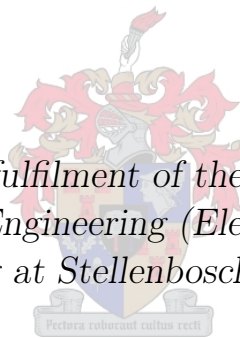# Partial end-to-end reinforcement learning for robustness towards model-mismatch in autonomous racing

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.

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
$\boldsymbol{x}^{\mathsf{T}}$	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 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 steering controller is used to generate steering commands that track the path. Meanwhile, a velocity controller generates desired acceleration commands, denoted $a_{\mathrm{long},d}$, 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 limits. The steering and velocity controllers, as well as
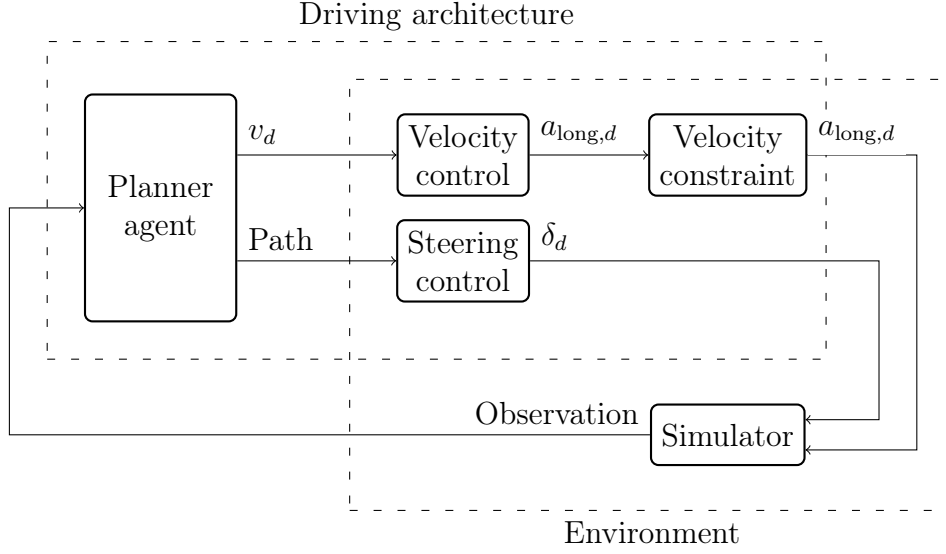
Driving architecture

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.

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 neural network 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 $(v_{\text{min}}, v_{\text{max}})$ 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, 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.
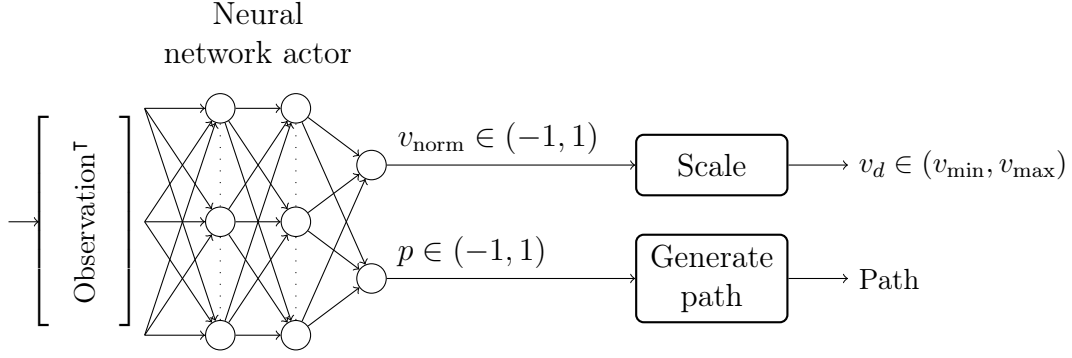
Neural
network actor

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.

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.

