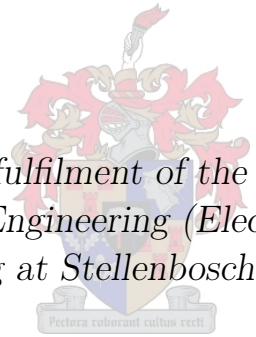
Partial end-to-end reinforcement learning for robustness towards model-mismatch in autonomous racing

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 .....

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

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, enabling the 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.

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, we show the process of determining the optimal hyper-parameters for the partial end-to-end algorithm to ensure the system operates at its peak performance. 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, we conducted a comparative analysis, contrasting our chosen partial end-to-end architecture with alternative variations of the partial end-to-end architecture. These variations include architectures with either solely a velocity controller or a steering controller.

1.1 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 1.1.

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 **??**, the observation space of our 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 maintains the desired velocity. Furthermore, the velocity constraint component, as introduced in Equation **??**, limits the desired acceleration so that the vehicle operates within safe speed

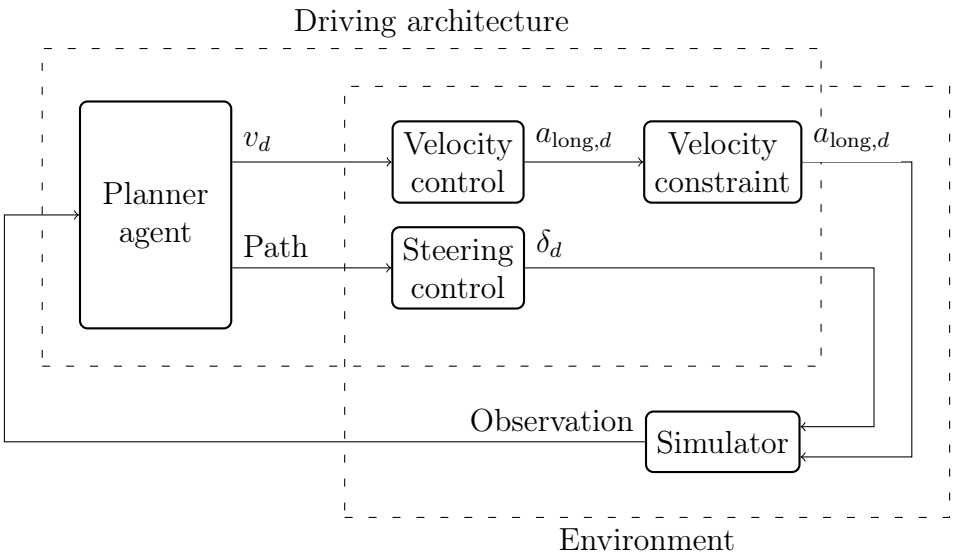


Figure 1.1: The partial end-to-end racing algorithm, which consists of an RL agent that outputs a plan comprised of a path (i.e, a series of x and y coordinates) and desired velocity, a steering and a velocity controller, as well as a velocity constraint. δ_d, v_d, and $a_{\text{long},d}$ denote the desired steering angle, longitudinal velocity and longitudinal acceleration respectively.

limits. The steering and velocity controllers, as well as velocity constraint operate at the 100 Hz sample rate of the simulator introduced in Section **??**. Furthermore, it is important to note that these components are treated as part of the environment to conform to the definition of the MDP.

1.1.1 Planner agent

The partial end-to-end planner agent, which is built using a DNN, is presented in Figure 1.2. 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.

1.1.2 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. [1] utilize bezier curves, where the control points of the curves correspond to the output of the neural network. Similarly, Capo et al. [2] predict the offset of a single point ahead of the vehicle in relation to the track centerline using the neural network's output.

On the other hand, classical approaches such as [3; 4; 5] utilize multiple motion primitives generated by forward simulating the vehicle dynamics to construct a path. However,

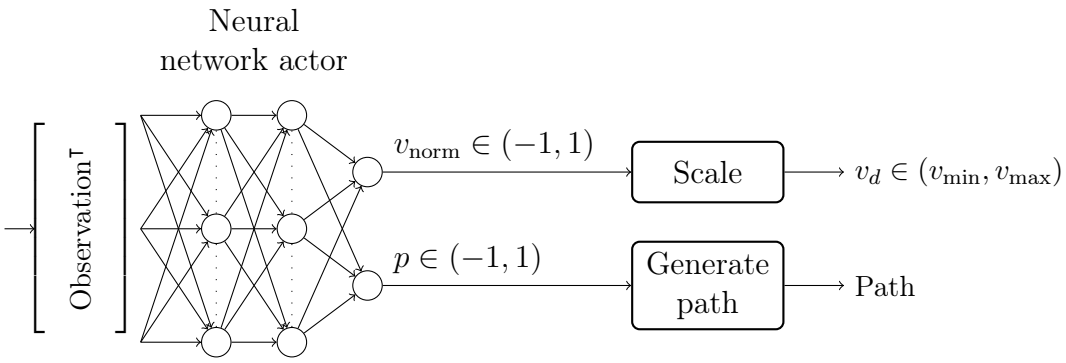


Figure 1.2: The partial end-to-end agent. The outputs of the DNN are initially both in the range $(-1, 1)$. The output denoted v_{norm} corresponds to the desired longitudinal velocity and is scaled to the range $(v_{\min}, v_{\max})$. The other output, denoted p, is used to construct the path.

