision}}$	**Failed laps [%]**	**Average lap time [s]**	**Standard deviation of lap time [s]**
0	4.00	5.69	0.16
-2	1.33	5.63	0.17
-4	1.00	5.69	0.17
-8	1.33	6.11	0.47
-10	0.00	6.07	0.20

Table 5.3: Percentage failed laps and lap times under evaluation conditions for agents trained with $r_{\text{collision}}$ values from ranging from 0 to -10.

Interestingly, the effect of increasing the collision penalty can be seen in the path taken by agent. Figure 5.11 shows the paths taken by agents with $r_{\text{collision}}$ set to -4 and -10. The agent with the lower collision penalty races close to the edge of the track, while the agent that is penalised more heavily takes a much more conservative path by staying clear of the track boundaries, instead preferring to drive near the centerline of the track.

5.6 Observation space

We also conducted investigations to determine which combiniation of elements in the observation space vector resulted in optimal performance. This was done by training and evaluating agents that received (a) only the pose, (b) only a LiDAR scan and (c) a combination of vehicle pose and LiDAR scan in their observation. For each of these observation space combinations, three agents with hyper-parameters listed in Table 5.1 were trained. The performance of these agents during training, in terms of percentage failed laps, average lap time and average reward, is shown in Figure 5.12.

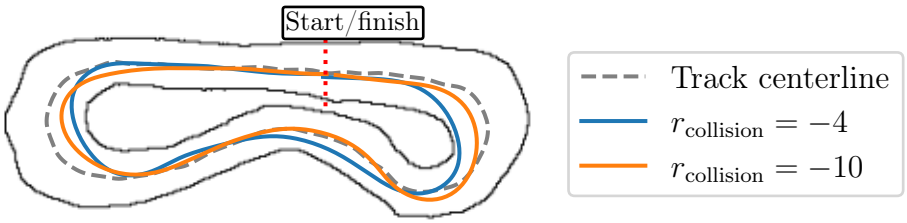


Figure 5.11: The paths taken by agents trained with $r_{\text{collision}}$ values of -4 and -10.

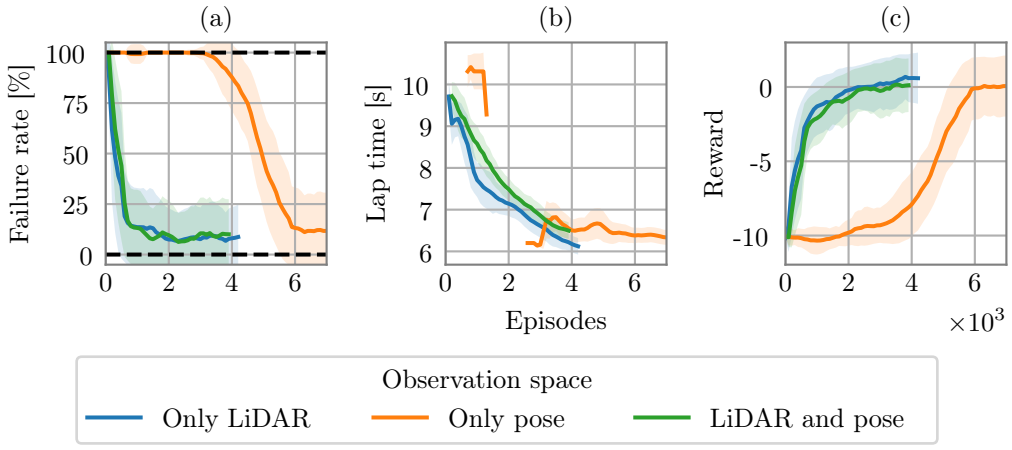


Figure 5.12: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves showing episode reward for end-to-end agents with different observation spaces.

From this figure, agents utilising each of the observation spaces converge to a similar values for all three evaluation metrics. However, agents that receive a LiDAR scan train significantly faster than agents without a LiDAR scan in their observation. Specifically, the LiDAR scan allows the agent to learn to avoid track boundaries without needing to sample collision experiences at every point along the track boundary. This is clearly demonstrated in Figure 5.13, which shows all of the locations where an agent observing only the pose, and an agent observing both pose and LiDAR scan crashed during training. Agents without LiDAR scans crashed 5183 times, whereas agents observing LiDAR scans crashed only 464 times during the same training period.

After determining that including the LiDAR scan in the observation improves training performance, we assessed agents utilising each observation space under evaluation conditions. The agent that utilised both the LiDAR scan and pose in the observation did not crash during evaluation, whereas agents with either only a LiDAR scan or pose failed to complete laps 0.67% and 6.00% of the time, respectively. Based on these results, both a LiDAR scan and the vehicle pose were included into the observation.

Given that a LiDAR scan is included in the observation, another parameter to consider

(a) Only pose **(b)** Pose and LiDAR

Figure 5.13: Crash locations of agents with (a) only the pose and (b) both the pose and LiDAR scan during training.

is the number of LiDAR beams to include. To determine the number of LiDAR beam that results in optimal performance, agents with LiDAR scans consisting of 5, 10, 20 and 50 were trained and tested. These beams are equally spaced, and have a field of view of 180°. As before, three agents with hyper-parameters from Table 5.1 were trained for every value of L LiDAR beams. Figure 5.14 displays the percentage failed laps and lap times during training, as well as the learning curves of these agents. The results indicate that increasing the number of LiDAR beams above 20 does not significantly impact the performance of agents in terms of any of the measured criteria. We therefore chose to incorporate 20 LiDAR into the observation space.

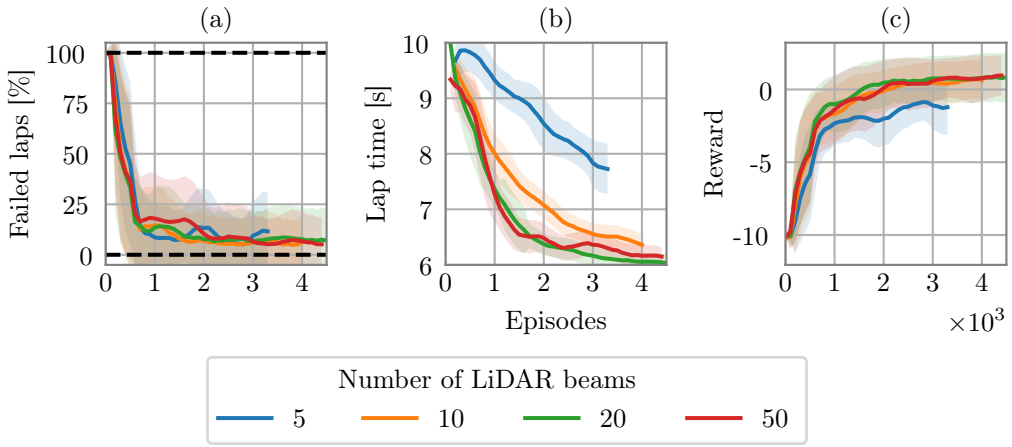


Figure 5.14: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves showing episode reward for end-to-end agents with different numbers of LiDAR beams during training.

The simulation environment described in Section 4.3 allows for the addition of noise to the observation vector. The tests conducted thus far have not included noise in the agents' observations. However, noise is added to the observation vector to increase the realism of the simulation when racing under evaluation conditions. It is therefore important to determine whether adding noise to the observation elements during training benefits the performance of the agent under evaluation conditions. Specifically, we trained three agents without any noise in the observation vector, and another three with added Gaussian noise which had standard deviations of 0.025 m for x and y coordinates, 0.05 rads for heading, 0.1 m/s for velocity, and 0.01 m for LiDAR scan.

The agents trained with noise achieved an average lap time of 6.77 seconds while completing 98.67% of the laps under evaluation conditions. In comparison, agents trained without noise completed all laps with an average time of 6.09 seconds. It is noteworthy that the agents trained with observation noise completed laps in a more erratic manner than agents trained without noise. Examples of paths completed by agents trained with and without noise are shown in Figure 5.15. This was despite the presence of noise under evaluation conditions. We therefore chose to train agents without observation noise.

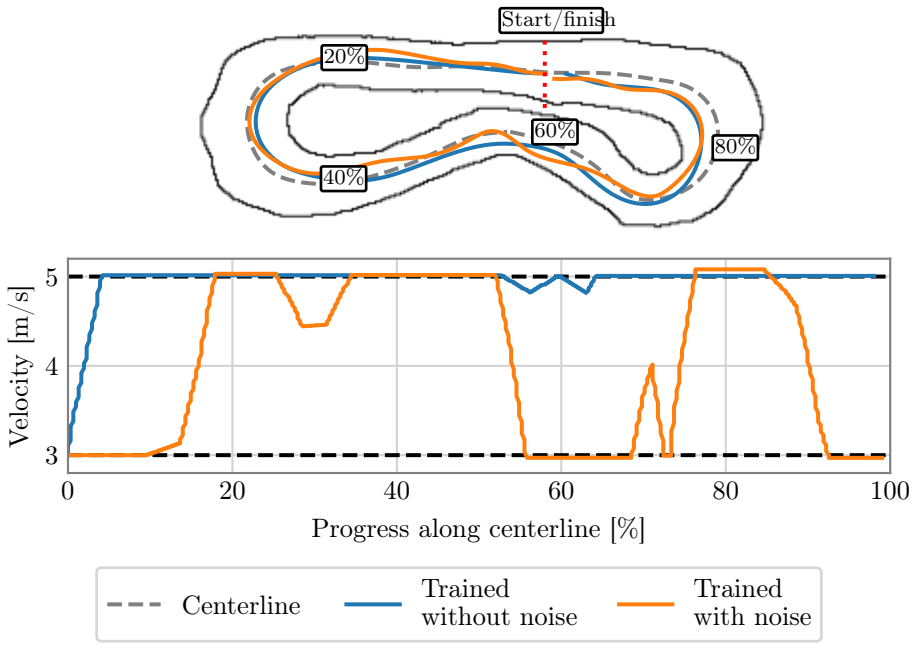


Figure 5.15: Path and velocity profiles of end-to-end agents that were trained with and without noise added to the observation vector.

5.7 Neural network hyper-parameters

Next, an investigation was conducted to determine the optimal DNN layer configuration for the actor and critics. In this experiment, the layer configuration of the actor and both critics were varied together, such that the input and hidden layers for all three DNNs remained identical in structure. The input and hidden layers of these DNNs were initially specified to be 400 and 300 units, respectively. Three agents were then trained with input and hidden layers that were 100 units larger and smaller than the initial DNN configuration. The remaining hyper-parameters of these agents were set equal to those listed in Table 5.1. The percentage failed laps, lap times, and learning curves while training these agents are depicted in Figure 5.16.

The experimental results from Figure 5.16 indicates that increasing or decreasing the number of units in the input and hidden layers leads to a deterioration in performance, particularly in terms of lap time. As a result, the input layer size was selected as 400 units, and the hidden layer size was selected as 300 units for both the actor and critic DNNs.

Additional experiments were conducted to determine the optimal learning rate α. The same value for α was used for the actor and critic DNNs. During this experiment, we

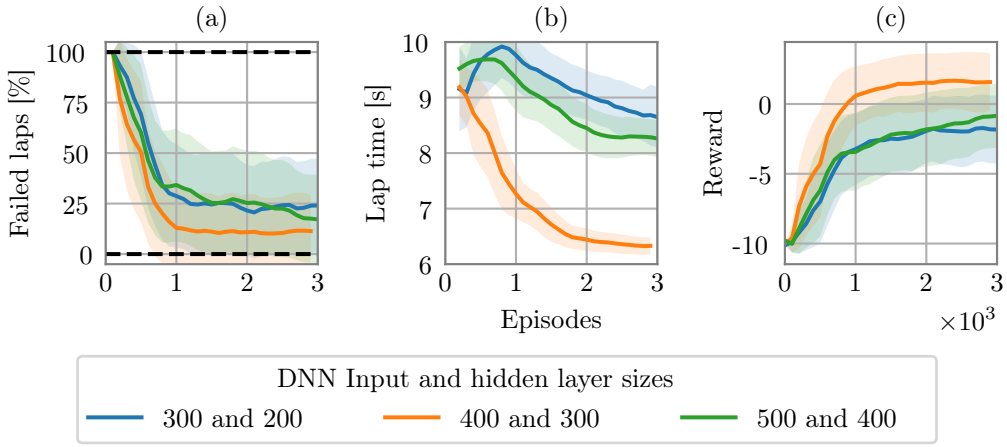


Figure 5.16: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves showing episode reward for end-to-end agents with different DNN layer sizes.

trained three agents with learning rates of 10^{-4}, 10^{-3} and $2 \cdot 10^{-3}$, and remaining hyper-parameters set equal to Table 5.1. The performance of these agents under evaluation conditions is shown in Table 5.4. The percentage successful laps and lap time of agents were maximised and minimised, respectively, when the learning rate was set to 10^{-3}.

Learning rate (for actor and both critics), α	**Successful evaluation laps [%]**	**Average evaluation lap time [s]**	**Standard deviation of test lap time [s]**
$1 \cdot 10^{-4}$	100	6.09	0.17
$1 \cdot 10^{-3}$	100	6.07	0.20
$2 \cdot 10^{-3}$	98.67	7.29	0.53

Table 5.4: Evaluation results of end-to-end agents with actor and critic DNN learning rates between $1 \cdot 10^{-4}$ and $2 \cdot 10^{-3}$.

5.8 Velocity constraint

In our final hyper-parameter tuning investigation, we conduct experiments to determine the minimum and maximum allowable velocities. Limiting the velocity is a common technique to ensure safe operation of the vehicle. For example, Ivanov et al. [15] restrict the torque applied to the vehicle's driving motors, thereby limiting its maximum speed to 2.4 m/s. Hsu et al. [16] adopt less conservative bounds, enforcing minimum and maximum speed limits of 1.125 and 9.3 m/s, respectively.

To determine the maximum safe velocity, we trained and evaluated the behaviour of agents with $v_{\max}$ values of 5, 6, 7 and 8 m/s. Figure 5.17 illustrates the velocity and slip angle profiles of the agents as they complete one lap under evaluation conditions. Interestingly, agents tend to maintain the maximum velocity around the track. This behaviour likely occurs because even small values of $a_{\text{long},d}$ result in large changes in

velocity in-between agent samples. Further exacerbating this effect is the slow rate at which actions can be sampled from the agent.

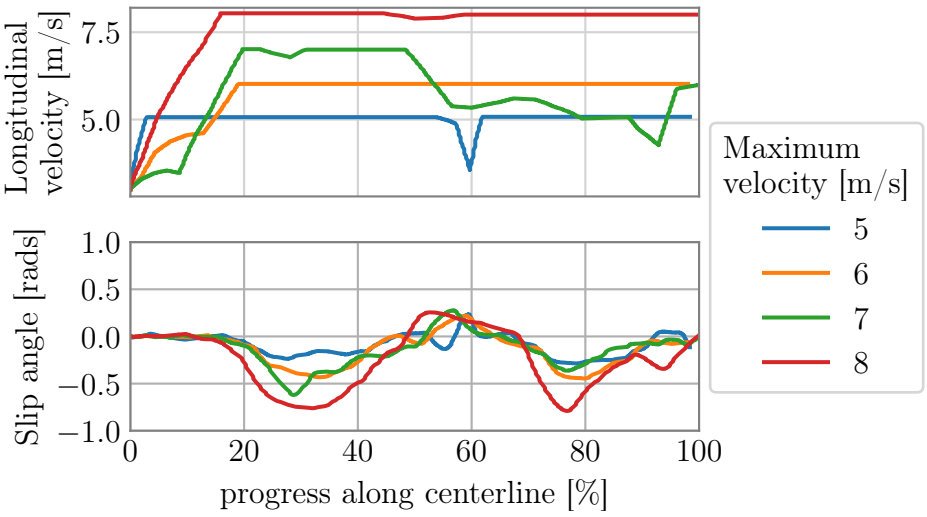


Figure 5.17: The velocity profile and slip angle of agents with different maximum velocities during one test lap.

Figure 5.17 shows that agents with a maximum velocity greater than 5 m/s experience slip angles larger than 0.2 radians, which is considered both dangerous and unrealistic drifting behavior. Furthermore, the dynamic bicycle model from Chapter 4 makes assumption that tire stiffness varies linearly with lateral force. This assumption is only valid for slip angles below 0.2 radians [80]. Allowing the agent to select large velocities enables it to exploit the simulation in an unrealistic manner to achieve fast lap times. Therefore, the vehicle's maximum speed was set to 5 m/s, which was the fastest velocity that did not result in end-to-end agents that drive dangerously and exploite the simulator by operating the car at large slip angles.

When there was no minimum velocity constraint in place, the agent would often choose to bring the car to a standstill during training, resulting in excessively long training times. The minimum speed was therefore set to to 3 m/s to prevent this behaviour. Importantly, this constraint did not significantly affect the agent's performance.

5.9 End-to-end racing without model uncertainty

Having determined a set of hyper-parameters that yield optimal performance in terms of both safety and lap time for the end-to-end agent racing on the Porto track, we trained and evaluated ten agents with these hyper-parameters. These agents completed 98.9% of evaluation laps with an average lap time of 6.05 seconds, achieving better performance than any other hyper-parameter set that was tested. In fact, varying any of the hyper-parameters resulted in decreased performance, showing that the selected hyper-parameter values are at least locally optimal. Figure 5.18 provides a visualization of one of these agents' laps, highlighting the path taken with a color map representing the agent's velocity. Notably, the agent maintained maximum velocity for the majority of the track length.

Nevertheless, the trajectory is smooth and the agent successfully navigated around the circuit.