An example trajectory of an agent racing around the Porto track is illustrated in Figure 1.3. The trajectory is presented 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, in Frenet coordinates, it becomes easier to create paths that avoid intersecting with the track boundaries 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.
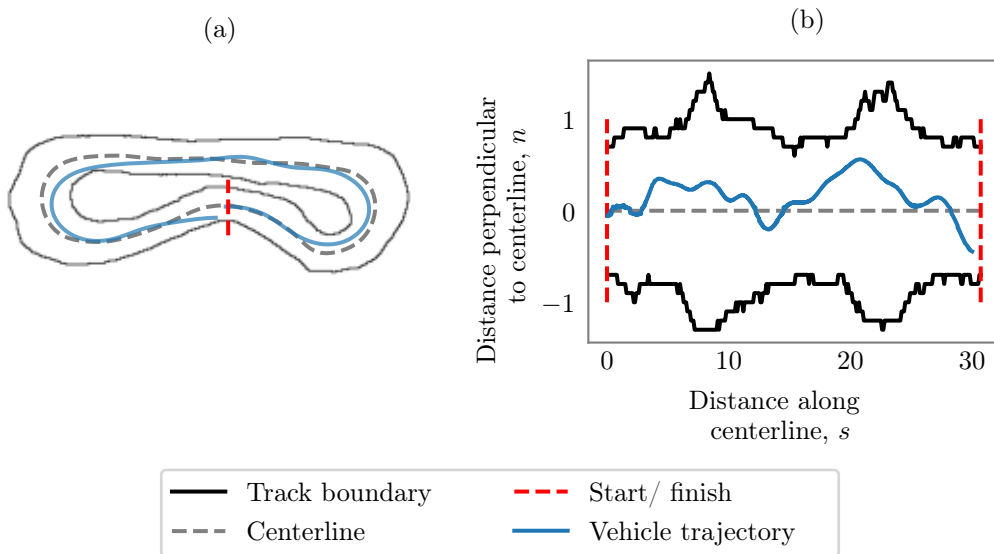

Figure 1.3: An example of a trajectory in (a) Cartesian coordinates, and (b) the equivalent trajectory in Frenet coordinates.

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 $f'(s_1) = 0$.

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

A steering controller takes a desired path and vehicle pose as input, and outputs steering commands such that the vehicle tracks a planned path. 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 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

$$l_d = k_s \cdot v + L_c, \tag{1.2}$$
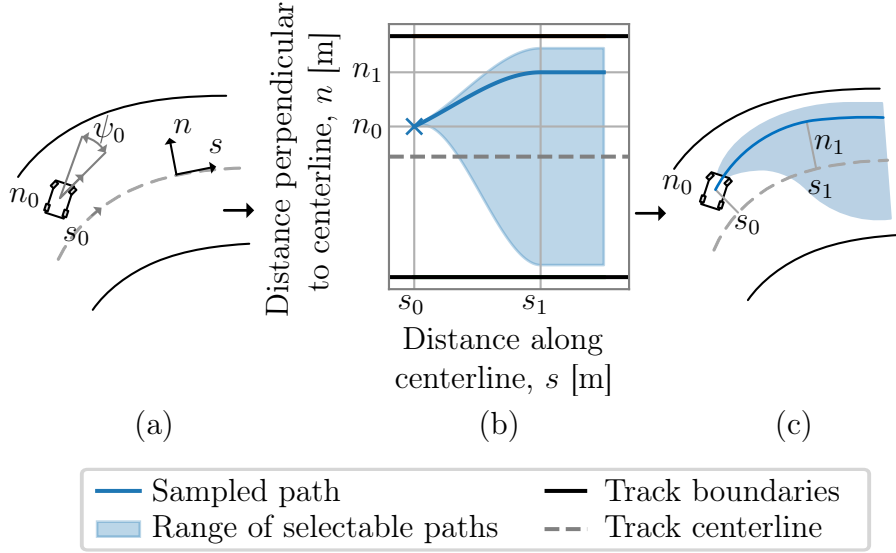
Figure 1.4: An illustration of the process of generating the polynomial path in the Frenet frame. (a) The vehicle coordinates are converted into the Frenet frame, then (b) a path is constructed within the Frenet frame, after which (c) the path is converted into Cartesian coordinates.