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 generate a path using the DNN output p in the Frenet frame [6]. 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 1.3 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 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

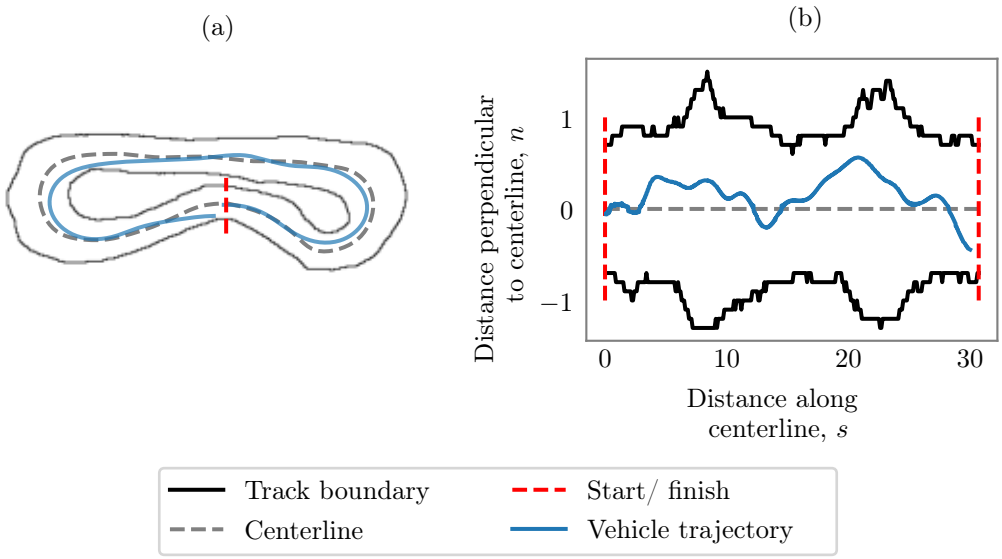


Figure 1.3: An example trajectory of an agent racing anti-clockwise around Porto, in (a) Cartesian coordinates, and (b) Frenet coordinates.

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 1.4 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 corresponding Cartesian coordinate of s_0 from the vehicle heading.

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

$$f(s) = As^3 + Bs^2 + Cs + D, \tag{1.1}$$

   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 $f'(s_0) = \tan(\psi_0)$.

   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 $f'(s_1) = 0$.

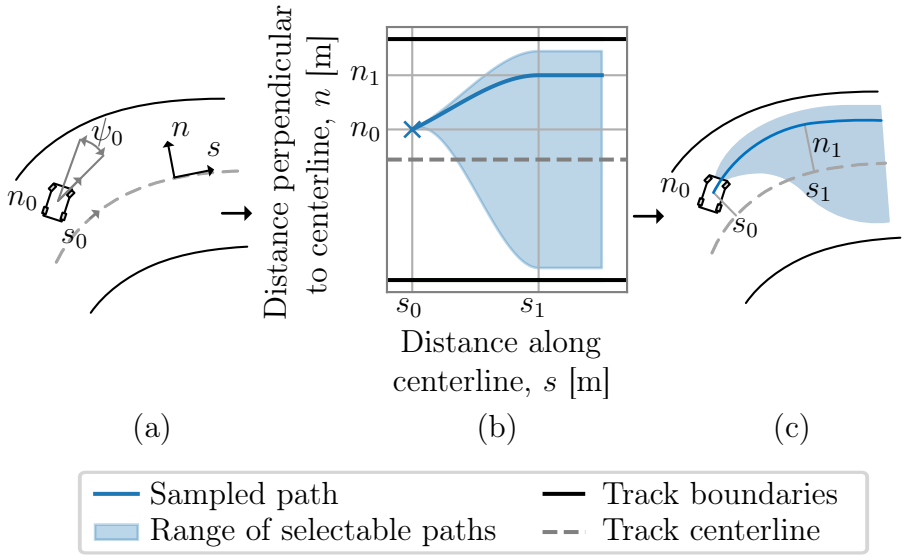


Figure 1.4: An illustration of the process of generating the polynomial path in the Frenet frame. (a) The vehicle coordinates are converted into the Frenet frame, then (b) a path is constructed within the Frenet frame, after which (c) the path is converted into Cartesian coordinates.

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.

1.1.3 Steering controller design

The pure pursuit steering controller is popular amongst partial end-to-end methods [7; 8]. 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. [9].

This controller facilitates the steering of the vehicle towards a designated *target point* on the planned path, as depicted in Figure 1.5. The target point is determined by a specified *look-ahead* distance, denoted as l_d, which is calculated using

$$l_d = k_s \cdot v + L_c, \tag{1.2}$$

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. [10] 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.