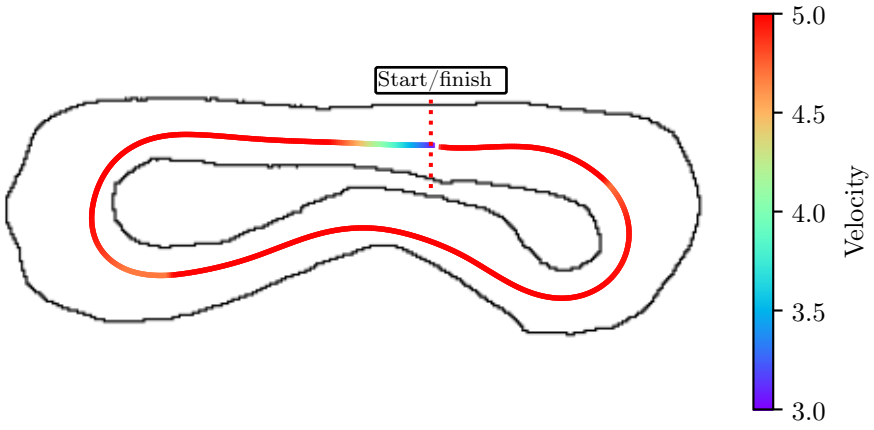


Figure 5.18: The path and velocity profile taken by an end-to-end agent completing Porto in the anti-clockwise direction.

So far, the focus has been on the relatively simple Porto track. However, the analysis was expanded to encompass more realistic racing scenarios by training agents to navigate scaled versions of actual Formula 1 tracks. Specifically, Circuit de Barcelona-Catalunya in Spain and the Circuit de Monaco in Monaco were selected. These tracks are not only considerably larger, but also feature sharper corners and more complex geometries compared to the Porto track.

When selecting hyper-parameters for the larger tracks, a tuning procedure similar to the one presented for the Porto track was utilised. That is, the hyper-parameter were systematically varied one at a time, while keeping the other hyper-parameters constant. This hyper-parameter tuning procedure resulted in the following adjustments to Table 5.1 for agents racing on these longer tracks: the number of MDP time steps (M) was increased to $2.5 \cdot 10^5$, agent sample rate (f_{agent}) was increased to 10 Hz, and the reward signal values for r_{dist} and $r_{\text{collision}}$ were changed to 0.3 and -2, respectively. The average learning curves for 10 agents trained to race on each of the tracks using the given hyper-parameters are shown in Figure 5.19. Importantly, we observe that these agents maximise reward on each of their respective tracks.

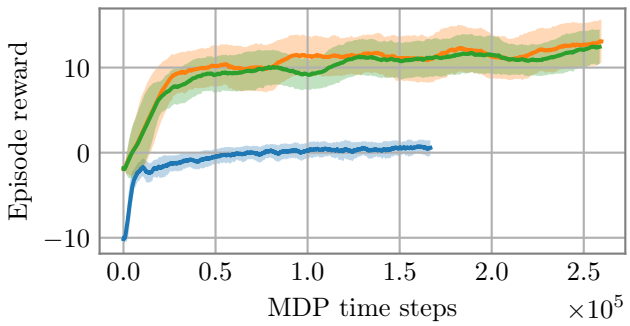


Figure 5.19: Learning curves for end-to-end agents trained on Porto, Circuit de Barcelona-Catalunya and Circuit de Monaco.

Agents trained to race on Circuit de Barcelona-Catalunya completed their laps 56.30% of the time, and achieved an average lap time of 47.39 seconds under evaluation conditions. Figure 5.20 shows the path and velocity profile taken by an agent completing Circuit de Barcelona-Catalunya under evaluation conditions. Similar to the findings on the Porto track, agents racing on the Circuit de Barcelona-Catalunya selected maximum velocity for the majority of the track, even when navigating sharp corners. Furthermore, an interesting phenomenon emerged on the Circuit de Barcelona-Catalunya that was not present on the shorter Porto track: agents tend to exhibit a slaloming behavior, which is characterized by a winding path. This slaloming effect is quite severe, occurring at nearly every section of the track.

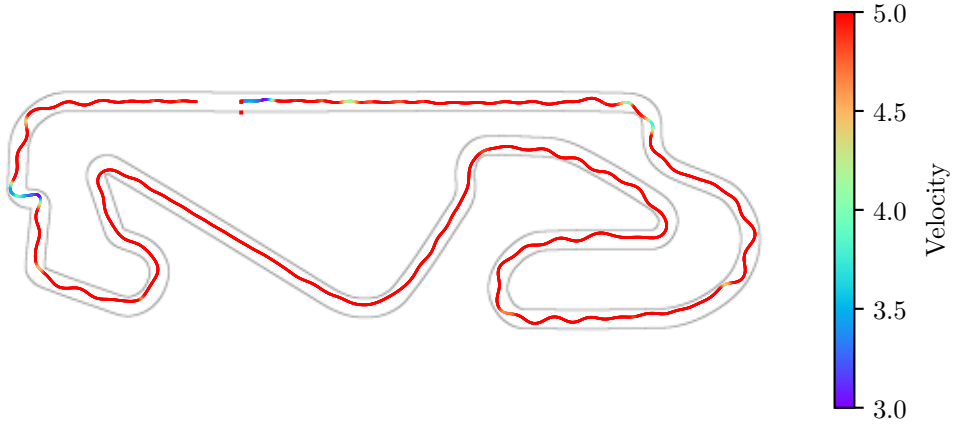


Figure 5.20: The path and velocity profile taken by an end-to-end agent completing Circuit de Barcelona-Catalunya

When assessing agents trained to race on Circuit de Monaco, we found that agents successfully completed their laps on 56.20% of their attempts, achieving an average lap time of 47.39 seconds. Figure 5.20 depicts one example of the path and velocity profile taken by an agent taht successfully completed the Circuit de Monaco under evaluation conditions. Interestingly, the slaloming is also present on the Circuit de Monaco, indicating that slaloming tends to be a common issue for end-to-end agents navigating long tracks. Slaloming can negatively impact the performance of an agent under model-mismatch conditions.

5.10 End-to-end racing with model uncertainty

Up to now, results have been presented for end-to-end agents that were trained and evaluated in identical environments. However, it is important to assess the performance of agents tasked with driving in situations where the vehicle model does not match the one utilised during training. During this initial investigation, we introduced model mismatches by modifying the vehicle model parameters after training, but prior to executing the evaluation process outlined in Algorithm 6. This adjustment allows us to gain insights into how the agent performs in a more realistic setting where variations in the vehicle model are present.

Our initial focus was on investigating the impact of altering the road surface friction coefficient on the evaluation performance of trained agents. Friction is influenced by

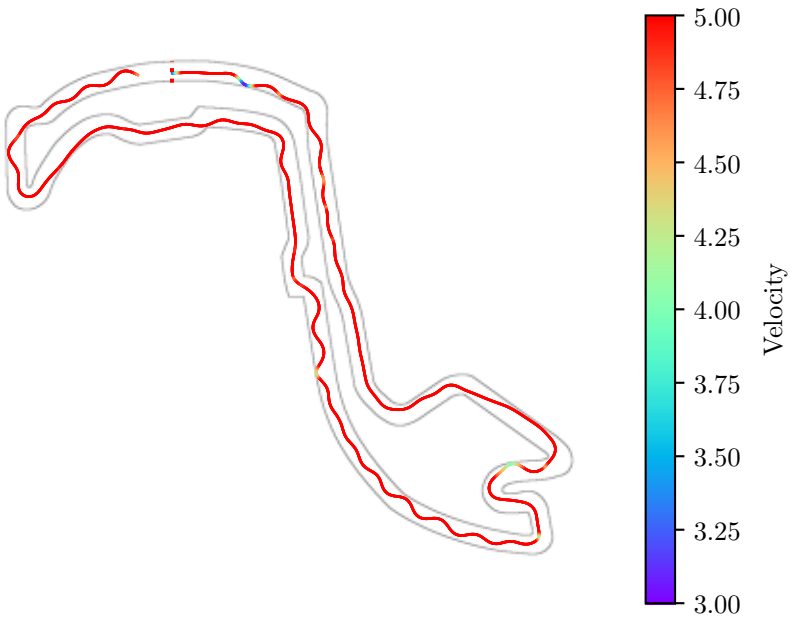
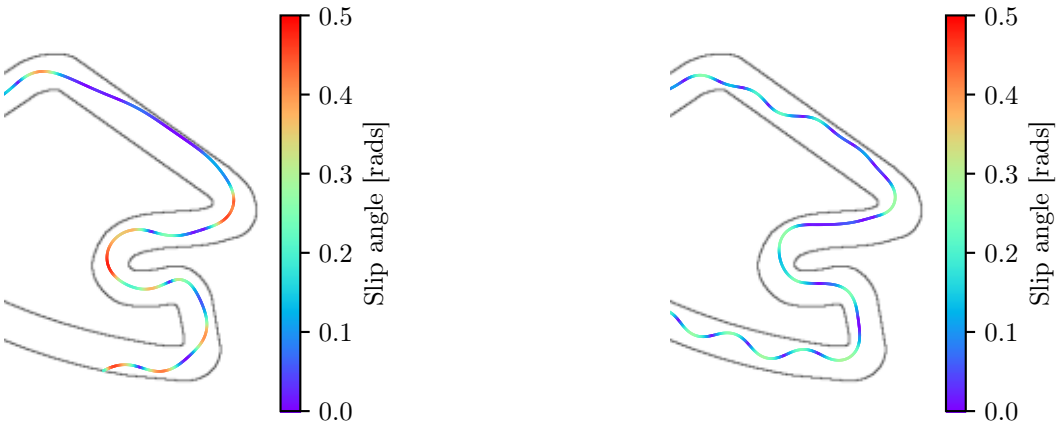


Figure 5.21: The path and velocity profile taken by an end-to-end agent completing Circuit de Monaco

various dynamic factors, including temperature and precipitation, making it challenging to predict accurately. Consequently, it is likely that model mismatches in the road friction coefficient occur. Figure 5.22 presents a comparison of paths taken by agents evaluated with (a) the nominal friction value of 1.04, and (b) a friction value of 0.6 (equivalent to wet asphalt conditions) on a section of the Monaco track. The slip angles of the agents are visualized by color-mapping them onto their respective paths. When evaluated with the nominal friction value, the agents display slaloming behavior, resulting in maximum slip angles of approximately 0.2 radians throughout most areas of the track. In contrast, agents evaluated with decreased friction exhibit drifting behavior, characterized by slip angles



(a) **(b)**

Figure 5.22: Trajectory and slip angle of an end-to-end agent racing on Circuit de Monaco with (a) the nominal road-surface friction value of 1.03, and (b) a decreased road-surface friction value of 0.6.

exceeding 0.4 radians. The drastic increase in slip angle indicates that the learned policy of standard end-to-end agents is dangerous under conditions where model mismatches are present.

Notably, Figure 5.22 illustrates an instance where an agent with decreased friction crashes shortly after executing a drift maneuver. This observation emphasizes the impact of friction on the agents' handling capabilities and reinforces the significance of creating algorithms that are robust to errors in the vehicle model parameters.

5.11 Training with domain randomisation

A commonly used technique to enhance robustness against modelling errors is domain randomisation, which involves randomising simulation parameters during training. The agent is then tasked with finding a single policy that performs optimally across different parameter settings [12]. Previous autonomous racing studies have explored various approaches in this regard. Chisari et al. [33] introduced Gaussian noise to the lateral force experienced by the tires at each time step, while Ghignone et al. [14] initialized each episode by adding Gaussian noise with a standard deviation of 0.0375 to the road surface friction coefficient, which remained constant throughout the episode.

In this investigation, we adopted the approach of Ghignone et al. [14], and modified the training procedure by sampling the friction value used during every episode from a Gaussian distribution. This Gaussian distribution had a mean of 1.0489 (the nominal friction value). Two agents were trained to race on the Porto track; one with a friction coefficient standard deviation of 0.01 and another with a standard deviation of 0.05. These agents were then tasked with completing 100 laps under evaluation conditions, with the mean value of 1.0489 used in every episode.

While agents trained with a friction coefficient standard deviation of 0.01 successfully completed 51% of their laps, agents trained with a standard deviation of 0.05 completed only 34% of their laps under evaluation conditions. These results indicate that domain randomisation has an adverse effect on the agents' performance, even when the agent is only tasked with racing under conditions where the average friction value is present. Figure 5.23 illustrates the paths taken by these agents during evaluation, and clearly indicates their inability to learn smooth driving behavior.

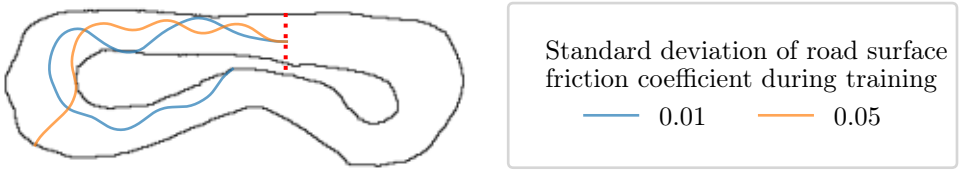


Figure 5.23: Paths taken by agents trained with randomised road-surface friction coefficients on the Porto track under evaluation conditions. During this evaluation lap, the friction coefficient was set to the nominal value of 1.0489.

Our findings suggest that the optimal policy for autonomous racing is highly sensitive to the friction coefficient of the road surface. Agents struggle to adapt their policies to changing friction values effectively, resulting in poorer performance. This sensitivity highlights the challenge of developing a single policy that performs optimally across a

range of friction coefficients, demonstrating the limitations of domain randomisation in the racing context.

5.12 Summary

In this chapter, we have motivated the design of an end-to-end autonomous racing algorithm. Agents utilising this algorithm were trained to race effectively on the Porto track, successfully completing all of their laps under evaluation conditions. However, this performance did not scale to larger tracks such as Circuit de Barcelona-Catalunya or Circuit de Monaco. On these longer tracks, the performance of the agents was hindered by slaloming, and they did not complete all of their laps.

The presence of slaloming is particularly concerning when considering scenarios in which model mismatches are present. In fact, during a preliminary investigation into the effect of model mismatch on the performance of end-to-end agents, we observed collisions. This is indicative of the limitations of end-to-end algorithms under conditions where model mismatches are present, and emphasises the need for algorithms that exhibit robustness against modeling errors.

In the next chapter, we introduce our partial end-to-end solution, which aims to enhance robustness towards modeling errors and address the challenges posed by the sensitivity of the optimal policy to vehicle model parameters.

Chapter 6

Partial end-to-end autonomous racing

Having designed the baseline end-to-end agent and motivated the need for driving algorithms that are robust towards modelling errors, we now introduce our partial end-to-end algorithm. This approach separates the planning and control tasks, which enables a planner agent to generate a desired trajectory, which is then tracked using a set of steering and velocity controllers. By decoupling the planning and control aspects, our algorithm aims to enhance robustness against modelling errors that commonly arise during the transfer from simulation to real-world environments. Furthmore, by planning a path relative to the track, partial end-to-end agents are embedded with environment knowledge, which results in a performance advantage during training and evaluation.

The chapter begins with a detailed description of the partial end-to-end algorithm. Subsequently, we outline the implementation of the TD3 algorithm to train the agent effectively. Next, the process of determining the optimal hyper-parameters for the partial end-to-end algorithm to ensure the system operates at its peak performance is shown. The performance of the partial end-to-end algorithm is then assessed in scenarios where no modeling errors are present, allowing us to gauge the algorithms performance under ideal conditions. Additionally, a comparative analysis is conducted, contrasting our chosen partial end-to-end architecture with alternative variations of the partial end-to-end approach. These variations include architectures with either solely a velocity controller or a steering controller.

6.1 Partial end-to-end racing algorithm

Our partial end-to-end algorithm has the decoupled the structure of classical algorithms, and is comprised of a planner RL agent, a steering and a velocity controller, as well as a velocity constraint, as depicted in Figure 6.1.

Given that the simulator provides a LiDAR scan and the vehicle's pose, the need for a perception algorithm to map the environment and localize the vehicle is eliminated. The output from the simulator is therefore passed directly to the agent. Applying the findings from Section 5.6, the observation space of the partial end-to-end agents consist of a LiDAR scan with 20 beams and vehicle pose. The agent maps this observation to an action space which comprises a path (represented by a series of x and y coordinates), and desired velocity (denoted as v_d) at a rate of f_{agent} Hz.

A pure pursuit steering controller is used to generate desired steering commands (denoted as δ_d) that track the path. Meanwhile, a proportional feedback velocity controller generates desired acceleration commands, denoted $a_{\text{long},d}$, to ensure the vehicle main-

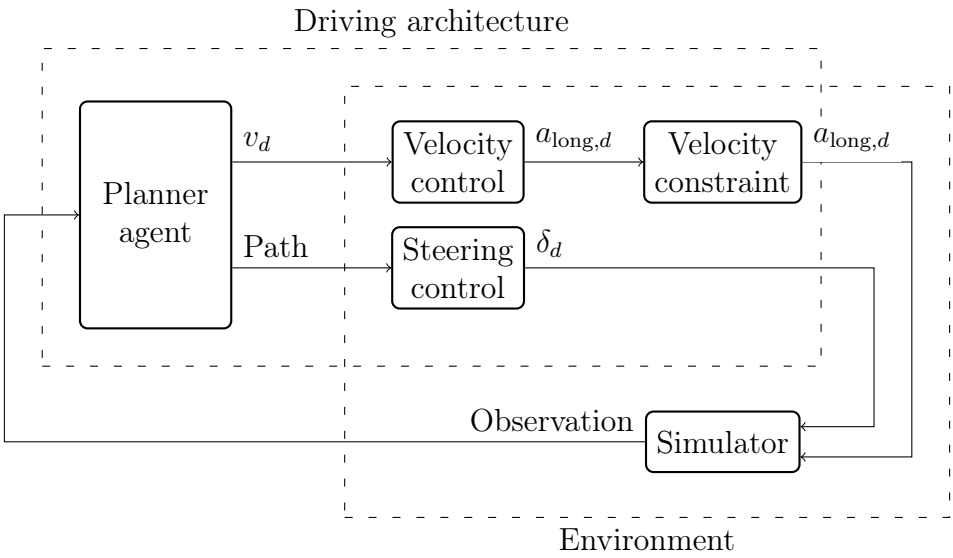


Figure 6.1: The partial end-to-end racing algorithm, which consists of an RL agent that outputs a plan comprised of a path (i.e, a series of x and y coordinates) and desired velocity, a steering and a velocity controller, as well as a velocity constraint. δ_d, v_d, and $a_{\text{long},d}$ denote the desired steering angle, longitudinal velocity and longitudinal acceleration respectively.

tains the desired velocity. Furthermore, the velocity constraint component, as introduced in Equation 5.1, limits the desired acceleration so that the vehicle operates within safe speed limits. To keep the comparison between partial and fully end-to-end agents fair, the allowable velocity range for the partial end-to-end agent is set between 3 and 5 m/s. Furthermore, the steering and velocity controllers, as well as velocity constraint operate at the 100 Hz sample rate of the simulator introduced in Section 4.3. It is important to note that these components are treated as part of the environment to conform to the definition of the MDP.