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, both l_d and L_c are measured from the center of the rear axle.
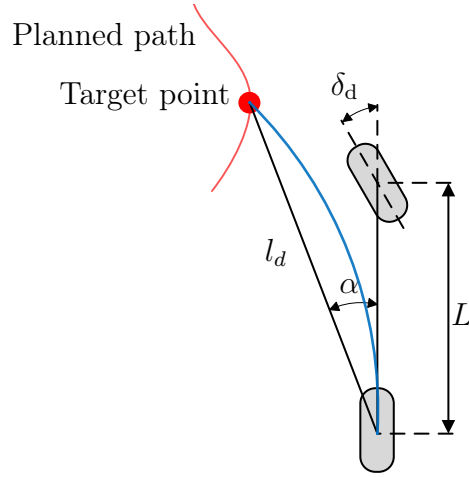
Figure 1.5: A depiction of the pure pursuit controller, which steers the vehicle towards a target point on the path. The symbols l_d, L, α and δ_d represent the look-ahead distance, wheelbase, angle between vehicle's heading and look-ahead distance vector, and desired steering angle respectively. Furthermore, 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, assuming 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 $a_{\mathrm{long},d}$ using proportional control according to the following conditions:

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

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

Having described the components of the partial end-to-end framework, we now discuss how the TD3 algorithm from Section **??** was applied to train the agent to race. To enable training of the partial end-to-end agent using TD3, we followed the steps as for the end-to-end agent detailed in Section **??**. Specifically, we worked from Algorithm **??** which was applied to the end-to-end agent, and incorporated the following modifications:

In line 7 of Algorithm **??**, an action is sampled. In the case of the end-to-end agent, this action comprised a longitudinal acceleration $a_{\mathrm{long},d}$ and steering angle δ_d. However, the action sampled from the partial end-to-end planner agent is comprised of a desired velocity and path. Line 7 was therefore changed to

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

In Algorithm **??**, the difference in sample rate between the agent and environment components are addressed by performing N environment samples for every MDP time step. The value of N is determined by Equation **??**. To implement this, a for loop from lines 8 to 14 is employed.

For the partial end-to-end agent, the steering and velocity controllers are included in the definition of the environment in addition to the velocity constraint and simulator. We therefore sample an acceleration action from the velocity controller between lines 8 and 9 of the for loop, based on the current observation o_t and the desired velocity v_d:

$$a_{\mathrm{long},d} \leftarrow \mathrm{velocityControl}(o_t, v_d). \tag{1.6}$$

Additionally, the pure pursuit steering controller is sampled to produce a steering angle, utilizing the observation o_t and path selected by the agent:

$$\delta_d \leftarrow \mathrm{steeringControl}(o_t, \mathrm{path}). \tag{1.7}$$

The incorporation of the steering and velocity controllers into the for loop is facilitated by the fact that they are sampled at the same rate as the other components of the environment.

Having made these modifications to the TD3 algorithm, we developed the remaining components required to implement the algorithm, namely, a critic and a reward signal. We adopted the same critic DNN architecture as the one used for the end-to-end agent, as described in Section **??**. The critic accepts a normalized observation and action as input, and is composed of three fully connected layers: an input layer with 400 neurons, a hidden layer with 300 neurons, and an output layer with a single neuron. The first two layers utilize the ReLU activation function, while the final layer employs a linear activation function to produce the output.

Lastly, we adopted the same reward signal as used for the end-to-end agent, as described by Equation **??**. This reward signal incentivizes the agent for making progress along the centerline, while penalizing the agent on each time step as well as for collisions.

After implementing these changes to the TD3 algorithm, we adapted Algorithm **??**, which is used to evaluate trained end-to-end agents, for compatibility with partial end-to-end agents. To allow the algorithm to be used to evaluate partial end-to-end agents, we substituted Equation 1.5 into line 4. Additionally, Equations 1.6 and 1.7, which are used to sample control actions, were added in-between lines 2 and 3. By incorporating these adjustments, we were able to evaluate both end-to-end and partial end-to-end agents 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 agnet, 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

We have introduced the partial end-to-end algorithm and modifications to TD3 with symbolic hyper-parameter values. As with the end-to-end algorithm, these values were tuned experimentally for optimal performance. We followed the same procedure as with the end-to-end agent to select hyper-parameter values, with the addition of hyper-parameters associated with the steering and velocity controllers. This procedure involved repeatedly training agents using Algorithm **??** with specific values of the hyper-parameter under consideration while keeping all other hyper-parameters fixed. Furthermore, three agents were trained for every hyper-parameter set to ensure consistency in the results. The values that were experimentally determined as optimal for all three tracks are listed in Table 1.1.