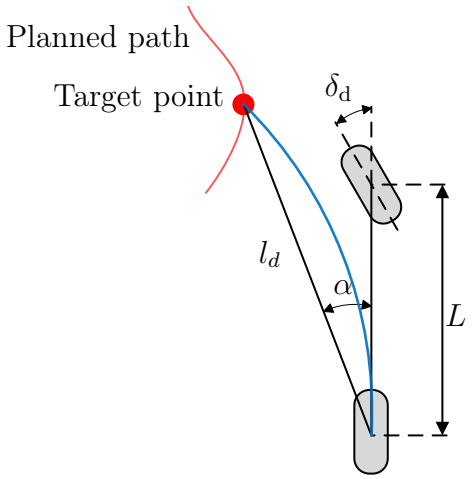


Figure 1.5: A depiction of the pure pursuit controller, which steers the vehicle towards a target point on the path. The symbols l_d, L, α and δ_d represent the look-ahead distance, wheelbase, angle between vehicle's heading and look-ahead distance vector, and desired steering angle respectively. Furthermore, the blue line represents the path the rear wheels travel to reach the target point.

The desired steering angle δ_d is then computed as

$$\delta_d = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right), \tag{1.3}$$

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 1.2 and 1.3 at a rate of 100 Hz.

1.1.4 Velocity controller design

The velocity controller is implemented in a similar manner to the proportional controller integrated into the official F1tenth simulator [11]. It takes the current vehicle velocity and the desired velocity v_d, determined by the planner agent, as inputs. It then calculates the desired longitudinal acceleration $a_{\text{long},d}$ using proportional control according to the following conditions:

$$a_{\text{long},d} = \begin{cases} k_v \frac{a_{\max}}{v_{\max}}(v_d - v) & \text{if } v_d \geq v \\ k_v \frac{a_{\max}}{v_{\min}}(v_d - v) & \text{if } v_d < v. \end{cases} \tag{1.4}$$

Here, k_v represents the controller gain, and $a_{\max}$ is the maximum longitudinal acceleration, Importantly, at this layer of abstraction, it is assumed that the vehicle has a lower-level controller that accurately tracks acceleration commands [12; 13]. Furthermore, the velocity controller is sampled at a rate of 100 Hz.

1.2 Applying TD3 to partial end-to-end racing

To enable training of the partial end-to-end agent using TD3, we followed similar steps as for the end-to-end agent, which are detailed in Section **??**. Specifically, we incorporated the following modifications to Algorithm **??**: Line 7, in which an action is sampled form the agent, is changed to

$$a_t = [v_d, \text{path}] \leftarrow \text{scale}(\pi_\phi(o_t + \epsilon)), \quad \epsilon \sim \mathcal{N}(0, \sigma_{\text{action}}). \tag{1.5}$$

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

Lines 8 to 14 of Algorithm **??** 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 1.3 and 1.4, respectively. The velocity contstraint component then limits the velocity using Equation **??**.

The remaining components required to implement the algorithm are a critic and a 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 **??**, 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 **??** was adapted to evaluate partial end-to-end agents. This was done by substituting Equation 1.5 into line 4, which samples an action from the agent. Additionally, Equations **??** and **??** 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, while Gaussian noise was introduced to the 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.

1.3 Empirical design and hyper-parameter values

The partial end-to-end algorithm, as well as modifications to TD3 have now 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 1.1.

To determine these hyperparameters, the same procedure as with 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 1.2 with various values of the hyper-parameter under consideration, while keeping all of the other hyper-parameters fixed at the values listed in Table 1.1. 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 **??**, 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 1.6 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 all agents 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 end-to-end agent. The agent sampling rate was chosen conservatively as 10 Hz.

1.4 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 1.2), 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 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. [10] 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.

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

The paths taken by the vehicles in this experiment are visually represented in Figure 1.7. 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 with look-ahead constants of 0.5, 1 and 1.5 meters, while setting the remaining hyper-parameters equal to those listed in Table 1.1. The average lap time, percentage failed laps and average cumulative episode reward during training for three agents trained with each look-ahead distance are in Figure 1.8.

From Figure 1.8, 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

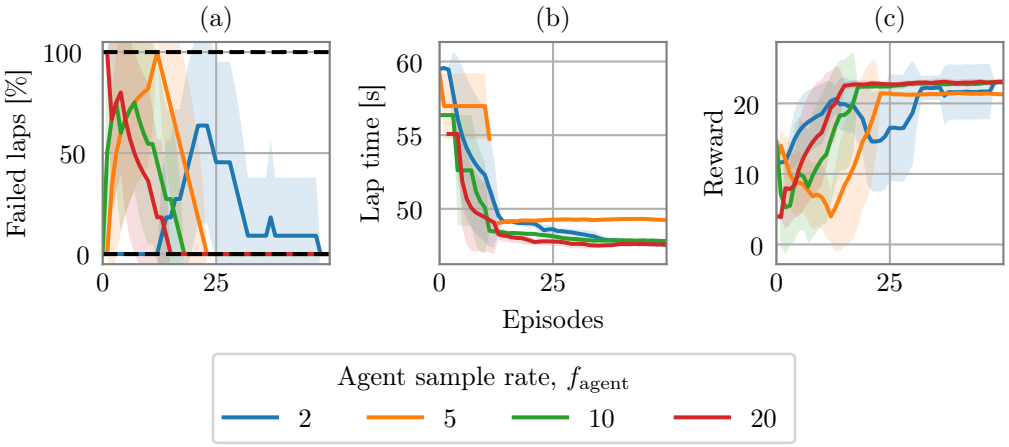


Figure 1.6: (a) The percentage failed laps and (b) average lap time of completed laps during training, as well as (c) the average learning curves of three partial end-to-end agents utilising agent sampling rate (f_{agent}) values ranging from 2 Hz to 20 Hz.