6.1.1 Planner agent

The partial end-to-end planner agent, which is built using a DNN, is presented in Figure 6.2. Similar to the end-to-end agent, the observation vector is normalized within the range of $[0, 1]$. The DNN consists of three fully connected layers. The input layer has m_1 neurons, followed by a hidden layer with m_2 neurons. Finally, the output layer has 2 neurons. The ReLU activation function is applied to the first two layers, while the output layer is activated using a hyperbolic tangent function. The use of a hyperbolic tangent function ensures that the outputs of the neural network are normalized within the range of $(-1, 1)$. One output of the DNN, denoted as v_{norm}, is scaled to the desired longitudinal velocity range $(v_{\min}, v_{\max})$ to yield the desired longitudinal velocity v_d. The other output, denoted as p, is utilized to construct the path that the vehicle will follow.

6.1.2 Path generation method

Partial end-to-end approaches adopt different methods to generate a path based on the output of the neural network. One common approach is to employ a predefined function that takes the neural network's output as parameters. For instance, Weiss et al. [19] utilize bezier curves, where the control points of the curves correspond to the output of the neural network. Similarly, Capo et al. [17] predict the offset of a single point ahead

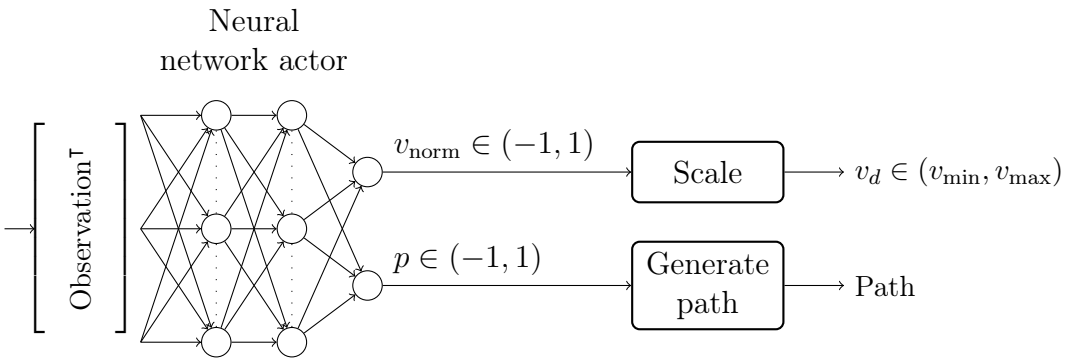


Figure 6.2: The partial end-to-end agent. The outputs of the DNN both have the range $(-1, 1)$. The output denoted v_{norm} corresponds to the desired longitudinal velocity and is scaled to the range $(v_{\min}, v_{\max})$. The other output, denoted p, is used to construct the path.

of the vehicle in relation to the track centerline using the neural network's output. On the other hand, classical approaches such as [81; 44; 82] utilize multiple motion primitives generated by forward simulating the vehicle dynamics to construct a path. However, these methods rely on direct access to the vehicle model, which is not available in our case as we are using model-free RL agents. Hence, we are limited to the former approach.

We chose to utilise the Frenet frame [45] to generate a path from the DNN output p. The Frenet frame is a curvilinear coordinate system where the horizontal axis is fixed to the centerline of the track. In this frame, distance along the horizontal axis corresponds to distance along the centerline, and is denoted as s. Additionally, the vertical axis represents the perpendicular distance from the centerline, and is denoted as n. We define the origin of the Frenet frame to coincide with the starting line.

To illustrate the Frenet frame, Figure 6.3 shows an example trajectory of an agent racing anti-clockwise around the Porto track. This figure presents the agent's trajectory in both Cartesian coordinates and Frenet coordinates. It is worth noting that within the Frenet frame, navigating around the track is equivalent to traveling along the horizontal

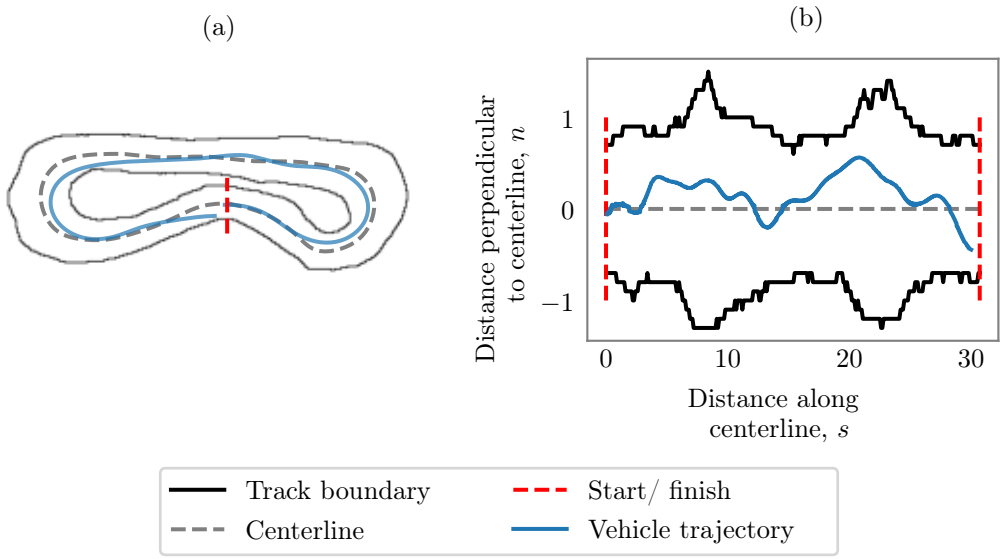


Figure 6.3: An example trajectory of an agent racing anti-clockwise around Porto, in (a) Cartesian coordinates, and (b) Frenet coordinates.

axis. Additionally, the track boundaries are conveniently expressed as vertical distances from the centerline. As a result, it is easier to create paths that avoid the track boundaries in Frenet coordinates compared to Cartesian coordinates. This advantage proves valuable because crashes can be prevented by constraining the action space to exclude trajectories that intersect with the track boundaries.

Our approach to generating paths involves solving a predefined third order polynomial function with specific constraints inside the Frenet coordinate system. Figure 6.4 illustrates this process, where the steps are as follows:

1. Convert the vehicle coordinates and heading into the Frenet frame, denoting the distance along the centerline as s_0 and the perpendicular distance from the centerline as n_0.

2. Determine the heading of the vehicle in the Frenet frame, denoted as ψ_0, by subtracting the heading of the path at the Cartesian coordinate corresponding to s_0 from the vehicle heading.

3. Construct a third-order polynomial within the Frenet frame, given by

$$f(s) = As^3 + Bs^2 + Cs + D, \tag{6.1}$$

   which is bounded horizontally by s_0 and s_1, where s_1 is chosen to be 2 meters ahead of s_0 along the centerline.

4. Apply the following constraints to the polynomial:

   a) The path must pass through the vehicle's center of gravity (CoG), satisfying $f(s_0) = n_0$.

   b) At s_0, the path is parallel to the vehicle's heading, which satisfies $f'(s_0) = \tan(\psi_0)$.

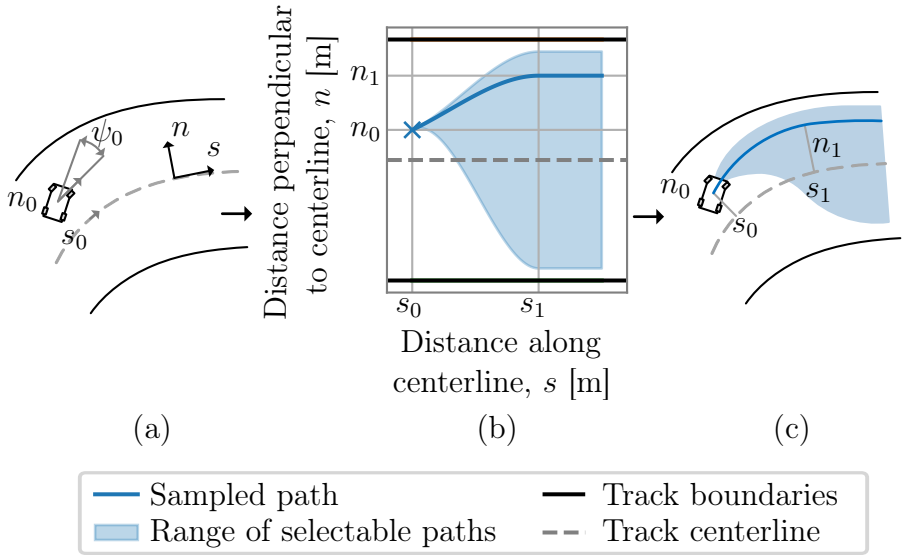


Figure 6.4: An illustration of the process of generating the polynomial path in the Frenet frame. (a) The vehicle coordinates are converted into the Frenet frame, then (b) a path is constructed within the Frenet frame, after which (c) the path is converted into Cartesian coordinates.

c) The perpendicular distance of the path from the centerline at s_1 is n_1, where n_1 is obtained by scaling the DNN output p by the track width. This is enforced by setting $f(s_1) = n_1$.

d) At s_1, the path is parallel to the centerline of the track, resulting in $f'(s_1) = 0$.

5. Extend the path with a horizontal line at (s_1, n_1) to prevent the vehicle from reaching the end of the path before sampling a new path.

6. Convert the path from Frenet frame coordinates to Cartesian coordinates for compatibility with the steering controller.

By scaling p to the width of the track, the paths that are generated do not intersect with the track boundary. In this way, agents are 'embedded' with knowledge about the track, which may enable them to train faster and with fewer collisions than model-free end-to-end agents that have no knowledge of the environment.

6.1.3 Steering controller design

The pure pursuit steering controller is popular amongst partial end-to-end methods [21; 18]. The popularity of this controller, combined with the fact that it does not require a vehicle dynamics model, guided our decision to implement it as the path tracker. Our implementation of pure pursuit is based on the work by Sakai et al. [78].

A pure pursuit controller facilitates the steering of the vehicle towards a designated *target point* on the planned path, as depicted in Figure 6.5. The target point is determined by a specified *look-ahead* distance, denoted as l_d, which is calculated using

$$l_d = k_s \cdot v + L_c, \tag{6.2}$$

where k_s is the look-ahead gain, L_c is a constant distance, and v is the longitudinal velocity of the vehicle in m/s. The look-ahead distance is adjusted according to the velocity based on the finding by Patnaik et al. [83] that larger look-ahead distances are required for higher velocities to maintain stability. Furthermore, l_d is measured from the center of the rear axle.

The desired steering angle δ_d is then computed as

$$\delta_d = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right), \tag{6.3}$$

where L is the wheelbase of the vehicle, and α is the angle between vehicle's heading and look-ahead distance vector. This ensures that the rear wheel travels in a circular arc to the target point, under the assumption that no slipping occurs. The target point and steering angle is recomputed using Equations 6.2 and 6.3 at a rate of 100 Hz.

6.1.4 Velocity controller design

The velocity controller is implemented in a similar manner to the proportional controller integrated into the official F1tenth simulator [6]. It takes the current vehicle velocity and the desired velocity v_d, determined by the planner agent, as inputs. It then calculates

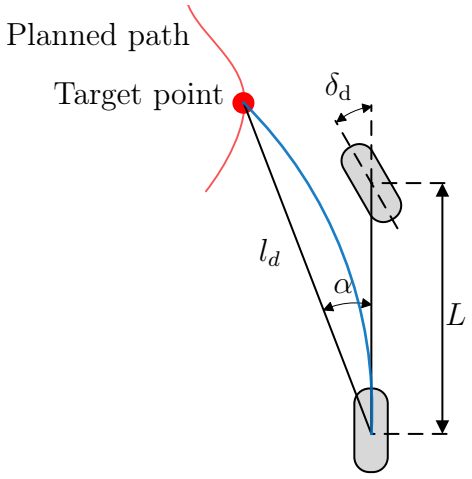


Figure 6.5: A depiction of the pure pursuit controller, which steers the vehicle towards a target point on the path. The symbols l_d, L, α and δ_d represent the look-ahead distance, wheelbase, angle between vehicle's heading and look-ahead distance vector, and desired steering angle respectively. Furthermore, the blue line represents the path the rear wheels travel to reach the target point.

the desired longitudinal acceleration $a_{\text{long},d}$ using proportional control according to the following conditions:

$$a_{\text{long},d} = \begin{cases} k_v \frac{a_{\max}}{v_{\max}}(v_d - v) & \text{if } v_d \geq v \\ k_v \frac{a_{\max}}{v_{\min}}(v_d - v) & \text{if } v_d < v. \end{cases} \tag{6.4}$$

Here, k_v represents the controller gain, and $a_{\max}$ is the maximum longitudinal acceleration as listed in Table 4.2. Importantly, at this layer of abstraction, it is assumed that the vehicle has a lower-level controller that accurately tracks acceleration commands [7; 84]. Furthermore, like the steering controller, the velocity controller is sampled at a rate of 100 Hz.

6.2 Applying TD3 to partial end-to-end racing

To enable the partial end-to-end agent to train using TD3, similar steps were followed as for the end-to-end agent. Specifically, we incorporated the following modifications to Algorithm 5: Line 7, in which an action is sampled form the agent, is changed to

$$a_t = [v_d, \text{path}] \leftarrow \text{scale}(\pi_\phi(o_t + \epsilon)), \quad \epsilon \sim \mathcal{N}(0, \sigma_{\text{action}}). \tag{6.5}$$

to account for the fact that partial end-to-end agents output a velocity and path, rather than an acceleration and steering angle.

Lines 8 to 14 of Algorithm 5 employ a for loop that performs N environment samples for every MDP time step. Since the steering and velocity controller, as well as the velocity constraint are included in the definition of the environment for the partial end-to-end agent, these components execute at line 8. Specifically, the desired steering angle (δ_d), and desired longitudinal acceleration ($a_{\text{long},d}$) is sampled using Equations 6.3 and 6.4, respectively. The velocity contstraint component then limits the velocity using Equation 5.1. The remainder of Algorithm 5 were unmodified for use with the Partial end-to-end agent.

Besides the modifications to the TD3 algorithm, two important considerations when implementing TD3 are the designs of the critics and reward signal. The architecture adopted for the critic DNN is analogous to that of the actor. It accepts a normalized observation and action as input, and outputs the action-value. It comprises three layers, the first two of which are identical to the actor (i.e., the input and hidden layers comprise m_1 and m_2 neurons with ReLU activation functions, respectively). Furthermore, the output layer has a single neuron with a linear activation. Lastly, the reward signal from Equation 5.3, which was used for the end-to-end agent, was adopted for use with the partial end-to-end agent.

After implementing these changes to the TD3 algorithm, Algorithm 6 was adapted to evaluate partial end-to-end agents. This was done by substituting Equation 6.5 into line 4, which samples an action from the agent. Additionally, Equations 6.3 and 6.4 were added in-between lines 2 and 3 to sample control actions. By incorporating these adjustments, both end-to-end and partial end-to-end agents were evaluated under identical conditions. Specifically, no exploration noise was added to the actions selected by the partial end-to-end agent, while Gaussian noise was introduced to its observation. As with the end-to-end agent, this Gaussian noise had standard deviations of 0.025 m for x and y coordinates, 0.05 rads for heading, 0.1 m/s for velocity, and 0.01 m for each element of the LiDAR scan.

6.3 Empirical design and hyper-parameter values

The partial end-to-end algorithm, as well as modifications to TD3, have been introduced with symbolic hyper-parameter values. As with the end-to-end algorithm, these values were tuned experimentally for optimal performance. The values that were determined as locally optimal for all three tracks (i.e., Porto, Barcelona-Catalunya, and Monaco) are listed in Table 6.1.

To determine these hyperparameters, the same tuning procedure as for the end-to-end agent was followed, with the addition of hyper-parameters associated with the velocity and steering controller. This procedure involved repeatedly training agents using the algorithm described in Section 6.2 with various values of the hyper-parameter under consideration, while keeping all of the other hyper-parameters fixed at the values listed in Table 6.1. Furthermore, three agents were trained for every hyper-parameter set to ensure consistency in the results.

Since the hyper-parameter tuning procedure was discussed in detail for end-to-end agents in Chapter 5, the complete hyper-parameter tuing procedure for parameters that were already shown will not be presented in this chapter. However, the process of tuning the agent sampling rate (f_{agent}) is demonstrated as a representative sample of the hyper-parameter tuning procedure. Figure 6.6 illustrates the results obtained while training agents on the Barcelona-Catalunya track with sampling rates ranging from 2 Hz to 20 Hz. The figure shows the percentage of failed laps, lap time, and reward achieved by these agents. It is worth noting that agents utilising each sample rate achieved a 0% failure rate within the first 50 training episodes. Furthermore, with the exception of agents utilizing a 5 Hz agent sampling rate, the lap time of all other agents convergence to approximately 47.3 seconds. Similarly, the reward of all agents, except those using a 5 Hz agent sampling rate, converged to a value of 22.5. These results indicate that our partial end-to-end algorithm design is more robust towards hyper-parameter changes than the

Hyper-parameter	Symbol	Value (Porto and Barcelona-Catalunya)	Value (Monaco)
Algorithm			
Maximum time steps	M	$5 \cdot 10^4$	$5 \cdot 10^4$
Target update rate	τ	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$
Replay buffer size	$\mathcal{B}$	$5 \cdot 10^5$	$5 \cdot 10^5$
Replay batch size	B	400	400
Exploration noise standard deviation	σ_{action}	0.1	0.1
Reward discount rate	γ	0.99	0.99
Agent samples between network updates	d	2	2
Agent sample rate	f_{agent}	5 Hz	5 Hz
Target action noise standard deviation	$\tilde{\sigma}$	0.2	0.2
Target action noise clip	c	0.5	0.5
Reward signal			
Distance reward	r_{dist}	0.2	0.2
Time step penalty	r_{time}	0.01	0.01
Collision penalty	$r_{\text{collision}}$	-5	-5
Observation			
Number of LiDAR beams	L	20	20
Neural network			
Learning rate	α	10^{-3}	10^{-3}
Input layer size	m_1	400	400
Hidden layer size	m_2	300	300
Velocity constraints			
Minimum velocity	v_{min}	3 m/s	3 m/s
Maximum velocity	v_{max}	5 m/s	5 m/s
Velocity controller			
Gain	k_v	0.5	0.5
Steering controller			
Look-ahead gain	k_s	0.1	0.1
Look-ahead constant	L_c	1 m	1 m

Table 6.1: Experimentally determined values of hyper-parameters for the partial end-to-end agents trained to race on all three tracks.

end-to-end algorithm. Additionally, the agent sampling rate was chosen conservatively as 10 Hz.