Since the hyper-parameter tuning procedure for the end-to-end agent was discussed in detail in Chapter **??**, we will not present the complete procedure for parameters that were already shown. However, we do demonstrate the process of tuning the agent sampling rate (f_{agent}) as a representative sample of the hyper-parameter tuning procedure.

Figure 1.6 illustrates the results obtained during training from agents racing 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

Hyper-parameter	Symbol	Value
Algorithm	$5 \cdot 10^4$	
Maximum time steps	M	
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}	0.2
Time step penalty	r_{time}	0.01
Collision penalty	$r_{\text{collision}}$	-5
Observation		
Number of LiDAR beams	L	20
Neural network		
Learning rate	α	10^{-3}
Input layer size	m_1	400
Hidden layer size	m_2	300
Velocity constraints		
Minimum velocity	v_{min}	3 m/s
Maximum velocity	v_{max}	5 m/s
Velocity controller		
Gain	k_v	0.5
Steering controller		
Look-ahead gain	k_s	0.1
Look-ahead constant	L_c	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.

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 conservatively chosen as 10 Hz.

We now present the hyper-parameter tuning procedure for components that are unique to the partial end-to-end agent. These are the steering and velocity controllers.

1.4   Steering controller tuning

To determine the optimal values for the pure pursuit steering controller's look-ahead gain (k_s) and constant (L_c), we conducted a series of experimental evaluations. Initially, we focused 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 path was predefined as a straight line, and actions
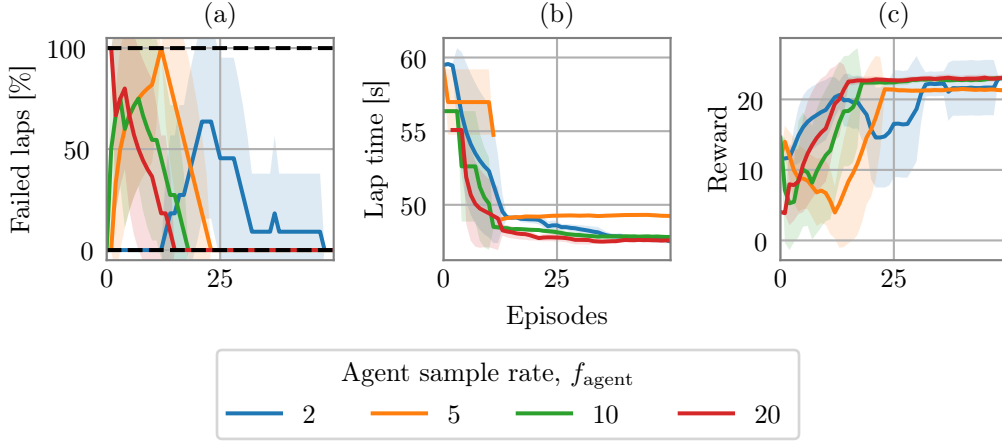
Figure 1.6: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of partial end-to-end agents utilising agent sampling rate (f_{agent}) values ranging from 2 Hz to 20 Hz.

were exclusively sampled 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.

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.
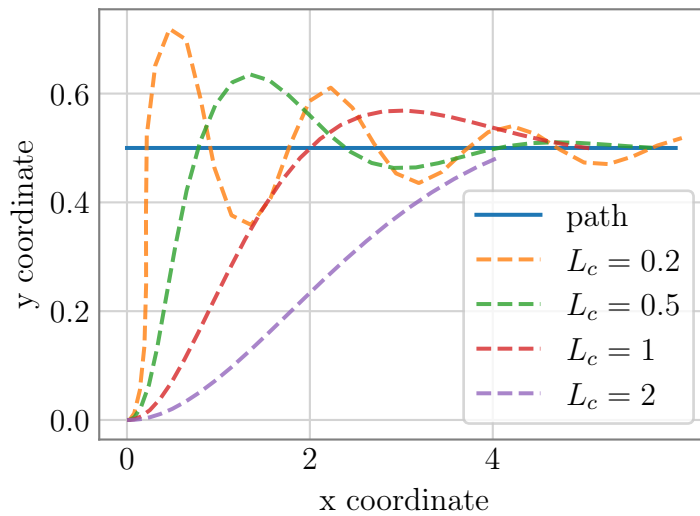

Figure 1.7: Trajectories taken by vehicles utilising a pure pursuit steering controller following a the straight blue line.