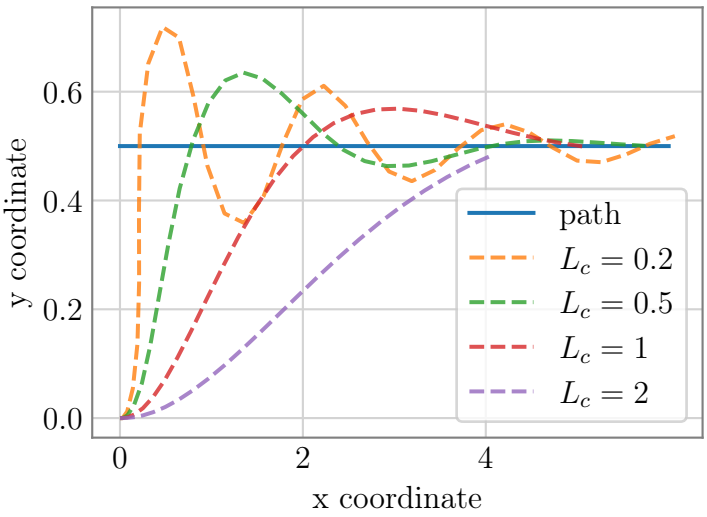


Figure 1.7: Trajectories taken by vehicles utilising a pure pursuit steering controller following a the straight blue line.

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 trained 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.

We then compared the paths driven 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 1.9, which shows that although partial end-to-end agents utilising a pure pursuit controller may complete laps quickly, the tracking performance of this controller in high speed conditions is poor. 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

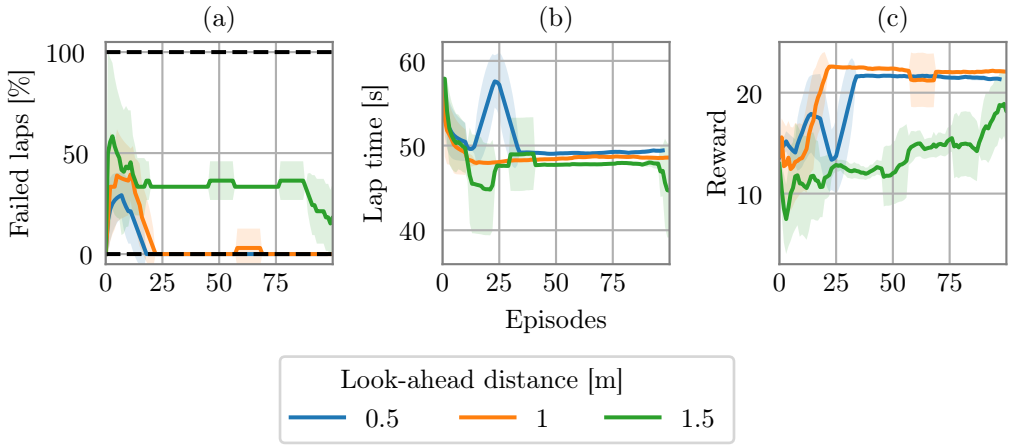


Figure 1.8: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of partial end-to-end agents with different pure pursuit look-ahead distances racing on Circuit de Barcelona-Catalunya.

performance. This is attributed to the fact that the agent is trained with the controllers in place.

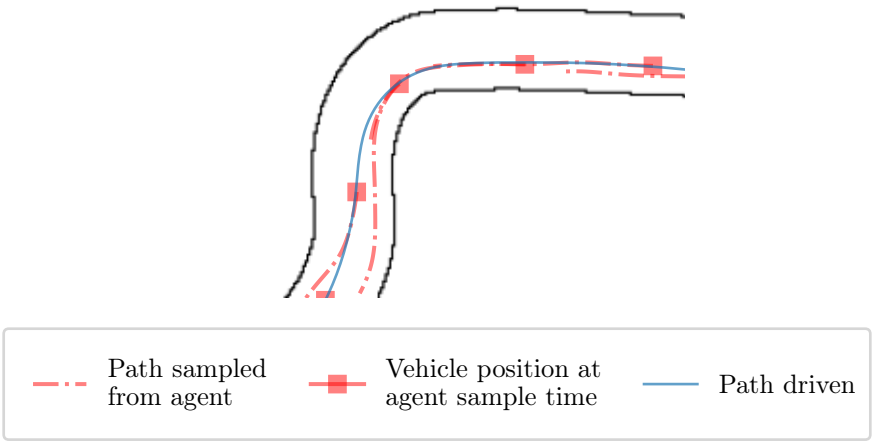


Figure 1.9: The path driven by a partial end-to-end agent on a section of Circuit de Barcelona-Catalunya, along with planned paths sampled from the agent.

1.5 Velocity controller tuning

Next, the velocity controller gain k_v (found in Equation 1.4) was tuned experimentally. Agents with k_v values of 0.5, 1 and 2 were trained to race on Barcelona-Catalunya. 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 1.2. From this table, we observe that each agents completed all of its laps, and that the differences in lap time between agents are minimal.