6.4 Steering controller tuning

To determine optimal values for the pure pursuit steering controller's look-ahead gain (k_s) and look-ahead constant (L_c) (as in Equation 6.2), a series of experimental evaluations was conducted. Initially, the focus of these evaluations was on assessing the impact of varying L_c while keeping k_s constant on the tracking capabilities of the pure pursuit controller. During the first experiment, the vehicle was tasked with following a straight line, while exclusively sampling actions from the pure pursuit controller. A constant speed of 3 m/s

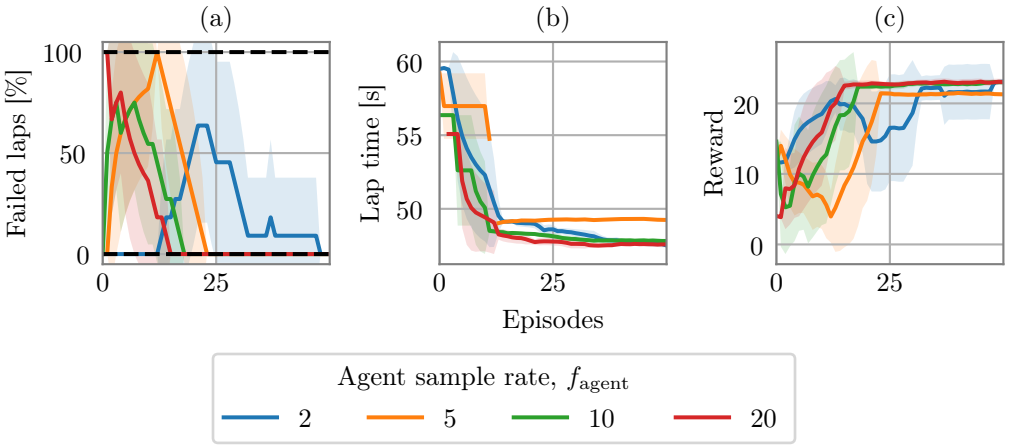


Figure 6.6: (a) The percentage failed laps and (b) average lap time of completed laps during training, as well as (c) the average learning curves of three partial end-to-end agents utilising agent sampling rate (f_{agent}) values ranging from 2 Hz to 20 Hz.

was assigned to the vehicle. Furthermore, the look-ahead gain k_s was set to a small value of 0.1, based on the finding by Patnaik et al. [83] that it should not be the dominant term in determining the look-ahead distance. The vehicle was initially positioned parallel to the path, with a distance of 0.5 m separating them. The experiment was repeated for L_c values between 0.2 and 2 meters.

The paths taken by the vehicles in this experiment are visually represented in Figure 6.7. We observed that while shorter look-ahead distances result in smaller tracking errors, they also cause steering oscillation. On the other hand, longer look-ahead distances result in less oscillation but larger tracking error. While an L_c value of 0.2 m produced extreme oscillations, controllers with an L_c value of 2 m took excessively long to reduce the positional error between the vehicle and path.

Based on these findings, agents were trained to race on the Barcelona-Catalunya track with look-ahead constants of 0.5, 1 and 1.5 meters, while setting the remaining hyper-

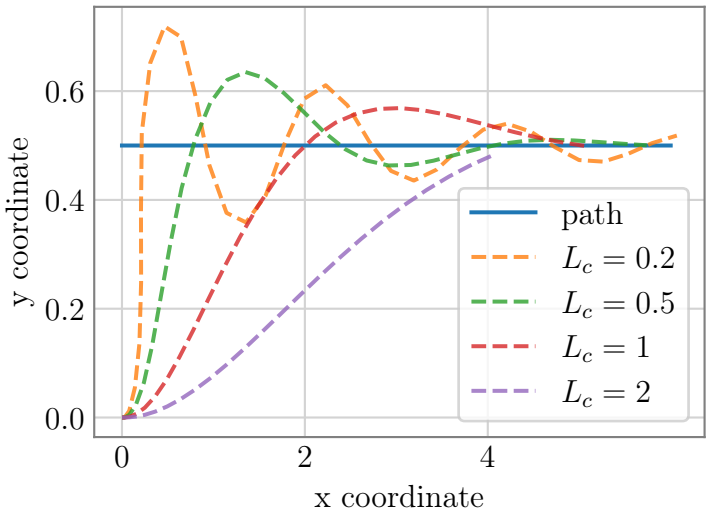


Figure 6.7: Trajectories taken by vehicles utilising a pure pursuit steering controller following a the straight blue line.

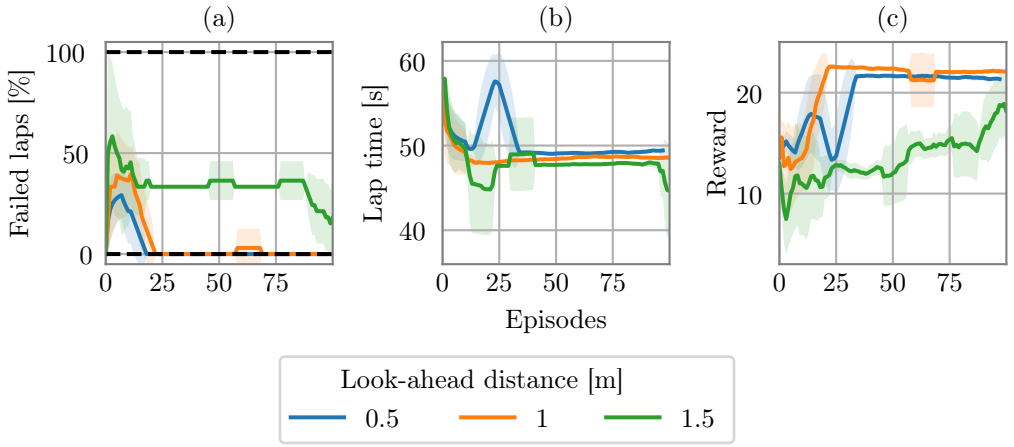


Figure 6.8: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of partial end-to-end agents with different pure pursuit look-ahead distances racing on Circuit de Barcelona-Catalunya.

parameters equal to those listed in Table 6.1. The average lap time, percentage failed laps and average cumulative episode reward during training for three agents trained with each look-ahead distance are graphed in Figure 6.8.

From Figure 6.8, partial end-to-end agents trained with look-ahead constant greater than 1 meter did not effectively learn to reduce their failure rate to 0%, whereas agents trained with look-ahead constants of 0.5 and 1 meters do. When comparing the agents trained with look-ahead constants of 0.5 and 1 meters, it is worth noting that the agents with an L_c of 1 meter achieve a slightly faster average lap time. In terms of overall performance, the agents trained with a look-ahead constant of 1 meter outperform the other agents and achieve the highest reward. The look-ahead constant L_c was therefore selected as 1 meter.

To verify that the pure pursuit steering controller is working, we compared the paths executed by agents with a look-ahead distance of one meter, to the paths that were selected by the RL agent. A comparison of these two paths is illustrated in Figure 6.9,

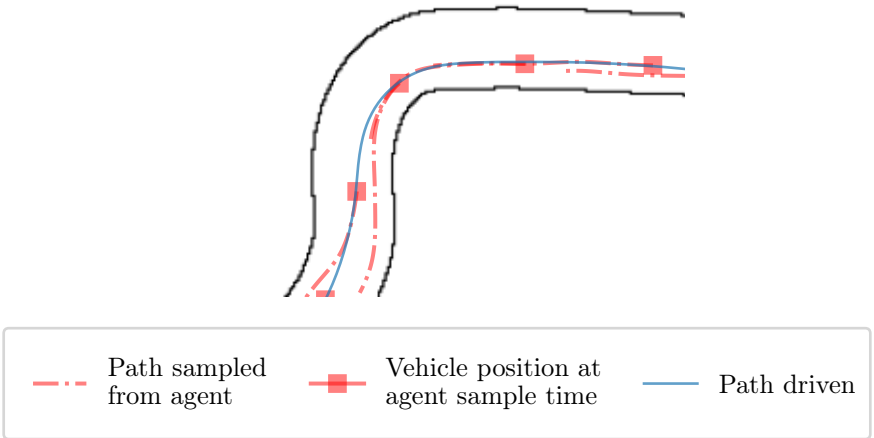


Figure 6.9: The path driven by a partial end-to-end agent on a section of Circuit de Barcelona-Catalunya, along with planned paths sampled from the agent.

which shows that the paths chosen by the agent do not coincide with the path travelled by the vehicle. This is because the pursuit controller utilises a geometric, rather than a dynamic model of the vehicle, and is less accurate at higher speeds. Interestingly, the agent outputs a path that approaches the track's edge, leading to a driven path that remains close to the center. Thus, the planner agent appears to be 'compensating' for the controller tracking performance. This is attributed to the fact that the agent is trained with the controllers in place.

6.5 Velocity controller tuning

Alongside the steering controller hyper-parameters, the velocity controller gain is also an important parameter to tune to optimise the performance of the vehicle. The velocity controller gain k_v (found in Equation 6.4) was tuned experimentally by training agents to race on Barcelona-Catalunya with k_v values of 0.5, 1 and 2. The percentage successful laps and average lap time that three agents achieved while utilising each of the k_v values and racing under evaluation conditions are given in Table 6.2. From this table, we observe that each agent completed all of its laps, and that the differences in lap time between agents are minimal.

Velocity controller gain, k_v	**Successful laps [%]**	**Lap time [s]**
0.5	100	48.37
1	100	48.34
2	100	47.34

Table 6.2: Results from agents racing under evaluation conditions using different values for the velocity controller gain (k).

A qualitative evaluation of the trajectories taken by agents utilising each velocity controller gain was therefore conducted. The paths followed by agents employing different controller gains on the final section of Barcelona-Catalunya is shown in Figure 6.10. Agents utilizing controller gains of 0.5 and 1 exhibit similar trajectories, while those using a controller gain of 2 tend to understeer at some corners. This behavior is evident at the final corner, where the agent with a controller gain of 2 approaches the outer edge of the track. As a result, we opt for a controller gain of 1 to balance vehicle safety and performance.

6.6 Racing without model uncertainties

Having determined a locally optimal hyper-parameter set for the partial end-to-end algorithm, we proceeded to assess its performance in comparison to the end-to-end baseline under conditions without model-mismatches. Figure 6.11 presents the average training performance, in terms of failed laps and lap time, of 10 partial end-to-end, as well as 10 fully end-to-end agents trained on the Barcelona-Catalunya and Monaco tracks. A trend is observed across both tracks: the partial end-to-end agents achieve a near 0% failure rate early in training, while the end-to-end agents continue to experience crashes throughout

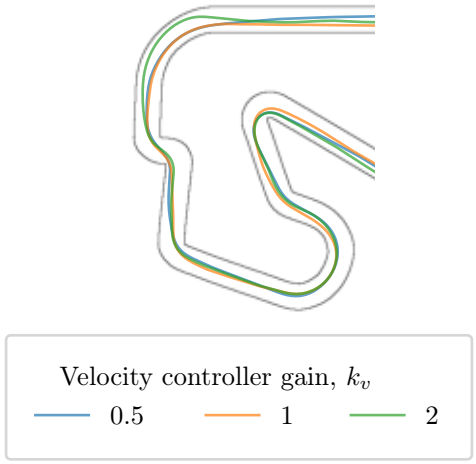


Figure 6.10: Paths taken by partial end-to-end agents utilising controller gain (k_v) values of 0.5, 1 and 2 on the final section of Barcelona-Catalunya.

the training process. However, it is worth noting that both the partial and fully end-to-end agents achieve similar lap times for the laps that are successfully completed.

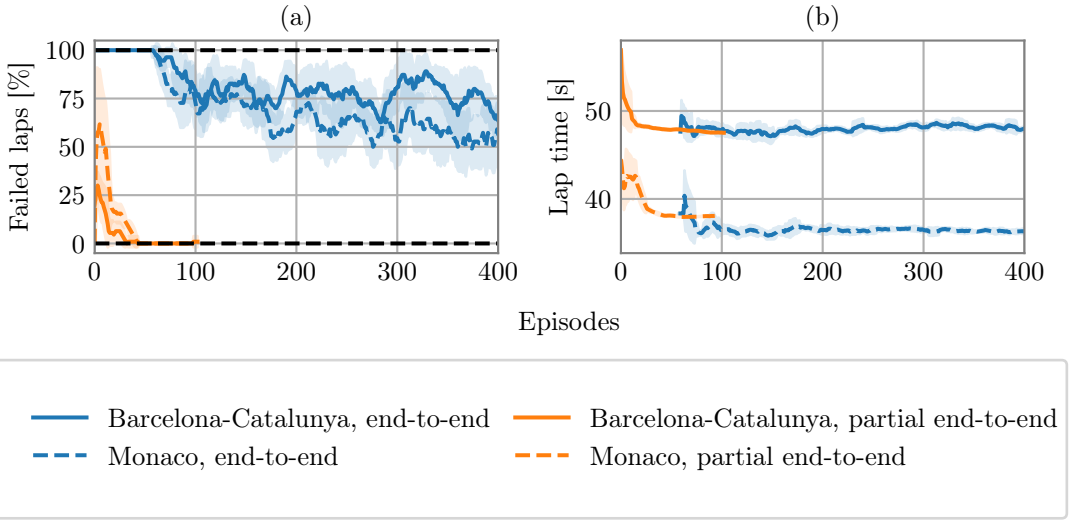


Figure 6.11: (a) Percentage failed laps and (b) lap time of partial and fully end-to-end agents during training. Dashed lines indicate agents trained to race on Barcelona-Catalunya, while solid lines indicate agents trained to race on Monaco.

Based on these results, utilising a trajectory planning approach coupled with a controller offers distinct advantages over end-to-end methods during training. In particular, many collisions can be avoided by constraining the generated paths such that they do not intersect with the track boundary. Thus, partial end-to-end agents have embedded knowledge of the track, and can avoid the boundaries. This is exemplified in Figure 6.12, which depicts the locations where an end-to-end, as well as a partial end-to-end agent crashed during one training run. Whereas the end-to-end agent experienced 726 crashes in 1170 training episodes, the partial end-to-end agent encountered only 11 crashes in 113

training episodes. Moreover, the crashes experienced by the partial end-to-end agent were as result of poor tracking performance of the pure pursuit controller at high speeds.

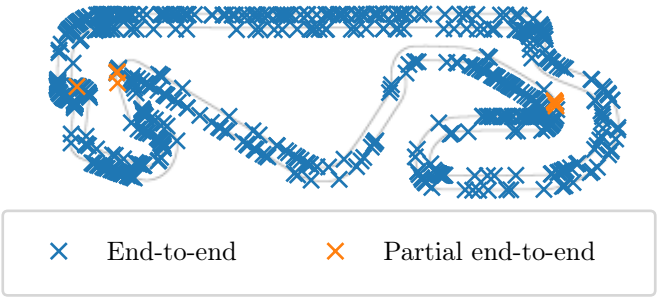


Figure 6.12: Locations where the end-to-end and partial end-to-end agents crashed during training.

The total percentage of successful laps and average lap time for 10 end-to-end and partial end-to-end agents racing under evaluation conditions on every track is presented in Table 6.3. From this table, we observe that while end-to-end agents successfully complete nearly all of their evaluation laps on the simple Porto track, they experience crashes on the more complex tracks, namely Barcelona-Catalunya and Monaco. On the other hand, partial end-to-end agents experience a low percentage of crashes on each of the the tracks. Furthermore, partial and fully end-to-end agents execute similar lap times on the Porto and Barcelona-Catalunya tracks. However, Partial end-to-end agents race slower on average on the Monaco track. This is due to an outlier partial end-to-end agent that learned to continuously select the slowest speed. Excluding this outlier, partial end-to-end agents achieve an average lap time of 35.73, which is competitive with end-to-end agents.

Track	**Algorithm**			
	End-to-end		**Partial end-to-end**	
	Successful laps [%]	Lap time [s]	Successful laps [%]	Lap time [s]
Porto	98.9	6.05	100.0	5.86
Barcelona-Catalunya	56.3	47.39	99.9	47.12
Monaco	59.2	35.63	100.0	37.91

Table 6.3: Performance of end-to-end and partial end-to-end agents racing on all three tracks under evaluation conditions.

While Table 6.3 provides the average results across a set of 10 agents, a more comprehensive depiction of the distribution of agents' performances is presented in Figure 6.13. This figure illustrates a histogram detailing the distribution of successfully completed laps under evaluation conditions. Each of the partial end-to-end agents completed all of their laps in their respective scenarios, except for one instance on the Barcelona-Catalunya

track, where 99 out of 100 laps were successfully completed. In contrast, only a single end-to-end agent out of the 10 managed to accomplish more than 90 out of 100 laps, when considering both the Barcelona-Catalunya and Monaco tracks. Furthermore, the range of evaluation outcomes for the end-to-end agents is large, with certain agents achieving less than 10 percent completion of their laps. Thus, the partial end-to-end framework allows agents to train and execute laps more consistently than an end-to-end framework.