Based on these findings, we trained agents with look-ahead constants of 0.5, 1 and 1.5 meters, while setting the look-ahead gain to 0.1. The lap time, percentage failed laps and cumulative episode reward during training of these agents are shown in Figure 1.8.
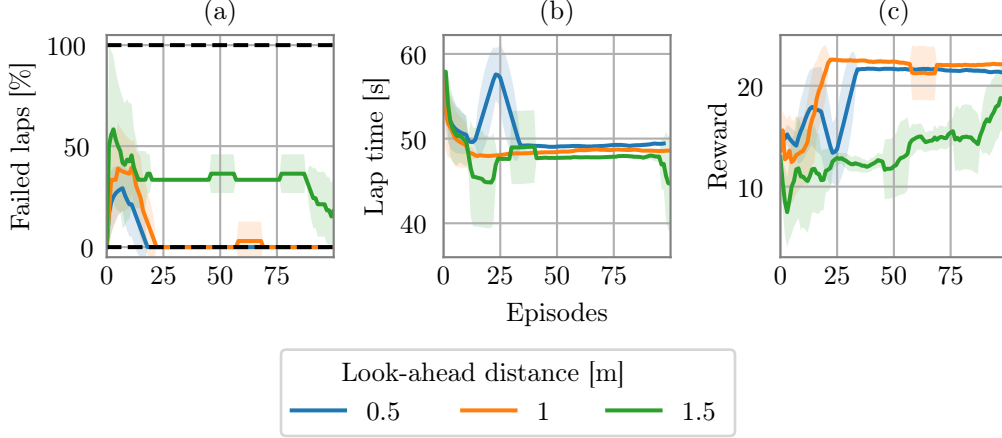

Figure 1.8: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of partial end-to-end agents with different pure pursuit look-ahead distances racing on Circuit de Barcelona-Catalunya.

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 trained with look-ahead constants of 0.5 and 1 do. 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, achieving the highest reward. We therefore selected the value for the look-ahead constant L_c as 1.

The paths driven by a partial end-to-end agent trained on Barcelona-Catalunya with its look-ahead constant set to 1 is shown in Figure 1.9, along with the planned paths
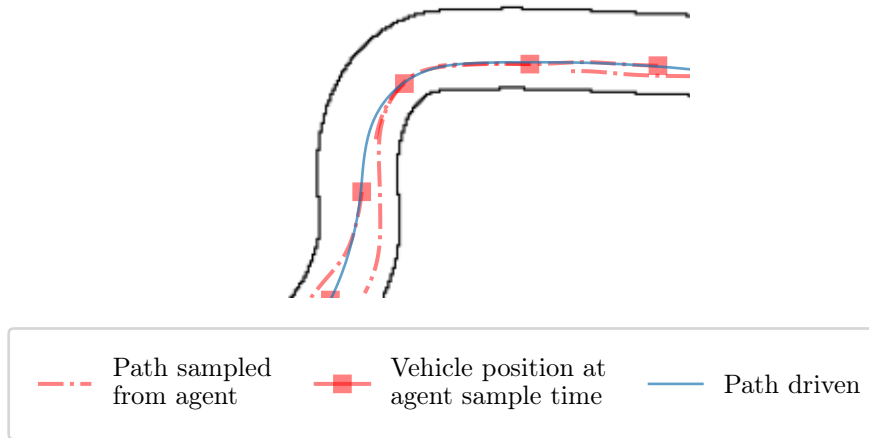

Figure 1.9: The path driven by a partial end-to-end agent on a section of Circuit de Barcelona-Catalunya, along with planned paths sampled from the agent.

sampled from the agent. The figure 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 performance. This is attributed to the fact that the agent is trained with the controllers in place.

1.5   Velocity controller tuning

Next, the velocity controller gain k_v 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 lap time that these agents achieved under evaluation conditions are given in Table 1.2. From this table, we observe that all of the agents completed all of their laps under evaluation conditions, and that the differences in lap time 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 1.2: Results from agents racing under evaluation conditions using different values for the velocity controller gain (k).