A qualitative evaluation of the trajectories taken by agents utilising each velocity controller gain was therefore conducted. The paths followed by agents employing differ-

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

Table 1.2: Results from agents racing under evaluation conditions using different values for the velocity controller gain (k).

ent controller gains on the final section of Barcelona-Catalunya is shown in Figure 1.10. Agents utilizing controller gains of 0.5 and 1 exhibit similar trajectories, while those using a controller gain of 2 tend to take wider turns. 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.

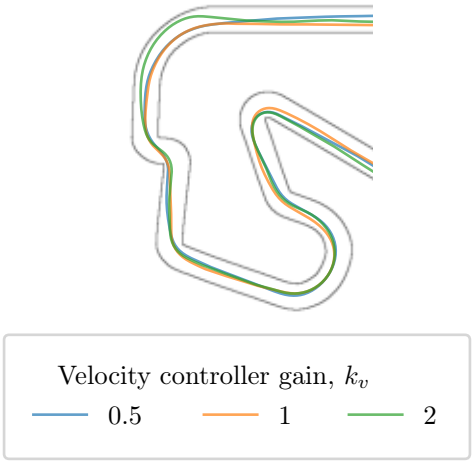


Figure 1.10: Paths taken by partial end-to-end agents utilising controller gain (k_v) values of 0.5, 1 and 2 on the final section of Barcelona-Catalunya.

1.6 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 1.11 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 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.

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

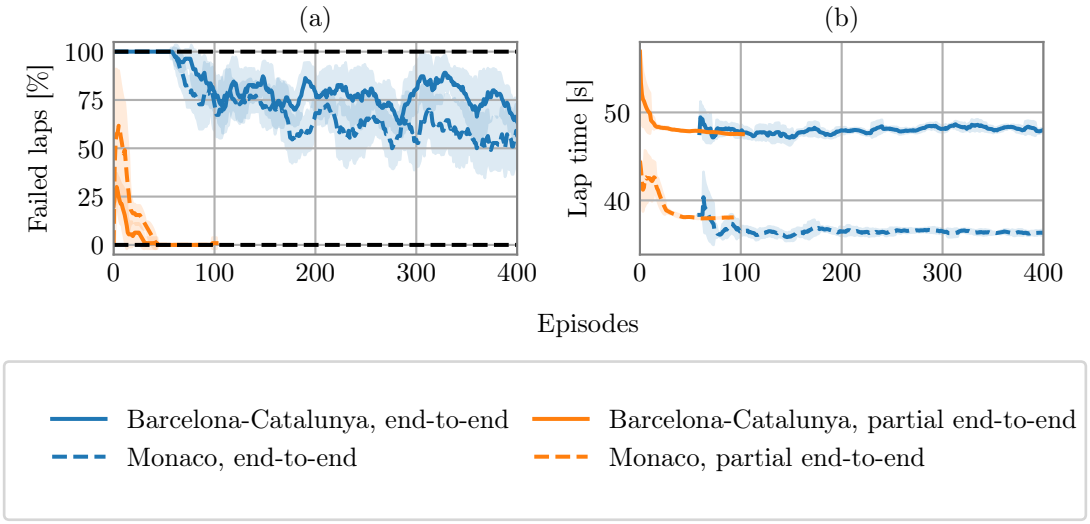


Figure 1.11: (a) Percentage failed laps and (b) lap time of partial and fully end-to-end agents during training. Dashed lines indicate agents trained to race on Barcelona-Catalunya, while solid lines indicate agents trained to race on Monaco.

not intersect with the track boundary. This is exemplified in Figure 1.12, 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.

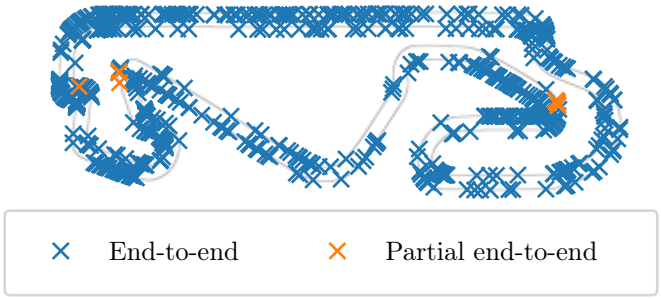


Figure 1.12: Locations where the end-to-end and partial end-to-end agents crashed during training.

The toal 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 1.3. 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 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 1.3: Performance of end-to-end and partial end-to-end agents racing on all three tracks under evaluation conditions.

While Table 1.3 provides the average results across a set of 10 agents, a more comprehensive depiction of the distribution of agents' performances is presented in Figure 1.13. 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.