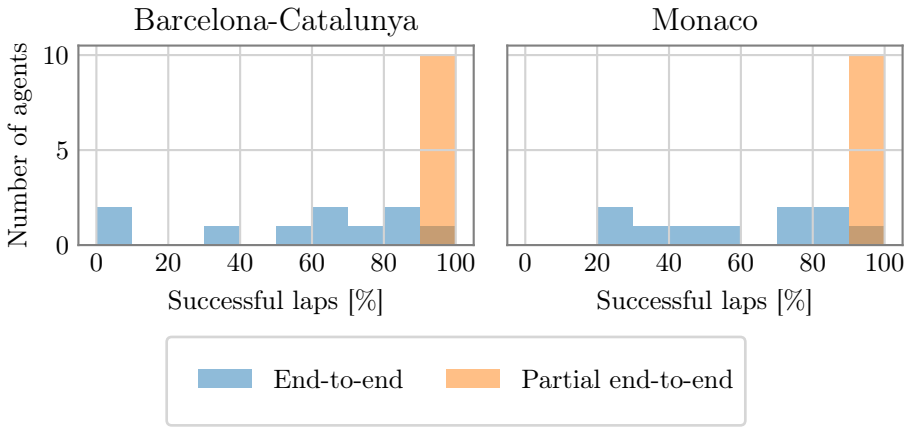


Figure 6.13: Distribution of percentage successful laps completed by agents under evaluation conditions for the Barcelona-Catalunya and Monaco tracks.

Figures 6.14 and 6.15 illustrate the paths executed by partial end-to-end agents while racing under evaluation conditions on the Barcelona-Catalunya and Monaco tracks, respectively. The velocities of these agents are color-mapped onto their paths. Furthermore, the paths taken by end-to-end agents racing on the same track is shown as a light blue dashed line. We observe from these figures that paths taken by partial end-to-end agents

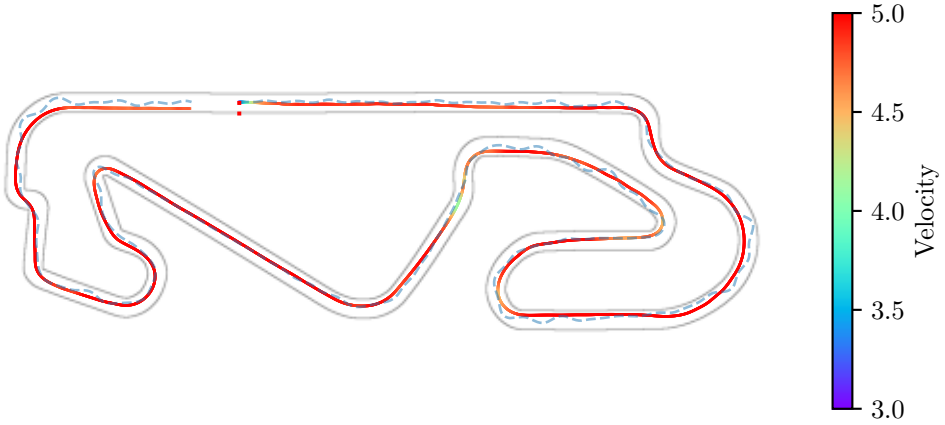


Figure 6.14: The path and velocity profile taken by a partial end-to-end agent completing Circuit de Barcelona-Catalunya. For comparison, the path of an end-to-end agent racing on the same track is depicted with a dashed light blue line.

are smooth compared to end-to-end agents. Notably, no slaloming was observed from any partial end-to-end agent. Furthermore, the partial end-to-end agents demonstrated the ability to appropriately decelerate when navigating some sharp corners.

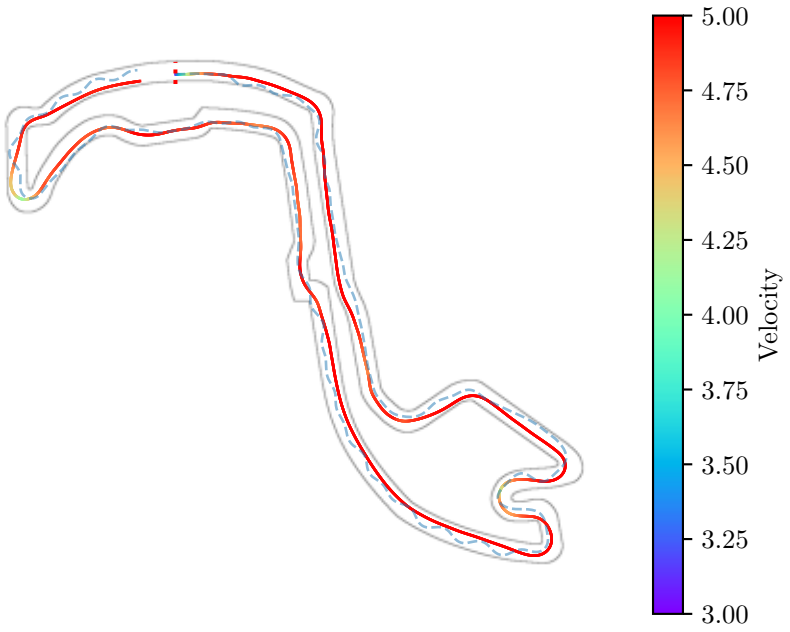


Figure 6.15: The path and velocity profile taken by a partial end-to-end agent completing Circuit de Monaco. For comparison, the path of an end-to-end agent racing on the same track is depicted with a dashed light blue line.

In Chapter 5, end-to-end agents that exhibited dangerous slaloming behavior associated with high slip angles were observed. This behavior was particularly problematic when model-mismatches were present, leading to frequent crashes. To ascertain whether the absence of slaloming behaviour resulted in a reduction in slip angles for partial end-to-end agents, we recorded the slip angles experienced by agents racing on a section of Circuit de Monaco. The path and slip angles of both partial and fully end-to-end agents are plotted in Figure 6.16. From this figure, partial end-to-end agents experience a slightly smaller peak slip angle than end-to-end agents. Additonally, the slaloming behaviour exhibited by end-to-end agents causes large oscillation in the slip angle throughout the lap. In contrast, the average slip angle exhibited by partial end-to-end agents is smaller than that of the end-to-end agent. The absence of slaloming behavior displayed by the partial end-to-end agents is a promising indication that our technique may offer improvements over the performance of end-to-end agents when model-mismatches are present.

6.7 Evaluation of alternative partial end-to-end algorithm architectures

An ablation study was conducted to assess the impact of each individual controller on the algorithm's behavior. In this study, agents were trained to race on the Barcelona-Catalunya track with either solely a steering controller or solely a velocity controller.

To ensure consistency, these partial end-to-end agents were trained using the hyperparameters listed in Table 6.1. Furthermore, three agents were trained utilising each proposed architecture. Figure 6.17 presents the percentage of failed laps and average lap time during training, as well as the learning curves for agents utilising each architecture.

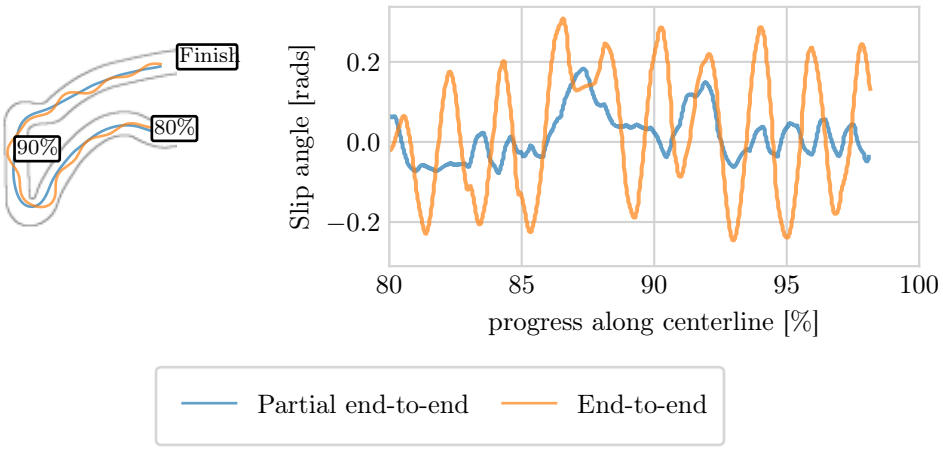


Figure 6.16: The paths and slip angles of agents racing on the final section of Circuit de Monaco.

From this figure, there is a clear distinction between agents with and without a steering controller when observing the percentage failed laps. Whereas the failure rate of agents utilising a steering controller quickly decreases to 0%, agents without a steering controller (i.e., end-to-end agents and partial end-to-end agents with solely a velocity controller) continue to crash throughout training. Interestingly, the lap time for all agents, except for those with only a steering controller, converged to a similar value of approximately 48 seconds. This is because agents relying solely on a steering controller exhibited a tendency to choose the slowest possible speed. In terms of overall performance, the partial end-to-end algorithm employing both controllers achieved a higher reward per episode compared to other agents.

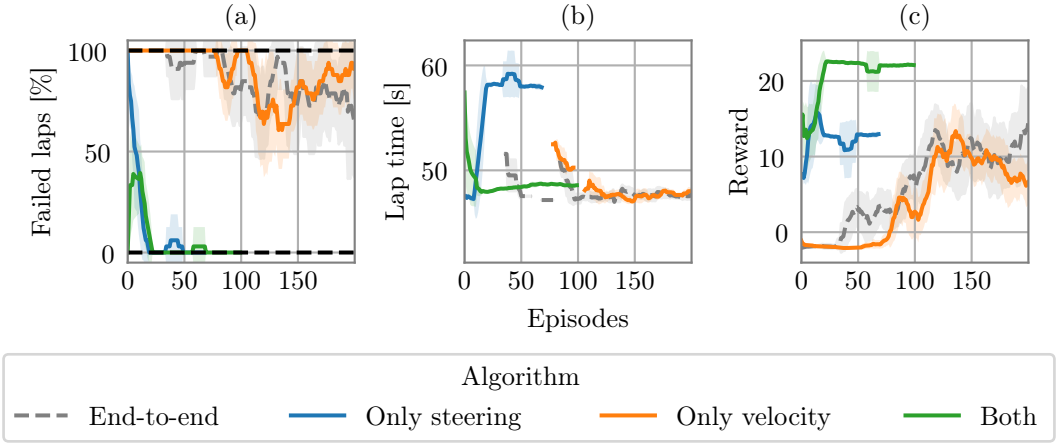


Figure 6.17: (a) The percentage failed laps and (b) lap time during training, as well as (c) the learning curves for agents utilising each algorithm structure, and trained to race on the Barcelona-Catalunya track.

As part of the ablation study, we qualitatively evaluated the trajectories executed by each proposed architecture on Barcelona-Catalunya. Figure 6.18 displays the paths taken by agents using the different architectures. The paths taken by agents without a steering controller are depicted on the left, whereas the paths taken by agents utilising a

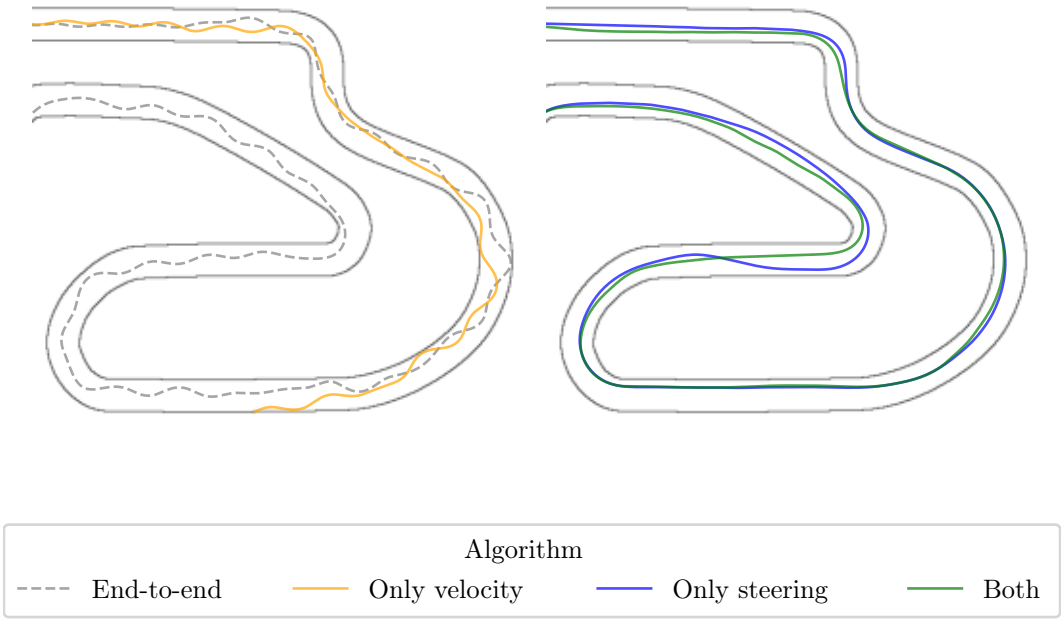


Figure 6.18: The paths taken by agents utilising each algorithm architecture. Paths taken by agents without a steering controller are depicted on the right, whereas the paths taken by agents utilising a steering controller are depicted on the left.

steering controller are depicted on the right. It is evident that partial end-to-end agents without a steering controller exhibit slaloming, similar to that of the end-to-end agent. Additionally, the behavior of both agents utilizing a steering controller are similar. We therefore determine that the steering controller is the dominant component in improving the performance of partial end-to-end agents over end-to-end agents. Furthermore, the best component configuration for partial end-to-end algorithms includes both steering and velocity control.

6.8 Summary

By constraining the path using the Frenet frame, partial end-to-end agents are able to significantly reduce the number of crashes during training and evaluation. In addition to a significant reduction in crashes, partial end-to-end agents are able to train significantly faster (in fewer than one-sixth the number of training steps) than end-to-end agents. Furthermore, when racing under evaluation conditions, partial end-to-end agents execute smooth trajectories which result in lower slip angles. This is a promising indicator that the partial end-to-end framework may offer offer further performance advantages in settings where model mismatches are present. Therefore, in the next chapter, we will assess the performance of the partial end-to-end agents under conditions where model mismatches are introduced intentionally.

Chapter 7

Racing under model uncertainty

In the previous chapter, we have observed that partial end-to-end agents have an advantage over fully end-to-end agents during both training and testing. However, the comparisons between the two racing algorithms were conducted under conditions that only accounted for uncertainty in the agent's observation by adding noise to the LiDAR scan and vehicle pose. In addition to this, when considering real-world deployment, uncertainties related to the vehicle model also emerge, leading to errors in the vehicle model itself. In this chapter, we compare our partial end-to-end algorithm with the baseline end-to-end algorithm in scenarios where the vehicle model used for evaluation differs from the one utilized during training.

Model mismatch is anticipated to pose a greater challenge in the broader context of road-going autonomous vehicles than racing vehicles, because public roads and cars are not monitored to the same extent as race cars and tracks. However, conducting experiments to assess the impact of model-mismatch on a simulated F1tenth car yields valuable insights applicable to the broader road-going problem. Specifically, we can ascertain which types of model inaccuracies jeopardize vehicle safety and determine the extent to which a given degree of model inaccuracy affects safety and performance. Accordingly, we investigated the types and magnitudes of practical modeling errors that are expected to be found in road-going cars. Our investigations encompass practical modeling errors stemming from three sources:

1. vehicle mass and mass distribution,
2. tire cornering stiffness coefficient, and
3. road surface friction coefficient.

It is important to note that our notion of model mismatch refers to a discrepancy between the vehicle model used during training, and the actual vehicle. Therefore, although online estimation of vehicle model parameters is possible, model mismatch will persist unless the updated vehicle model can be utilized to either retrain the agent online, or replace it with another agent that was trained with a set of vehicle parameters more similar to the real vehicle. These approaches may be prohibitively expensive for complex policies, such as those required by road-going vehicles. Furthermore, while model mismatching is a phenomenon that takes place when deploying agents in the real-world, our analysis is restricted to simulation to prevent unnecessary damage to physical vehicle hardware.

To simulate model-mismatch scenarios, we introduced variations in the vehicle model between the training and evaluation. Algorithm 6 was then employed to evaluate the

agent's performance when racing with the modified vehicle model. This approach is similar to the ones taken by Fuchs et al. [30] and Ghignone et al. [14]. It is important to note that these experiments were conducted on a single agent for each algorithm architecture. The selected agents were representative of the median performance of each framework.

7.1 Adding a dynamic mass

A vehicle's occupants and cargo change its total mass (m), center of gravity relative to the front axle (l_f) and moment of inertia (I_z). The parameters m, I_z and l_f appear in the heading rate and slip angle terms of the dynamic bicycle model described by Equation 4.8, which is used for higher speeds. However, only l_f appears in the slip angle term for the kinematic bicycle model in Equation 4.9. This indicates that a dynamic mass has a greater effect on the vehicle dynamics at higher speeds, where the motion of the vehicle is accurately represented by Equation 4.8.

To investigate the effect of a model mismatch in vehicle mass, we simulated the addition of various masses, ranging from 0.3 kg to 1.5 kg, along the longitudinal axis of the vehicle. Subsequently, the performance of both a partial and fully end-to-end agent was evaluated using Algorithm 6. The percentage of successful laps achieved by the agents is plotted as a function of the position of the masses along the length of the vehicle in Figure 7.1. Furthermore, we chose to perform this experiment on the Porto track, because the difference in performance between the partial and fully end-to-end agents under nominal (i.e., no model mismatch) conditions were minimal on this track.

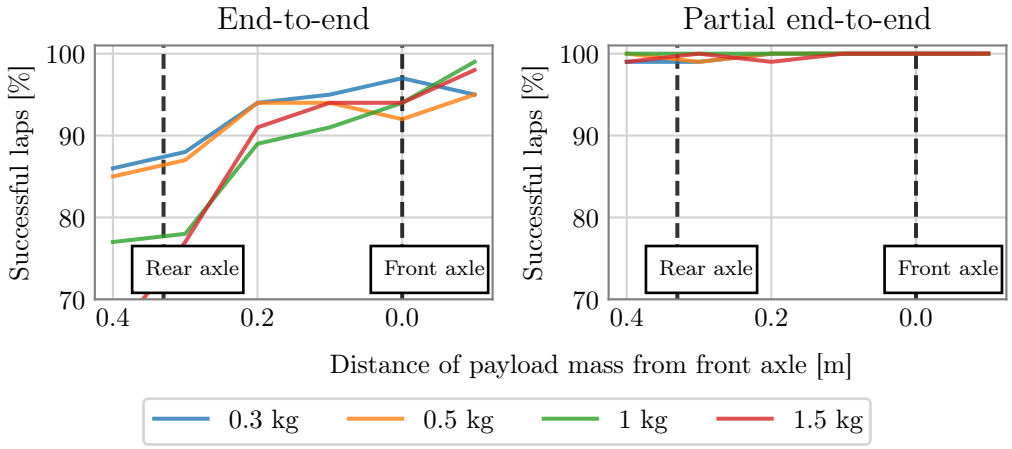


Figure 7.1: The percentage successful laps under evaluation conditions for agents with a masses placed along the longitudinal axis of the vehicle. The front and rear axles are indicated by black dashed lines.