Therefore, we conducted a qualitative evaluation of the trajectories taken by agents utilising each velocity controller gain. The paths followed by agents employing different 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 conservative controller gain of 0.5 to prioritize vehicle safety.

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 on the Barcelona-Catalunya and Monaco tracks. We compared the training performance of both algorithms before assessing the performance of both agents under the specified evaluation conditions.

Figure 1.11 presents the training performance of both the partial and fully end-to-end algorithms on the Barcelona-Catalunya and Monaco tracks. The performance is evaluated in terms of the percentage of failed laps and lap time. A consistent trend is observed across all tested tracks: the partial end-to-end agents achieve either a 0% failure rate or come close to it, while the end-to-end agents continue to experience crashes throughout the training process. Furthermore, partial end-to-end agents train faster on average than
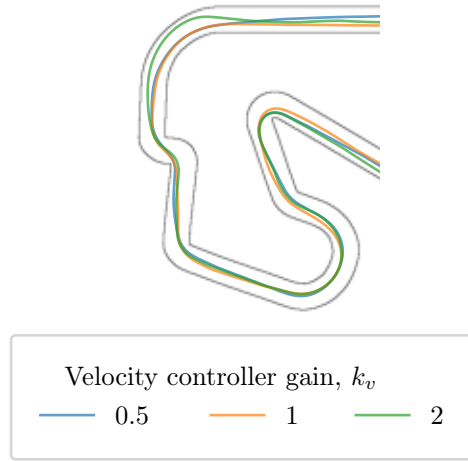
Figure 1.10: Paths taken by partial end-to-end agents utilising controller gain (k_v) values of 0.5, 1 and 2 on the final section of Barcelona-Catalunya.
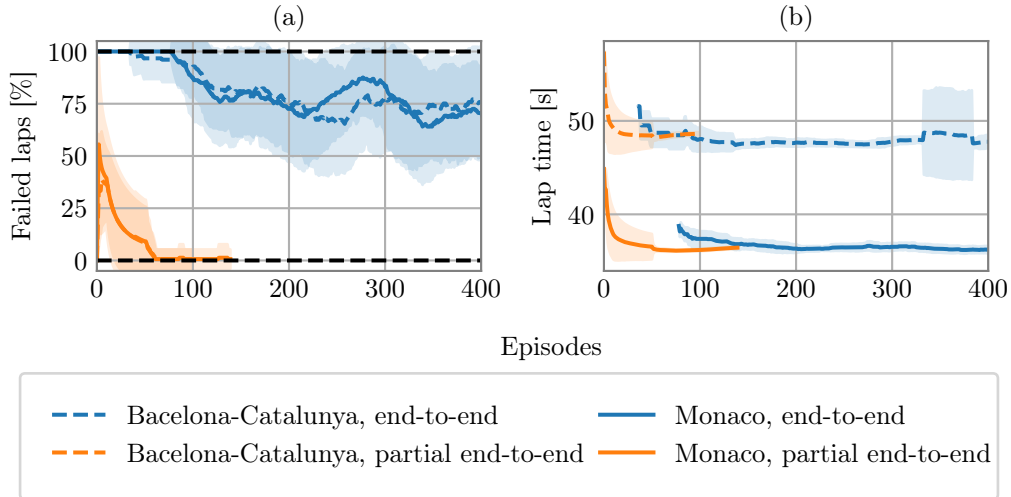

Figure 1.11: (a) Percentage failed laps and (b) lap time of partial and fully end-to-end agents during training. Dashed lines indicate agents trained to race on Barcelona-Catalunya, while solid lines indicate agents trained to race on Monaco.

end-to-end agents. 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 findings, utilising a trajectory planning approach that avoids track boundaries and executing it using a set of feedback controllers offers distinct advantages over end-to-end methods during training. In particular, by constraining the paths to prevent intersections with track boundaries, many collisions can be avoided. This is exemplified in Figure 1.12, which depicts the locations where end-to-end and partial end-to-end agents crashed during training. Whereas the end-to-end agent experienced 726 crashes in 1170 training episodes, the partial end-to-end agent encountered 11 crashes in 113 training episodes. Moreover, these crashes were as result of poor tracking performance of the pure pursuit controller at high speeds.