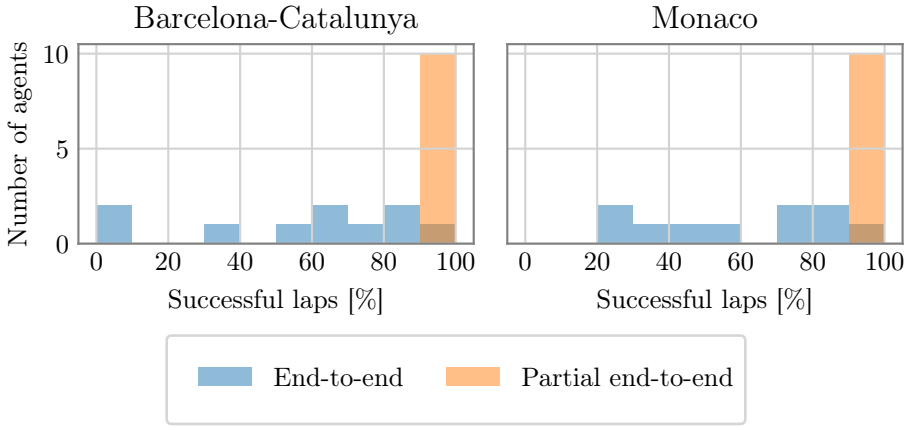


Figure 1.13: Distribution of percentage successful laps completed by agents under evaluation conditions for the Barcelona-Catalunya and Monaco tracks.

Figures 1.14 and 1.15 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

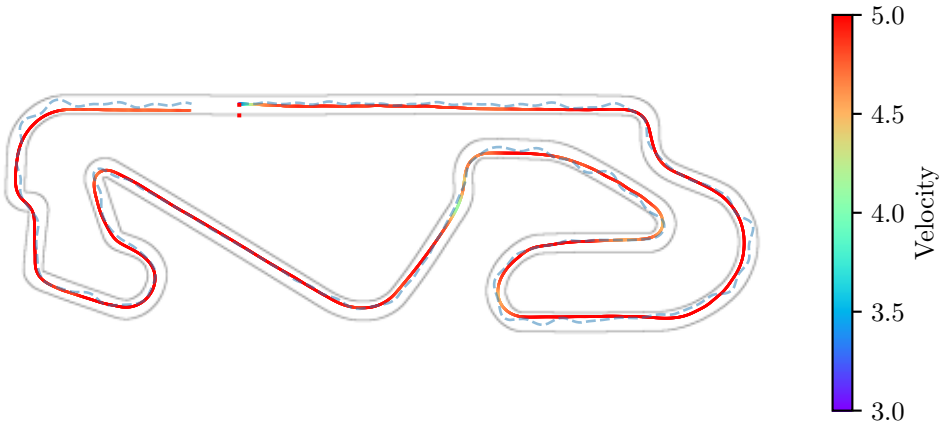


Figure 1.14: The path and velocity profile taken by a partial end-to-end agent completing Circuit de Barcelona-Catalunya. For comparison, the path of an end-to-end agent racing on the same track is depicted with a dashed light blue line.

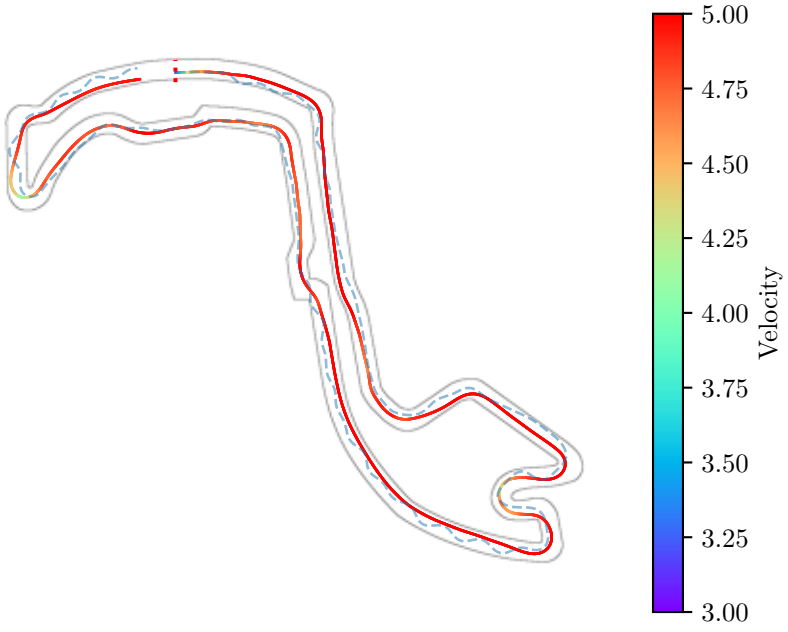


Figure 1.15: The path and velocity profile taken by a partial end-to-end agent completing Circuit de Monaco. For comparison, the path of an end-to-end agent racing on the same track is depicted with a dashed light blue line.

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.

In Chapter **??**, 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 1.16. 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.