Our findings indicate that the end-to-end agent exhibited sensitivity to an unaccounted-for mass on the vehicle, particularly when the mass was positioned towards the back. This observation is supported by the significant decrease in lap completion rate when the mass is placed closer to the rear axle. Interestingly, the end-to-end agent displayed some resilience towards a mass located at the front of the vehicle. In contrast, the partial end-to-end algorithm demonstrated a higher level of robustness against modeling errors stemming from unaccounted-for masses. Regardless of the position of the mass placement

on the vehicle, the partial end-to-end agent successfully completed a large percentage of it's evaluation laps.

In addition to observing the failure rates of agents with an added mass on the vehicle, we also qualitatively evaluated the behaviour of each agent with a 0.3 kg mass placed directly above the front axle over one lap of the Porto track. Figure 7.2 shows sample trajectories of fully and partial end-to-end agents under nominal conditions, as well as with model-mismatch, over one lap.

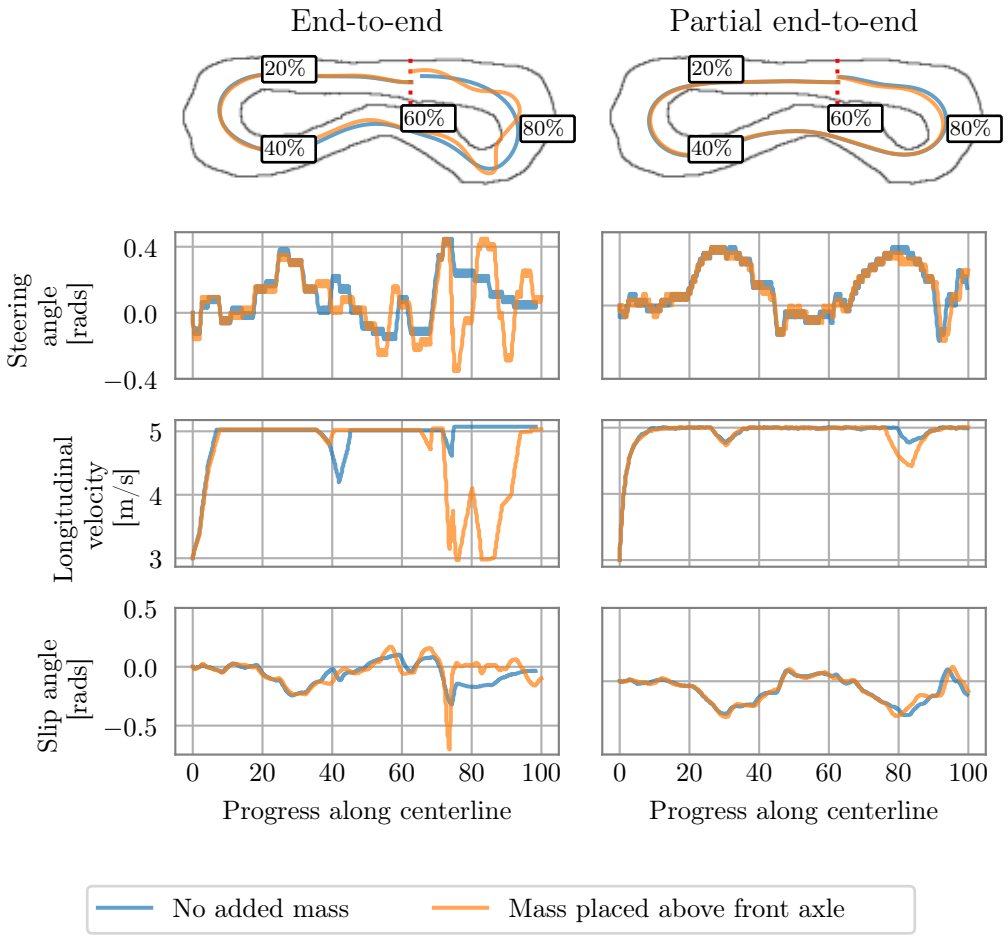


Figure 7.2: The paths, steering angles, longitudinal velocities and slip angles of fully end-to-end agents in the left column, and partial end-to-end agents in the right column, over a single lap of the Porto track. The trajectories of agents with and without the added mass above the front axle are superimposed for comparison.

In Figure 7.2, it is observed that the end-to-end agent, when racing with an added mass, begins to deviate from the nominal trajectory at approximately the halfway point. Initially, the steering angle exhibits slight oscillations. As the agent approaches the corner, these oscillations intensify, accompanied by a significant spike in slip angle, indicating a loss of control. Consequently, the velocity decreases to $v_{\min}$ in an attempt to regain control. However, even after the slip angle returns to the normal range, the agent continues to exhibit a slaloming behavior. Thus, in addition to the fact that end-to-end agents crash on 5% of the evaluation laps, their behaviour is still dangerous on laps that are completed.

In contrast, the partial end-to-end agent did not deviate from the nominal trajectory in any of the evaluation laps. Decoupling path planning from control therefore provides stability and robustness to modeling errors in the total vehicle mass for RL approaches.

7.2 Uncertain cornering stiffness

Another practical model mismatch that vehicles encounter is a discrepancy in the cornering stiffness terms $(C_{S,r}, C_{S,f})$ which characterises their tires. Tire construction and dimensions, the type and quality of the tread, and inflation pressure are significant factors when determining cornering stiffness [80]. Once again, the effect of a cornering stiffness model mismatch on the vehicle dynamics is more pronounced at higher speeds, as evident from the absence of tire stiffness terms in the kinematic model applicable at low speeds, and their inclusion in the dynamic model described by Equation 4.8.

We conducted an investigation whereby we evaluated the performance of agents after simulating changes in cornering stiffness coefficient for (a) the front tires, (b) the rear tires, then (c) both the front and rear tires together by the same percentage. These changes were once again applied in-between training and evaluation on the Porto track. During these experiments, the tire stiffnesses were varied up to ±20% of the nominal values of 4.72 and 5.45 rads^{-1} for the front and rear tires, respectively. The percentage of successful laps of agents racing with these mismatched cornering stiffness coefficients are shown in Figure 7.3.

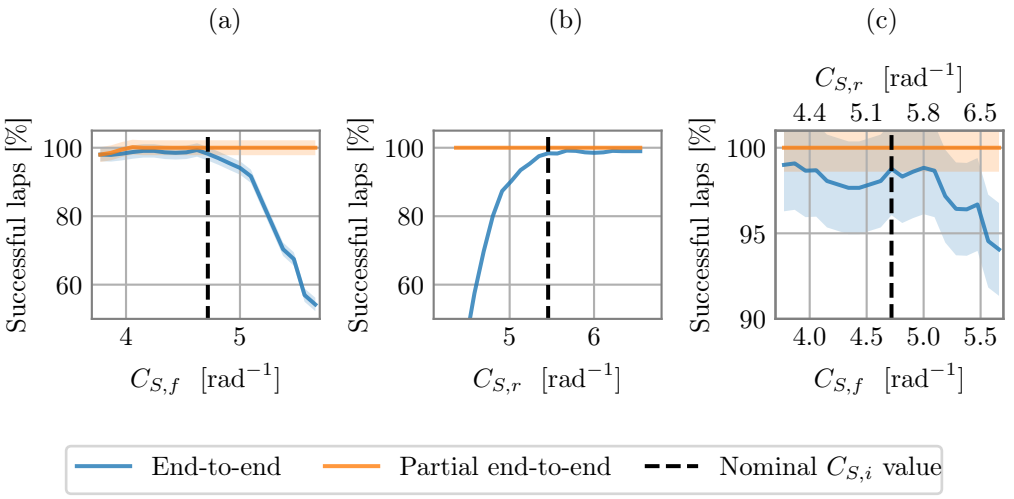


Figure 7.3: Success rates under evaluation conditions of agents with mismatched tire cornering stiffness. Subplot (a) shows the effect of varying only the front tire stiffness, (b) the effect of varying only rear tire stiffness, and (c) the effect of varying both front and rear tire stiffness together.

As is evident from Figure 7.3, the end-to-end agent is sensitive to an increase in the front cornering stiffness coefficient, while also being sensitive to a decrease in the rear cornering stiffness coefficient. In addition, when both the front and rear cornering stiffness coefficients are altered simultaneously, the end-to-end agent tends to experience crashes. In contrast, the partial end-to-end agent demonstrates resilience to changes in either front or rear tires. Although it does experience failures when both the front and rear cornering

stiffness coefficients are decreased together by 20%, the failure rate is comparatively lower than end-to-end agents.

A decrease in rear cornering stiffness coefficient was identified as the worst case scenario for the end-to-end agent. To further investigate this, we compared the trajectories executed by both the partial and fully end-to-end agents on the Porto track, considering a scenario whereby the rear cornering stiffness coefficient was decreased to 4.36 rads^{-1}. Figure 7.4 illustrates the trajectories taken by the agents with and without this model mismatch. In this evaluation, the agents' starting point is at the bottom of the track. When model-mismatch in the rear cornering stiffness coefficient is present, the end-to-end agent exhibits slaloming behavior characterized by significant oscillations in slip angle from the beginning of the lap. In this instance, the end-to-end agent crashes after the second turn.

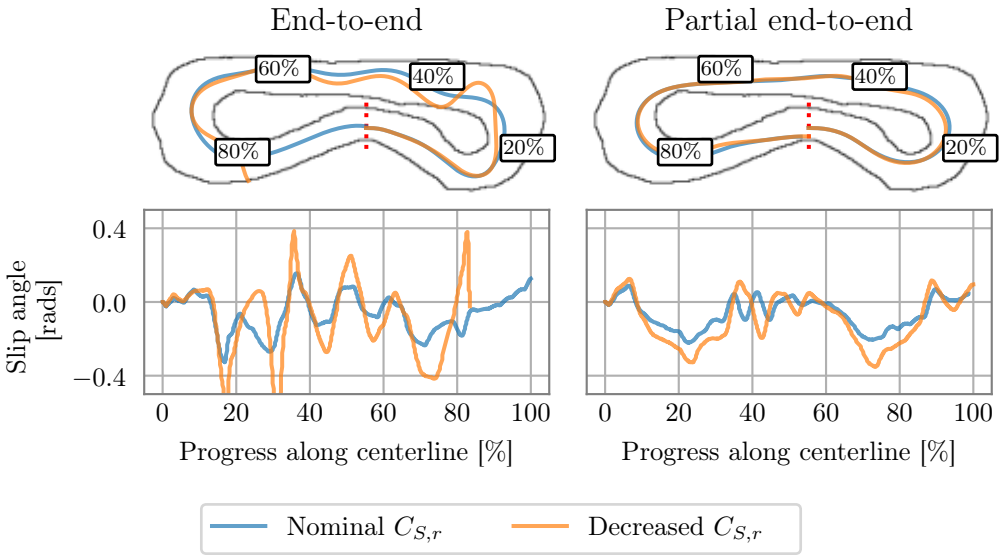


Figure 7.4: Trajectories of agents racing on Porto with and without a decreased rear cornering stiffness coefficient $C_{S,r}$. Trajectories executed by an end-to-end agent are shown in the left column, while trajectories executed by a partial end-to-end agent are shown in the right column. Furthermore, the trajectories of agents racing with and without model mismatch are superimposed for comparison.

In contrast, the trajectories followed by the partial end-to-end agents, with and without the model mismatch, are very similar. Hence, when there is a model mismatch in the cornering stiffness, partial end-to-end agents outperform end-to-end agents both quantitatively (in terms of the number of failed laps) and qualitatively (as evident from the analysis of their trajectories). Whereas end-to-end agents tend to slalom when model mismatching occur in the cornering stiffness coefficients, partial end-to-end agents demonstrate more predictable behaviour. Moreover, they did not deviate from the trajectory that they would have executed had there been no model mismatch present.

7.3 Discrepancy in road surface friction

In our final model mismatch investigation, we compared the performance of partial and fully end-to-end agents under conditions where the road surface friction coefficient value

used to train the agents is erroneous. Accurately determining the road surface friction coefficient at every point on the road surface is impractical. Moreover, the road surface friction value for a given road surface is influenced significantly by weather conditions. For instance, the minimum friction coefficient values for dry and wet asphalt are 0.7 and 0.4, whereas the minimum road surface friction coefficients for dry and wet gravel are 0.6 and 0.3, respectively [63]. As a result, there is substantial variation of friction values encountered by vehicles on both public roads and racetracks, making a mismatch in friction coefficient highly likely to occur.

For this investigation, we perturbed the road surface friction coefficient in-between training and evaluation. Agents were evaluated with road surface friction values ranging from 0.5 (representing a typical value for wet asphalt) to 1.04 (corresponding to the nominal training value for F1tenth cars given in Table 4.3) on all three tracks. Importantly, our simulator assumes a spatially and temporally uniform road friction coefficient throughout a lap. With this assumption, the worst case scenario whereby the entire road surface has a changed friction coefficient is considered.

Figure 7.5 illustrates the percentage of successful evaluation laps for both end-to-end and partial end-to-end agents when facing a mismatch in the road surface friction coefficient across all three tracks. Under nominal conditions, the end-to-end agents achieved success rates of 99%, 83%, and 61% for Porto, Barcelona-Catalunya, and Monaco, respectively. However, when considering the worst case scenario of racing on a wet asphalt surface, the success rates significantly decreased to 55%, 13%, and 8% for these respective tracks.

On the other hand, the partial end-to-end agents successfully completed all their evaluation laps under nominal conditions. When a model mismatch in the road surface friction coefficient was introduced simulating a wet asphalt surface, the partial end-to-end agent was still able to complete all of its laps on Circuit de Barcelona-Catalunya. However, it's success rate decreased to 87% and 6% for the Porto and Monaco tracks. The decrease in success rate for the partial end-to-end agent on Circuit de Monaco is especially severe. However, we observe that only agents racing with a friction coefficient

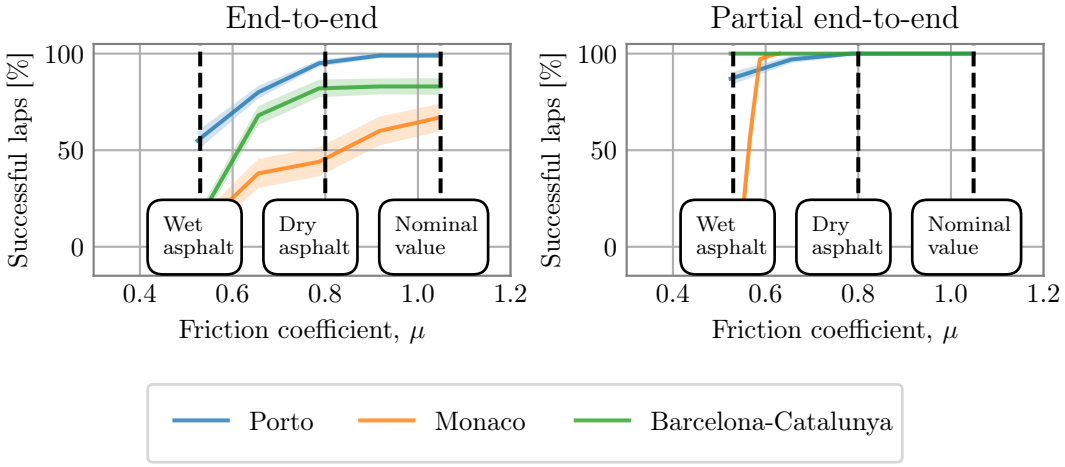


Figure 7.5: Success rates under evaluation conditions of agents racing with decreased road surface friction values. Results for the end-to-end agent racing on all three tracks are shown in the left subplot, while results for the corresponding results for partial end-to-end agents are shown on the right. The values of friction corresponding to the nominal training value, dry, as well as wet asphalt are marked with a black dashed line.

of less than 0.6 crash at all.

To investigate this large decrease in successful laps further, we observe the locations where agents crashed while racing on a wet asphalt surface on the Monaco circuit, as depicted in Figure 7.6. Apart from one crash, all the partial end-to-end agent failed at one of two corners. Conversely, end-to-end agents crashed at various parts of the track while racing with a mismatched friction coefficient.

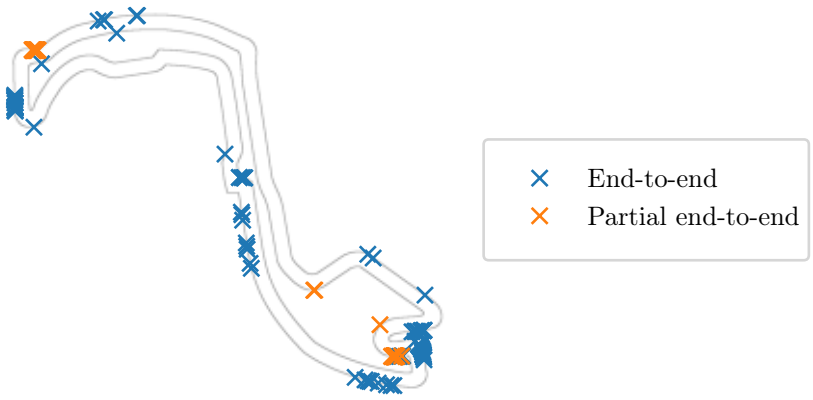


Figure 7.6: Locations where agents crashed while racing on a surface with a friction coefficient of 0.5 (corresponding to wet asphalt).