Having determined that partial end-to-end have an advantage over end-to-end agents during training, we assessed the performance of end-to-end and partial end-to-end agents
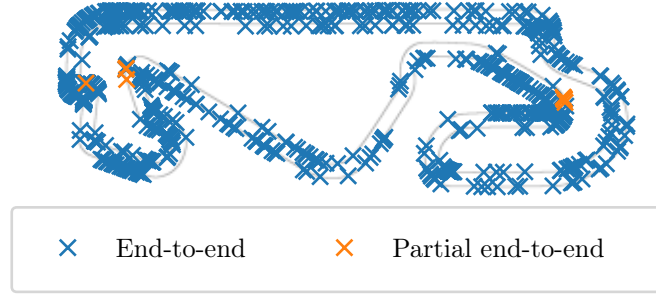
Figure 1.12: Locations where the end-to-end and partial end-to-end agents crashed during training.

under the evaluation conditions specified in Section **??**. The percentage successful laps and lap time of both end-to-end and partial end-to-end agents racing on every track is presented in Table 1.3. We observe similar trends during evaluation as in training.

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	5.86
Barcelona-Catalunya	79	47.47	100	48.34
Monaco	61.67	35.82	99.33	36.39

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

While end-to-end agents do successfully complete all their evaluation laps on the Porto track, they experience crashes on the more complex tracks, namely Barcelona-Catalunya and Monaco. Notably, end-to-end agents race faster than partial end-to-end agents on these more complex tracks, at the cost of safety. On the other hand, partial end-to-end agents experience a relatively low percentage of crashes on the Monaco track, while successfully completing all their laps on the remaining tracks.

Figures 1.13 and 1.14 illustrate the paths executed by partial end-to-end agents 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 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. These observations were consistent across both the Barcelona-Catalunya and Monaco tracks.
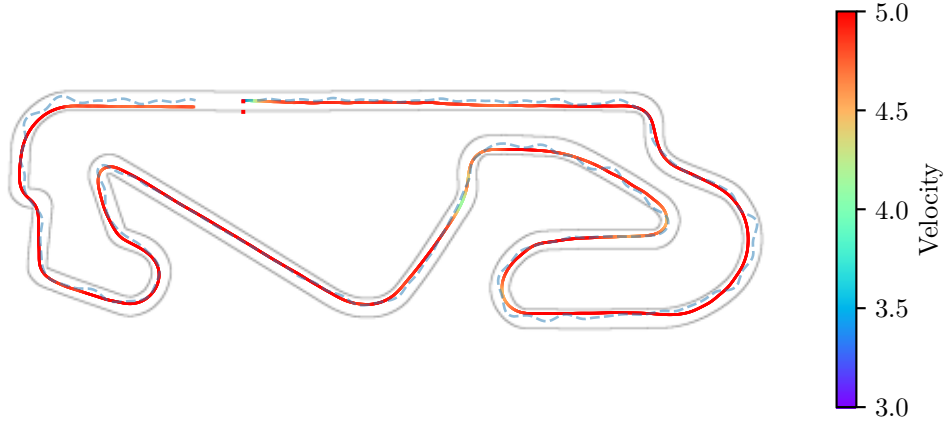
Figure 1.13: The path and velocity profile taken by a partial end-to-end agent completing Circuit de Barcelona-Catalunya. For comparison, the path of an end-to-end agent racing on the same track is depicted with a dashed light blue line.