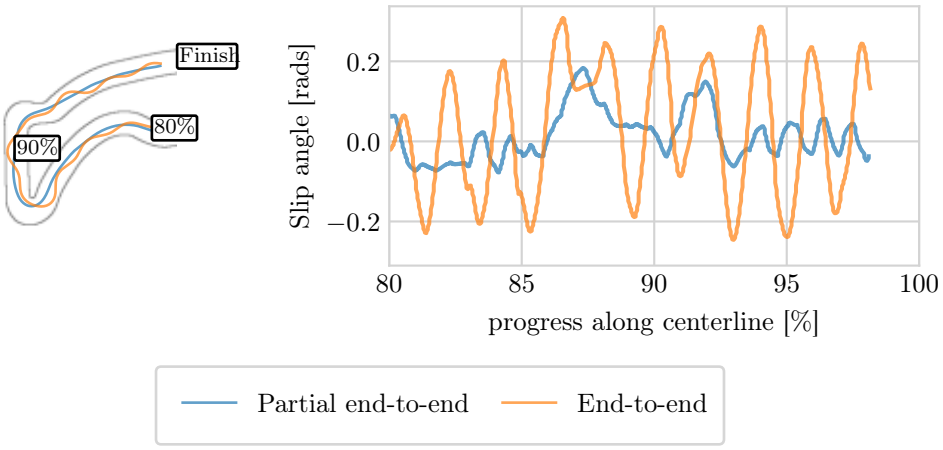


Figure 1.16: The paths and slip angles of agents racing on the final section of Circuit de Monaco.

1.7 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 1.1. Furthermore, three agents were trained utilising each proposed architecture. Figure 1.17 presents the percentage of failed laps and average lap time during training, as well as the learning curves for agents utilising each architecture. 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

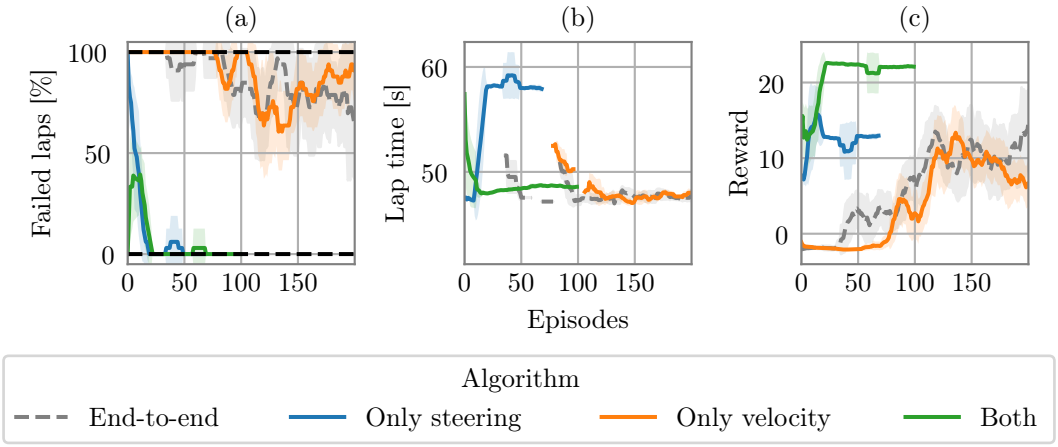


Figure 1.17: (a) The percentage failed laps and (b) lap time during training, as well as (c) the learning curves for agents utilising each algorithm structure.

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.

As part of the ablation study, we qualitatively evaluated the trajectories executed by each proposed architecture on Barcelona-Catalunya. Figure 1.18 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

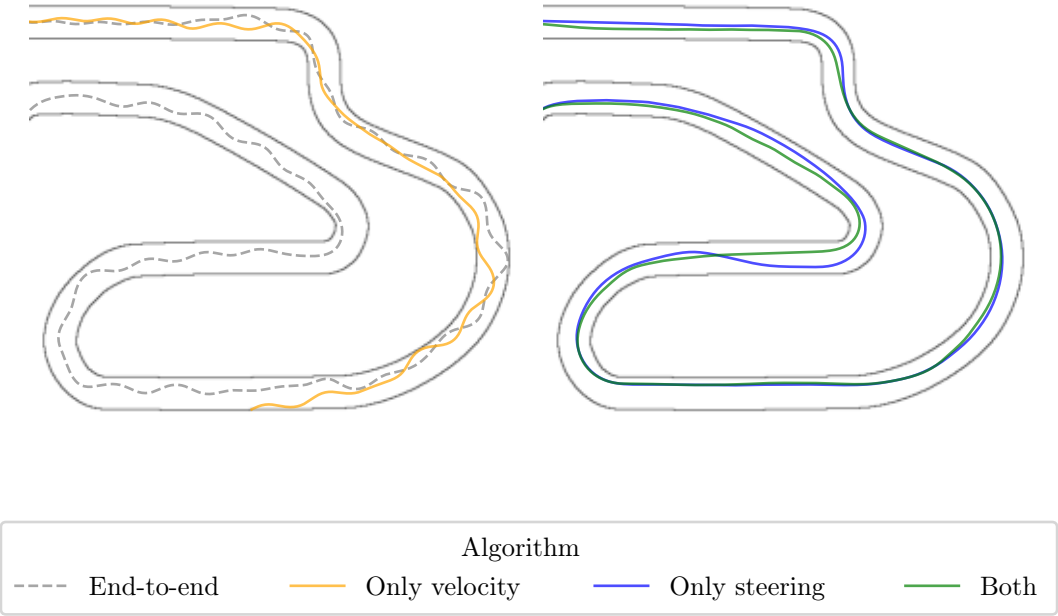


Figure 1.18: The paths taken by agents utilising each algorithm architecture. Paths taken by agents without a steering controller are depicted on the right, whereas the paths taken by agents utilising a steering controller are depicted on the left.

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.