The fact that the partial end-to-end agent fails at only two locations on Circuit de Monaco when racing with a mismatched friction coefficient is an indicator that it executes trajectories consistently. To illustruate, an example of trajectories executed by both partial and fully end-to-end agents racing on a section of Circuit de Barcelona-Catalunya is shown in Figure 7.7. The figure shows the trajectories executed by agents under nominal conditions, as well as conditions with decreased surface friction. Note that the trajectories executed by the partial end-to-end agent remain similar when mismatches are introduced. The most notable difference between trajectories executed by the partial end-to-end agent on the nominal and slippier surfaces, is that the trajectories executed on the slippier surfaces exhibit reduced curvature, resulting in wider paths being followed by the agents. In contrast, the trajectories by the end-to-end agent are always erratic. Thus, the partial end-to-end agent outperformed the end-to-end agent on the majority of road surface conditions.

7.4 Summary

In this chapter, we conducted a comprehensive comparison between our partial end-to-end solution and the end-to-end baseline under conditions where model mismatch is present. Specifically, we introduced model mismatches by using a slightly different vehicle model for training compared to the one used for evaluating the agents. This approach allowed us to simulate the uncertainties often encountered in real-world deployments.

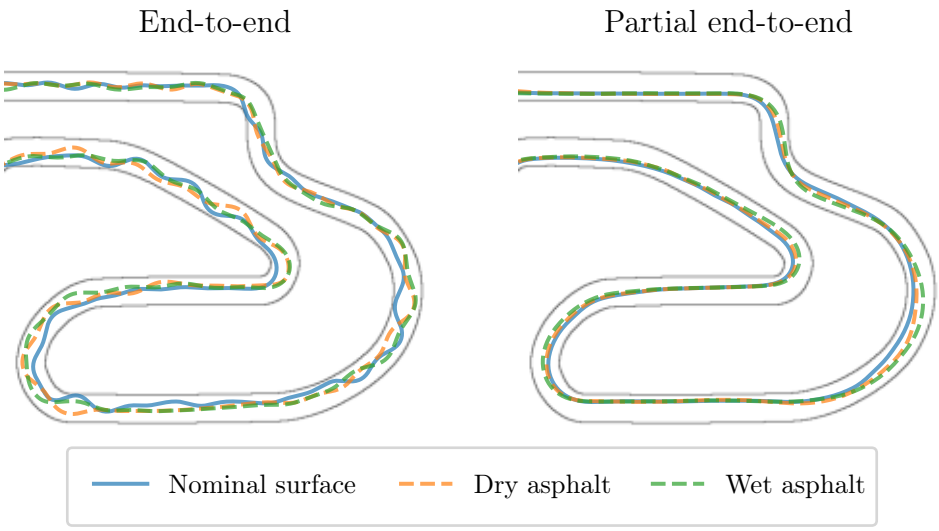


Figure 7.7: Trajectories of agents racing on a section of Barcelona-Catalunya under nominal conditions, as well as with road surface friction values corresponding to dry asphalt and wet asphalt.

In our investigation, we examined three specific scenarios involving practical model mismatches: errors in (a) the mass, (b) tire cornering stiffness coefficient, and (c) road surface friction coefficient. Across all three scenarios, end-to-end agents performed poorly, as evident by their tendency to crash, as well as to exhibit erratic behavior characterized by slaloming. The erratic behaviour of end-to-end agents highlights their drawbacks in scenarios where model mismatches are present.

In contrast to end-to-end agents, the partial end-to-end agents displayed more predictable behavior. The trajectories of these agents did not deviate significantly from the trajectories observed when racing without any model mismatches. This observation held true across all three model mismatch scenarios considered. These results demonstrate that combining an RL planner with a classic feedback controller is advantageous when considering scenarios in which practical model mismatches are present. Feedback controllers not only offer faster operation but also greater predictability than end-to-end agents, as they are designed to minimize the error between the vehicle and its intended trajectory.

Chapter 8

Conclusion

Existing learning-based solutions to autonomous racing often fail to integrate insights from classical approaches, resulting in algorithms that are not robust against practical model mismatches. Consequently, the performance of many learning-based approaches deteriorates when faced with such scenarios.

This thesis considered the design and implementation of a learning-based autonomous racing algorithm that demonstrates robustness towards practical model mismatches by incorporating insights from classical approaches. In this chapter, we assess the progress and development of this project based on the aim and objectives outlined in Chapter 1.

8.1 Work completed

The aim of this research project was to develop a learning-based racing algorithm that is robust towards practical model mismatches that vehicles may encounter during real-world deployment. To achieve this goal, several key steps were undertaken, including the implementation of a bicycle vehicle model that accurately captures high-speed dynamics, the development of a baseline end-to-end learning-based solution for comparative analysis, and the design of our algorithm. Additionally, an investigation was conducted to evaluate the performance of both our solution and the baseline approach under conditions involving model mismatches.

Chapter 5 presents the end-to-end baseline solution, which directly maps observation data to control commands. When evaluated on the relatively simple Porto track, end-to-end agents demonstrated competitive performance, successfully completing all of their laps. However, end-to-end agents exhibited erratic behaviour such as slaloming when tasked with racing on longer and more intricate tracks. Furthermore, they failed to complete a significant portion of their evaluation laps on longer tracks.

Additionally, during a preliminary investigation into the robustness of end-to-end algorithms towards a decrease in road surface friction, we observed that the agent tended to crash. To enhance robustness towards model mismatches in the friction coefficient, domain randomisation was investigated by adding Gaussian noise to the nominal friction coefficient at the start of every episode. However, randomising the domain offered no major benefits in agent performance. The unsatisfactory performance of end-to-end agents on longer tracks and under model mismatch conditions, combined with the limited effectiveness of domain randomisation techniques further motivated the development of our solution.

Our proposed solution, which is presented in Chapter 6, introduced a partial end-to-end autonomous driving algorithm that decouples the planning and control tasks. Within the partial end-to-end framework, an RL agent generates a trajectory comprising a path and velocity, which is subsequently tracked using a pure pursuit steering controller and a proportional velocity controller, respectively. To facilitate path generation, we allowed agents to select the constraints of a cubic polynomial function in the Frenet frame, enabling the consistent generation of smooth and continuous paths.

By specifying the path within the Frenet frame, partial end-to-end agents were able to execute smooth trajectories, and also constrained the agents to select paths that remained within the track boundaries. As a result, training time was reduced significantly, and the number of collisions during training was minimal compared to end-to-end agents. Additionally, the partial end-to-end agents successfully completed all evaluation laps under conditions where no model mismatch was present, even on complex circuits such as Barcelona-Catalunya and Monaco. Thus, even under nominal conditions, partial end-to-end agents demonstrated substantial advantages over their end-to-end counterparts.

Following the evaluation of the partial end-to-end algorithm under ideal conditions, we examined its performance in scenarios involving practical model mismatches. These investigations specifically considered model mismatches in the vehicle mass, cornering stiffness coefficient, and road surface friction coefficient. In each of the model mismatch scenarios analyzed, the partial end-to-end agent consistently outperformed the baseline end-to-end agent. Notably, the partial end-to-end agents exhibited similar trajectories prior to and following the introduction of model mismatches. This trend was particularly evident when model mismatches related to the vehicle mass and cornering stiffness were considered. Thus, the decoupling of trajectory generation and control, combined with the use of feedback controllers, offers better performance than an end-to-end approach to vehicle control in model mismatch conditions.

8.2 Future work

The performance of partial end-to-end agents under nominal conditions, as well as conditions where model mismatches are present, highlights the potential of these algorithms in developing robust learning-based driving systems that can handle uncertainties in vehicle dynamics. To further strengthen the case for the partial end-to-end approach, the next development step is to conduct model mismatch experiments on a physical vehicle. This would contribute to building a stronger argument for the practical implementation of these algorithms in real-world driving scenarios.

Although our experiments focused on racing, the results have implications for addressing the broader road-going autonomous driving problem. Future work should therefore investigate the application of partial end-to-end algorithms to road-driving scenarios. For example, when considering the road-going autonomous driving task, decoupling planning from control may provide additional benefits, as it allows the planner and controller to be developed independently of each other. This approach facilitates the use of a single generic planning agent that can be applied to similar car models, while specific controllers are developed for each car model. Consequently, it may lead to consistent planning behavior across a fleet of vehicles, as well as enable easier algorithm development.

In closing, learning-based algorithms are gaining popularity in road-going production cars due to their scalability. It is therefore critical to ensure that learning-based driving algorithms handle the sim-to-real gap as best as possible. As part of the sim-to-real gap,

these algorithms should be robust towards practical model mismatches. In this thesis, we have found that the partial end-to-end framework represents a promising solution for designing learning-based algorithms that are robust towards practical model mismatches. Furthermore, the proposed solution trains quickly and executes safe trajectories with very few crashes compared to the baseline end-to-end agent.

Appendices

Appendix A

Supporting results

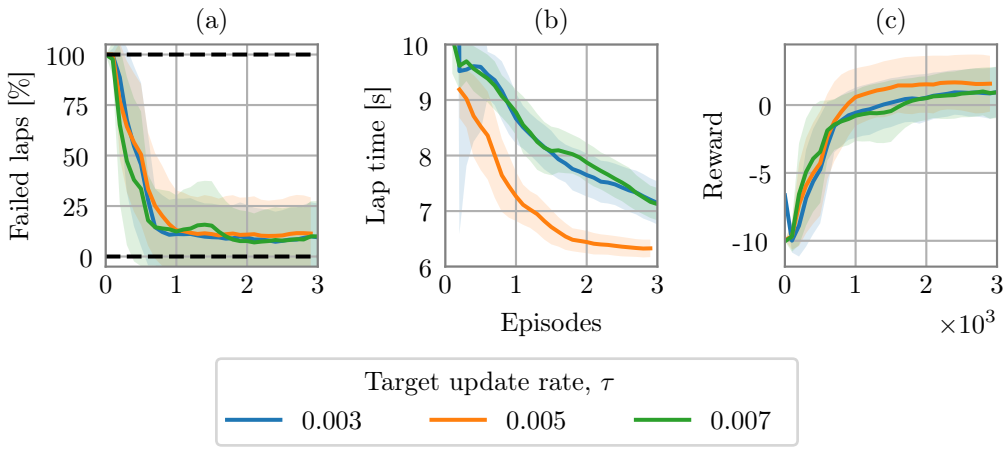


Figure A.1: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with target update rates ranging from 0.003 to 0.007.

Target update rate, τ	**Successful test laps [%]**	**Average test lap time [s]**	**Standard deviation of test lap time [s]**
$3 \cdot 10^{-3}$	99	6.85	1.23
$5 \cdot 10^{-3}$	100	6.07	0.20
$7 \cdot 10^{-3}$	96	6.94	0.74

Table A.1: Evaluation results and training time of end-to-end agents with target update rates ranging from $3 \cdot 10^{-3}$ to $7 \cdot 10^{-3}$.

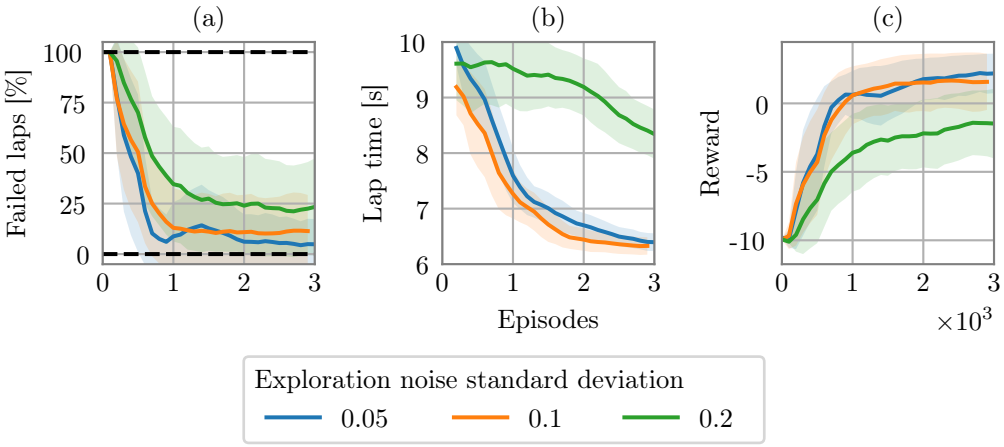


Figure A.2: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with exploration noise standard deviations ranging from 0.05 to 0.2.

Exploration noise standard deviation, σ_{action}	**Successful test laps [%]**	**Average test lap time [s]**	**Standard deviation of test lap time [s]**
0.05	96	6.13	0.46
0.1	100	6.07	0.20
0.2	100	7.27	0.67

Table A.2: Evaluation results and training time of end-to-end agents with exploration noise varying from 0.05 to 0.15.

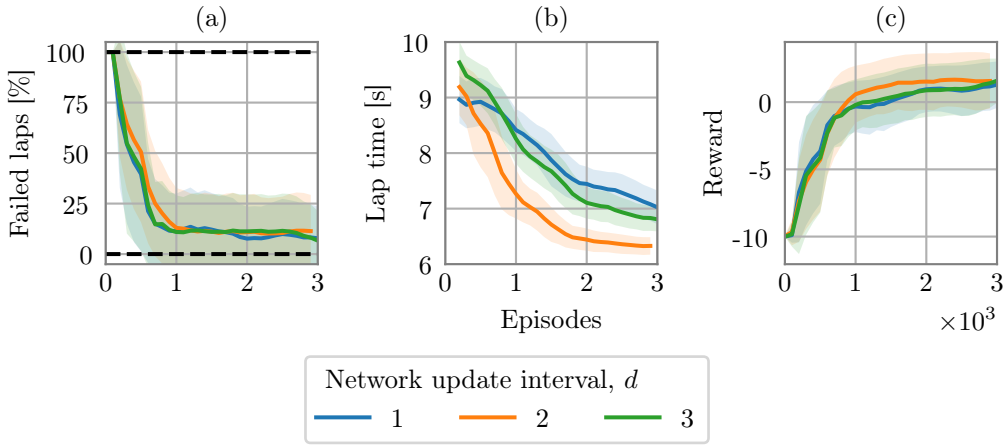


Figure A.3: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with network update intervals d ranging from 1 to 3.

Number of action samples between network updates, d	**Successful test laps [%]**	**Average test lap time [s]**	**Standard deviation of test lap time [s]**
1	99	6.85	1.23
2	100	6.07	0.20
3	96	6.94	0.74

Table A.3: Evaluation results and training time of end-to-end agents with number of action samples between network updates ranging from 1 to 3.

List of References

[1] Klaver, F.: The economic and social impacts of fully autonomous vehicles. 2020. [Online; accessed 30-June-2022].
Available at: https://www.compact.nl/en/articles/the-economic-and-social-impacts-of-fully-autonomous-vehicles/

[2] Barabás, I., Todoruţ, A., Cordoş, N. and Molea, A.: Current challenges in autonomous driving. *IOP Conference Series: Materials Science and Engineering*, vol. 252, p. 012096, oct 2017.
Available at: https://doi.org/10.1088/1757-899x/252/1/012096

[3] Tian, h., NI, J. and HU, J.: Autonomous driving system design for formula student driverless racecar. In: *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1–6. 2018.
Available at: https://doi.org/10.1109/IVS.2018.8500471

[4] Wischnewski, A., Geisslinger, M., Betz, J., Betz, T., Fent, F., Heilmeier, A., Hermansdorfer, L., Herrmann, T., Huch, S., Karle, P., Nobis, F., Ögretmen, L., Rowold, M., Sauerbeck, F., Stahl, T., Trauth, R., Lienkamp, M. and Lohmann, B.: Indy autonomous challenge - autonomous race cars at the handling limits. In: Pfeffer, P. (ed.), *12th International Munich Chassis Symposium 2021*, pp. 163–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 2022.
Available at: https://doi.org/10.1007/978-3-662-64550-5_10

[5] Babu, V.S. and Behl, M.: f1tenth.dev - an open-source ros based f1/10 autonomous racing simulator. In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pp. 1614–1620. 2020.
Available at: https://doi.org/10.1109/CASE48305.2020.9216949

[6] F1tenth Foundation: F1tenth. 2020. [Online; accessed 9-September-2022].
Available at: https://f1tenth.org/

[7] Betz, J., Zheng, H., Liniger, A., Rosolia, U., Karle, P., Behl, M., Krovi, V. and Mangharam, R.: Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.
Available at: https://doi.org/10.1109/OJITS.2022.3181510

[8] Evans, B., Engelbrecht, H.A. and Jordaan, H.W.: Reward signal design for autonomous racing. *International Conference on Advanced Robotics*, 2021.
Available at: http://arxiv.org/abs/2103.10098

[9] Tatulea-Codrean, A., Mariani, T. and Engell, S.: Design and simulation of a machine-learning and model predictive control approach to autonomous race driving for the F1/10 platform. *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6031–6036, 2020.
Available at: https://doi.org/10.1016/j.ifacol.2020.12.1669

[10] Song, Y., Lin, H., Kaufmann, E., Duerr, P. and Scaramuzza, D.: Autonomous overtaking in gran turismo sport using curriculum reinforcement learning. 2021.
Available at: https://doi.org/10.48550/arXiv.2103.14666

[11] Wurman, P.R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T.J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., Gilpin, L., Khandelwal, P., Kompella, V., Lin, H., MacAlpine, P., Oller, D., Seno, T., Sherstan, C., Thomure, M.D., Aghabozorgi, H., Barrett, L., Douglas, R., Whitehead, D., Durr, P., Stone, P., Spranger, M. and Kitano, H.: Outracing champion gran turismo drivers with deep reinforcement learning. February 2022.
Available at: https://doi.org/10.1038/s41586-021-04357-7

[12] Zhao, W., Queralta, P. and Westerlund, T.: Sim-to-real trainsfer in deep reinforcement learning for robotics: a survey. *IEEE Symposium Series on Computational Intelligence*, pp. 737–744, 2020.
Available at: https://doi.org/10.48550/arXiv.2009.13303

[13] Zhoa, Y.-Q., Li, H.-Q., Lin, F., Wang, F. and Ji, X.-W.: Estimation of road friction coefficient in different road conditions based on vehicle braking dynamics. *Chinese Journal of Mechanical Engineering*, vol. 36, pp. 982–990, January 2017.
Available at: https://doi.org/10.1007/s10033-017-0143-z

[14] Ghignone, E., Baumann, N., Boss, M. and Magno, M.: TC-Driver: Trajectory Conditioned Driving for Robust Autonomous Racing - A Reinforcement Learning Approach. In: *International conference on robotics and automation*. 2022. 2205.09370.
Available at: http://arxiv.org/abs/2205.09370