1.8 Summary

In this chapter, we have detailed the design of our partial end-to-end algorithm, which is comprised of an RL planner, a pure pursuit steering controller and a proportional velocity controller. The RL planner agent utilises the Frenet frame to output paths that do not intersect with the track boundary.

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, compared to end-to-end agents. 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. They also exhibit improved performance over end-to-end agents under evaluation conditions, executing 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.

Appendices

Appendix A

Supporting results

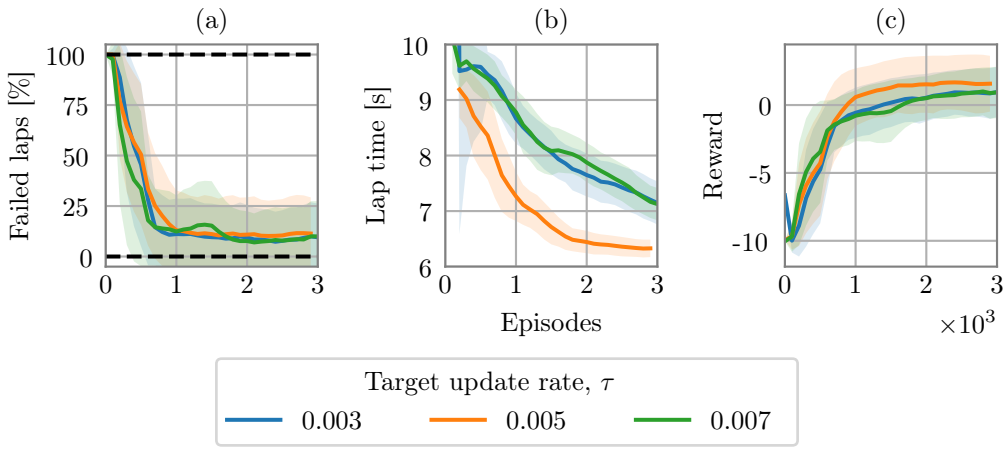


Figure A.1: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with target update rates ranging from 0.003 to 0.007.

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 $7 \cdot 10^{-3}$.

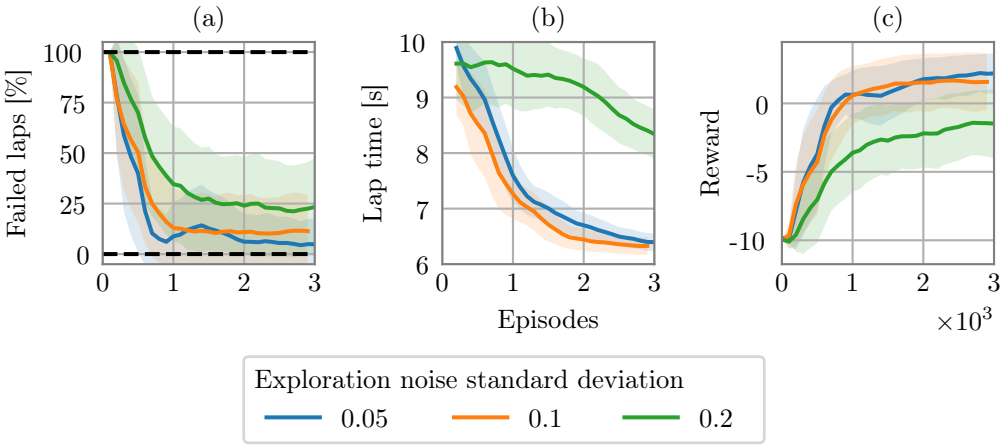


Figure A.2: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with exploration noise standard deviations ranging from 0.05 to 0.2.

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.

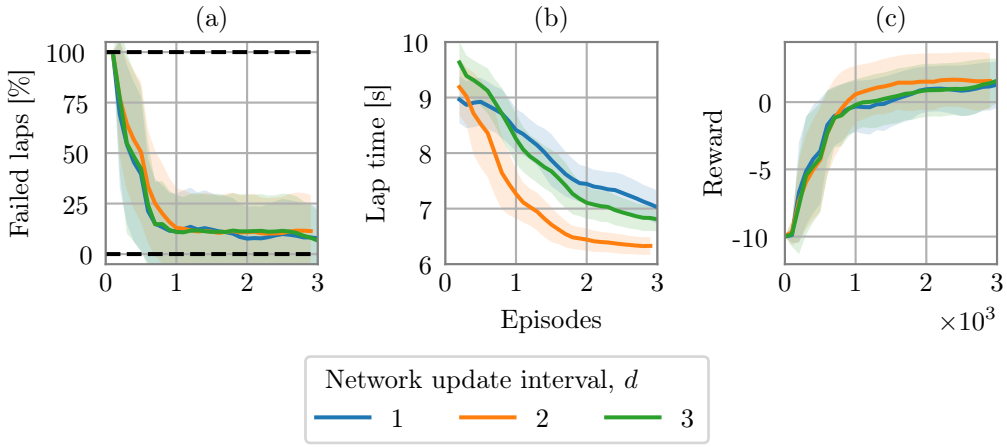


Figure A.3: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with network update intervals d ranging from 1 to 3.

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

[2] 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

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

[4] 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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

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

[12] 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`

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