[15] Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J. and Lee, I.: Case study: Verifying the safety of an autonomous racing car with a neural network controller. *HSCC 2020 - Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control ,part of CPS-IoT Week*, 2020.
Available at: https://doi.org/10.1145/3365365.3382216

[16] Hsu, B.-J., Cao, H.-G., Lee, I., Kao, C.-Y., Huang, J.-B. and Wu, I.-C.: Image-based conditioning for action policy smoothness in autonomous miniature car racing with reinforcement learning. 2022.
Available at: https://arxiv.org/abs/2205.09658

[17] Capo, E. and Loiacono, D.: Short-Term Trajectory Planning in TORCS using Deep Reinforcement Learning. *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, pp. 2327–2334, 2020.
Available at: https://doi.org/10.1109/SSCI47803.2020.9308138

[18] Weiss, T. and Behl, M.: Deepracing: A framework for autonomous racing. *Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition*, pp. 1163–1168, 2020.
Available at: https://doi.org/10.23919/DATE48585.2020.9116486

[19] Weiss, T. and Behl, M.: Deepracing: Parameterized trajectories for autonomous racing. 2020.
Available at: https://doi.org/10.48550/arXiv.2005.05178

[20] Weiss, T. and Behl, M.: This is the way: Differential bayesian filtering for agile trajectory synthesis. *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10414–10421, 2022.
Available at: https://doi.org/10.1109/LRA.2022.3193496

[21] Evans, B., Jordaan, H.W. and Engelbrecht, H.A.: Learning the subsystem of local planning for autonomous racing. 2021.
Available at: `https://arxiv.org/abs/2102.11042`

[22] Wadekar, S.N., Schwartz, B.J., Kannan, S.S., Mar, M., Manna, R.K., Chellapandi, V., Gonzalez, D.J. and Gamal, A.E.: Towards end-to-end deep learning for autonomous racing: On data collection and a unified architecture for steering and throttle prediction. 2021.
Available at: `https://arxiv.org/abs/2105.01799`

[23] Mahmoud, Y., Okuyama, Y., Fukuchi, T., Kosuke, T. and Ando, I.: Optimizing Deep-Neural-Network-Driven Autonomous Race Car Using Image Scaling. *SHS Web of Conferences*, vol. 77, p. 04002, 2020.
Available at: `https://doi.org/10.1051/shsconf/20207704002`

[24] Pan, Y., Cheng, C.-A., Saigol, K., Lee, K., Yan, X., Theodorou, E. and Boots, B.: Agile autonomous driving using end-to-end deep imitation learning. 2017.
Available at: `https://arxiv.org/abs/1709.07174`

[25] Lee, K., Wang, Z., Vlahov, B.I., Brar, H.K. and Theodorou, E.A.: Ensemble bayesian decision making with redundant deep perceptual control policies. 2018.
Available at: `https://arxiv.org/abs/1811.12555`

[26] Schwarting, W., Seyde, T., Gilitschenski, I., Liebenwein, L., Sander, R., Karaman, S. and Rus, D.: Deep latent competition: Learning to race using visual control policies in latent space. In: Kober, J., Ramos, F. and Tomlin, C. (eds.), *Proceedings of the 2020 Conference on Robot Learning*, vol. 155 of *Proceedings of Machine Learning Research*, pp. 1855–1870. PMLR, 16–18 Nov 2021.
Available at: `https://proceedings.mlr.press/v155/schwarting21a.html`

[27] Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J.M., Boots, B. and Theodorou, E.A.: Information theoretic MPC for model-based reinforcement learning. *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1714–1721, 2017. ISSN 10504729.
Available at: `https://doi.org/10.1109/ICRA.2017.7989202`

[28] Jaritz, M., De Charette, R., Toromanoff, M., Perot, E. and Nashashibi, F.: End-to-End Race Driving with Deep Reinforcement Learning. *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2070–2075, 2018.
Available at: `https://doi.org/10.1109/ICRA.2018.8460934`

[29] Perot, E., Jaritz, M., Toromanoff, M. and Charette, R.D.: End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2017-July, pp. 474–475, 2017.
Available at: `https://doi.org/10.1109/CVPRW.2017.64`

[30] Fuchs, F., Song, Y., Kaufmann, E., Scaramuzza, D. and Durr, P.: Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, 2021.
Available at: `http://doi.org/10.1109/LRA.2021.3064284`

[31] Niu, J., Hu, Y., Jin, B., Han, Y. and Li, X.: Two-Stage Safe Reinforcement Learning for High-Speed Autonomous Racing. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, vol. 2020-Octob, pp. 3934–3941, 2020.
Available at: `https:doi.org/10.1109/SMC42975.2020.9283053`

[32] Remonda, A., Krebs, S., Veas, E., Luzhnica, G. and Kern, R.: Formula rl: Deep reinforcement learning for autonomous racing using telemetry data. 2021.
Available at: `https://arxiv.org/abs/2104.11106`

[33] Chisari, E., Liniger, A., Rupenyan, A., van Gool, L. and Lygeros, J.: Learning from Simulation, Racing in Reality. *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2021-May, no. December, pp. 8046–8052, 2021.
Available at: `https://doi.org/10.1109/ICRA48506.2021.9562079`

[34] Brunnbauer, A., Berducci, L., Brandstätter, A., Lechner, M., Hasani, R., Rus, D. and Grosu, R.: Model-based versus model-free deep reinforcement learning for autonomous racing cars. 2021.
Available at: `https://arxiv.org/pdf/2103.04909v1.pdf`

[35] Kritayakirana, K. and Gerdes, C.: Autonomous vehicle control at the limits of handling. *International Journal for Vehicle Autonomous Systems*, vol. 10, no. 4, pp. 271–296, 2012.
Available at: `https://doi.org/10.1504/IJVAS.2012.051270`

[36] Valls, M.I., Hendrikx, H.F., Reijgwart, V.J., Meier, F.V., Sa, I., Dube, R., Gawel, A., Burki, M. and Siegwart, R.: Design of an Autonomous Racecar: Perception, State Estimation and System Integration. *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2048–2055, 2018. ISSN 10504729. `1804.03252`.
Available at: `https://doi.org/10.1109/ICRA.2018.8462829`

[37] Alvarez, A., Denner, N., Feng, Z., Fischer, D., Gao, Y., Harsch, L., Herz, S., Large, N.L., Nguyen, B., Rosero, C., Schaefer, S., Terletskiy, A., Wahl, L., Wang, S., Yakupova, J. and Yu, H.: The software stack that won the formula student driverless competition. 2022.
Available at: `https://arxiv.org/abs/2210.10933`

[38] Nekkah, S., Janus, J., Boxheimer, M., Ohnemus, L., Hirsch, S., Schmidt, B., Liu, Y., Borbely, D., Keck, F., Bachmann, K. and Bleszynski, L.: The autonomous racing software stack of the kit19d. 2020.
Available at: `https://arxiv.org/abs/2010.02828`

[39] Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M. and Lohmann, B.: Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, vol. 58, no. 10, pp. 1497–1527, 2020.
Available at: `https://doi.org/10.1080/00423114.2019.1631455`

[40] Betz, J., Betz, T., Fent, F., Geisslinger, M., Heilmeier, A., Hermansdorfer, L., Herrmann, T., Huch, S., Karle, P., Lienkamp, M., Lohmann, B., Nobis, F., Ogretmen, L., Rowold, M., Sauerbeck, F., Stahl, T., Trauth, R., Werner, F. and Wischnewski, A.: Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge. *Journal of Field Robotics*, 2023.
Available at: `https://doi.org/10.1002%2Frob.22153`

[41] Kelly, D. and R.S.Sharp: Time-optimal control of the race car: a numerical method to emulate the ideal driver. *Vehicle System Dynamics*, vol. 48, no. 12, pp. 1461–1474, 2010. `https://doi.org/10.1080/00423110903514236`.
Available at: `https://doi.org/10.1080/00423110903514236`

[42] Herrmann, T., Christ, F., Betz, J. and Lienkamp, M.: Energy Management Strategy for an Autonomous Electric Racecar using Optimal Control. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pp. 720–725, 2019.

[43] Liniger, A., Domahidi, A. and Morari, M.: Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015. ISSN 10991514. 1711.07300.
Available at: https://doi.org/10.1002/oca.2123

[44] Liniger, A. and Lygeros, J.: A viability approach for fast recursive feasible finite horizon path planning of autonomous RC cars. *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC 2015*, pp. 1–10, 2015.
Available at: https://doi.org/10.1145/2728606.2728620

[45] Stahl, T., Wischnewski, A., Betz, J. and Lienkamp, M.: Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios. In: *IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3149–3154. 2019.
Available at: https://doi.org/10.1109/ITSC.2019.8917032

[46] Werling, M., Ziegler, J., Kammel, S. and Thrun, S.: Optimal trajectory generation for dynamic street scenarios in a frenet frame. In: *2010 IEEE International Conference on Robotics and Automation*, pp. 987–993. 2010.
Available at: https://doi.org/10.1109/ROBOT.2010.5509799

[47] Li, X., Sun, Z., Cao, D., Liu, D. and He, H.: Development of a new integrated local trajectory planning and tracking control framework for autonomous ground vehicles. *Mechanical Systems and Signal Processing*, vol. 87, pp. 118–137, 2017. ISSN 0888-3270. Signal Processing and Control challenges for Smart Vehicles.
Available at: https://www.sciencedirect.com/science/article/pii/S0888327015004811

[48] Coulter, R.C.: Implementation of the pure pursuit path tracking algorithm. Tech. Rep. CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburgh, PA, January 1992.
Available at: https://www.ri.cmu.edu/publications/implementation-of-the-pure-pursuit-path-tracking-algorithm/

[49] Becker, J., Imholz, N., Schwarzenbach, L., Ghignone, E., Baumann, N. and Magno, M.: Model- and acceleration-based pursuit controller for high-performance autonomous racing. 2022. 2209.04346.
Available at: https://doi.org/10.48550/arXiv.2209.04346

[50] Hoffmann, G.M., Tomlin, C.J., Montemerlo, M. and Thrun, S.: Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. *Proc. Am. Control Conf.*, pp. 2296–2301, 2007.
Available at: https://doi.org/10.1109/ACC.2007.4282788

[51] Beal, C.E. and Gerdes, J.C.: Model predictive control for vehicle stabilization at the limits of handling. *IEEE Transactions on Control Systems Technology*, vol. 21, no. 4, pp. 1258–1269, 2013. ISSN 10636536.

[52] Gotlib, A., Lukojc, K. and Szczygielski, M.: Localization-based software architecture for 1:10 scale autonomous car. *International Interdisciplinary PhD Workshop*, pp. 7–11, 2019.
Available at: http://doi.org/10.1109/IIPHDW.2019.8755418

[53] Wischnewski, A., Stahl, T., Betz, J. and Lohmann, B.: Vehicle dynamics state estimation and localization for high performance race cars. *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 307–312, 2019. ISSN 24058963.
Available at: https://doi.org/10.1016/j.ifacol.2019.08.064

[54] Anderson, J.R., Ayalew, B. and Weiskircher, T.: Modeling a professional driver in ultra-high performance maneuvers with a hybrid cost mpc. In: *2016 American Control Conference (ACC)*, pp. 1981–1986. 2016.
Available at: https://doi.org/10.1109/ACC.2016.7525209

[55] Funke, J., Brown, M., Erlien, S.M. and Gerdes, J.C.: Collision avoidance and stabilization for autonomous vehicles in emergency scenarios. *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1204–1216, 2017.
Available at: https://doi.org/10.1109/TCST.2016.2599783

[56] Jain, A., O'Kelly, M., Chaudhari, P. and Morari, M.: Bayesrace: Learning to race autonomously using prior experience. In: Kober, J., Ramos, F. and Tomlin, C. (eds.), *Proceedings of the 2020 Conference on Robot Learning*, vol. 155 of *Proceedings of Machine Learning Research*, pp. 1918–1929. PMLR, 16–18 Nov 2021.
Available at: https://proceedings.mlr.press/v155/jain21b.html

[57] Williams, G., Drews, P., Goldfain, B., Rehg, J.M. and Theodorou, E.A.: Aggressive driving with model predictive path integral control. *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1433–1440, 2016.
Available at: https://doi.org/10.1109/ICRA.2016.7487277

[58] Liniger, A. and Lygeros, J.: Real-Time Control for Autonomous Racing Based on Viability Theory. *IEEE Transactions on Control Systems Technology*, vol. 27, no. 2, pp. 464–478, 2019. ISSN 1558-0865. 1701.08735.

[59] Brunner, M., Rosolia, U., Gonzales, J. and Borrelli, F.: Repetitive learning model predictive control: An autonomous racing example. *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, vol. 2018-Janua, no. Cdc, pp. 2545–2550, 2018.

[60] Osa, T., Pajarinen, J., Neumann, G., Bagnell, J.A., Abbeel, P. and Peters, J.: An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.
Available at: https://doi.org/10.1561/2300000053

[61] Xinlei Pan, Yurong You, Z.W. and Lu, C.: Virtual to real reinforcement learning for autonomous driving. In: Tae-Kyun Kim, Stefanos Zafeiriou, G.B. and Mikolajczyk, K. (eds.), *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 11.1–11.13. BMVA Press, September 2017. ISBN 1-901725-60-X.
Available at: https://dx.doi.org/10.5244/C.31.11

[62] Plaat, A.: *Deep Reinforcement Learning*. Springer Nature Singapore, 2022.
Available at: https://doi.org/10.48550/arXiv.2201.02135

[63] Novikov, A., Novikov, I. and Shevtsova, A.: Study of the impact of type and condition of the road surface on parameters of signalized intersection. *Transportation Research Procedia*, vol. 36, pp. 548–555, January 2018.
Available at: https://doi.org/10.1016/j.trpro.2018.12.154

[64] Cai, P., Mei, X., Tai, L., Sung, Y. and Liu, M.: High-speed autonomous drifting with deep reinforcement learning. April 2020.
Available at: https://doi.org/10.1109/LRA.2020.2967299

[65] Hafner, D., Lillicrap, T., Ba, J. and Norouzi, M.: Dream to control: Learning behaviors by latent imagination. 2019.
Available at: https://arxiv.org/abs/1912.01603

[66] Mysore, S., Mabsout, B., Mancuso, R. and Saenko, K.: Regularizing action policies for smooth control with reinforcement learning. 2021.
Available at: https://doi.org/10.48550/arXiv.2012.06644

[67] Betz, J., Wischnewski, A., Heilmeier, A., Nobis, F., Stahl, T., Hermansdorfer, L. and Lienkamp, M.: A software architecture for an autonomous racecar. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pp. 1–6. 2019.
Available at: https://doi.org/10.1109/VTCSpring.2019.8746367

[68] White, D.: Real applications of markov decision processes. *INFORMS*, vol. 15, pp. 73–83, 1985.
Available at: http://www.jstor.org/stable/25060766

[69] Sutton, R. and Barto, A.: *Reinforcement Learning, An Introduction*. 2nd edn. MIT Press, Cambridge, Massachusetts, 2020. ISBN 9780262039246.

[70] Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P. and Ba, J.: Benchmarking model-based reinforcement learning. 2019. 1907.02057.
Available at: https://doi.org/10.48550/arXiv.1907.02057

[71] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M.: Deterministic policy gradient algorithms. In: Xing, E.P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, vol. 32 of *Proceedings of Machine Learning Research*, pp. 387–395. PMLR, Bejing, China, 22–24 Jun 2014.
Available at: https://proceedings.mlr.press/v32/silver14.html

[72] Fujimoto, S., van Hoof, H. and Meger, D.: Addressing function approximation error in actor-critic methods. In: Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 1587–1596. PMLR, 10–15 Jul 2018.
Available at: https://proceedings.mlr.press/v80/fujimoto18a.html

[73] Goodfellow, I., Bengio, Y. and Courville, A.: *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[74] Ng, A. and Katanforoosh, K.: Lcs229 lecture notes in deep learning. July 2019. https://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf.

[75] Kingma, D.P. and Ba, J.: Adam: A method for stochastic optimization. 2014.
Available at: https://arxiv.org/abs/1412.6980

[76] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M.: Playing atari with deep reinforcement learning. 2013.
Available at: https://arxiv.org/abs/1312.5602

[77] Althoff, M. and Würsching, G.: CommonRoad: Vehicle Models. *Technische Universität München*, 2020.
Available at: https://gitlab.lrz.de/tum-cps/commonroad-vehicle-models

[78] Sakai, A., Ingram, D., Dinius, J., Chawla, K., Raffin, A. and Paques, A.: Pythonrobotics: a python code collection of robotics algorithms. 2018.
Available at: https://arxiv.org/abs/1808.10703

[79] Li, M., Wang, Y., Zhou, Z. and Yin, C.: Sampling rate selection for trajectory tracking control of autonomous vehicles. In: *2019 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pp. 1–5. 2019.
Available at: https://doi.org/10.1109/VPPC46532.2019.8952506

[80] Vorotic, G., Rakicevic, B. and Mitic, Sasa Stamenkovic, D.: Determination of cornering stiffness through integration of a mathematical model and real vehicle exploitation parameters. *FME Transactions*, vol. 41, pp. 66–71, 2013.
Available at: https://www.mas.bg.ac.rs/_media/istrazivanje/fme/vol41/1/08_gvorotovic.pdf

[81] Keefer, E., Bryan, W. and Bevly, D.: International conference for robotics and automation. 2022.

[82] Wang, R., Han, Y. and Vaidya, U.: Deep koopman data-driven optimal control framework for autonomous racing deep koopman data-driven optimal control framework for autonomous racing. *Early access*, pp. 1–6, 2021.
Available at: https://linklab-uva.github.io/icra-autonomous-racing/contributed_papers/paper12.pdf

[83] Patnaik, A., Patel, M., Mohta, V., Shah, H., Agrawal, S., Rathore, A., Malik, R., Chakravarty, D. and Bhattacharya, R.: Design and implementation of path trackers for ackermann drive based vehicles. 2020.
Available at: https://arxiv.org/abs/2012.02978

[84] Rajamani, R.: *Vehicle Dynamics and Control*. Springer, Minneapolis, 1988.
Available at: https://doi.org/10.1007/978-1-4614-1433-9