In the Chapter **??**, we observed that end-to-ends agent exhibited dangerous slaloming behavior associated with high slip angles. This behavior was particularly problematic when model-mismatches were present, leading to frequent crashes. Therefore, the absence of slaloming behavior in the partial end-to-end agents is a promising indication that may offer improvements over the performance of end-to-end agents when model-mismatches are present. To ascertain whether the absence of slaloming behaviour has resulted in a reduction in slip angles, we observe the paths and slip angles of agents racing on a section of Circuit de Monaco, as depicted in Figure 1.15.
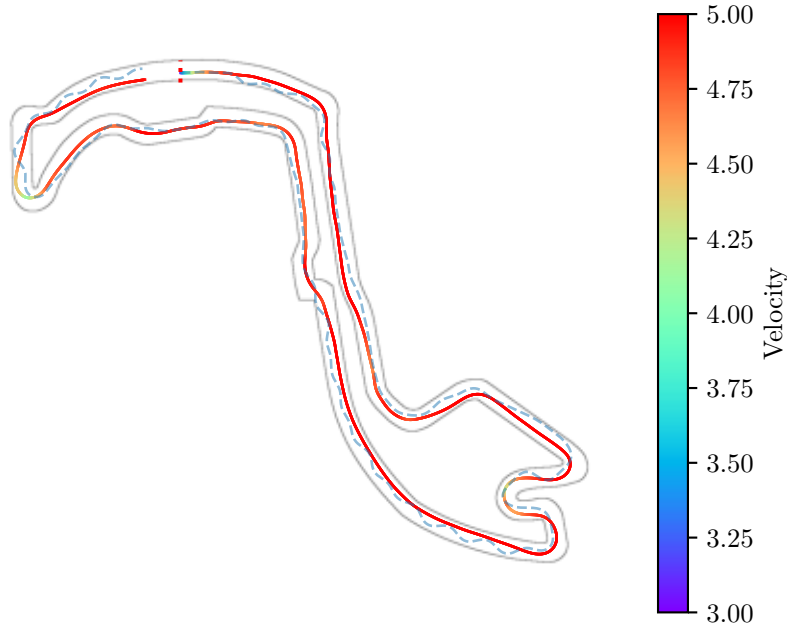

Figure 1.14: The path and velocity profile taken by a partial end-to-end agent completing Circuit de Monaco. For comparison, the path of an end-to-end agent racing on the same track is depicted with a dashed light blue line.

Although the peak slip angle of the partial end-to-end agent is 0.26 radians when rounding the second-to-last corner, which is similar to the end-to-end agent, the slip angles of the partial end-to-end agents are generally close to 0 radians. In contrast, end-to-end agents experience large slip angles throughout the track due to slaloming. Therefore, although the absence of slaloming has not reduced the peak slip angle, it has resulted in a reduction in the average slip angle across the track.
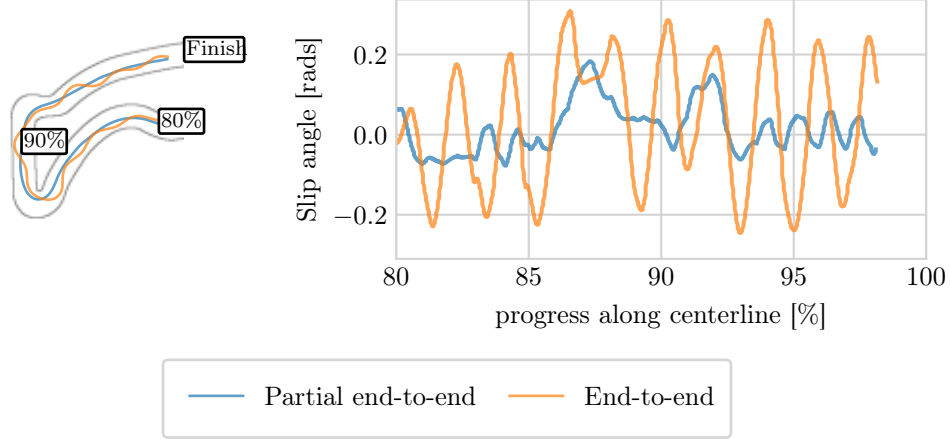

Figure 1.15: The paths and slip angles of agents racing on the final section of Circuit de Monaco.

1.7 Evaluating alternative partial end-to-end algorithm architectures

After having determined that partial end-to-end agents have and advantage over end-to-end agents in terms of training performance and learned behaviour, an ablation study was conducted to assess the impact of each individual controller on the algorithm's behavior. In this study, we trained agents 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 with one controller removed were trained using the same hyper-parameters as the partial end-to-end agents with both controllers. Additionally, we followed the same training procedure as described in Section **??** for the end-to-end agents.

Figure 1.16 presents the percentage of failed laps, lap time, and learning curves for each algorithm, including the end-to-end approach, during the training process. 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. 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. This is attributed to its ability to minimize both the percentage of failed laps and lap time simultaneously.
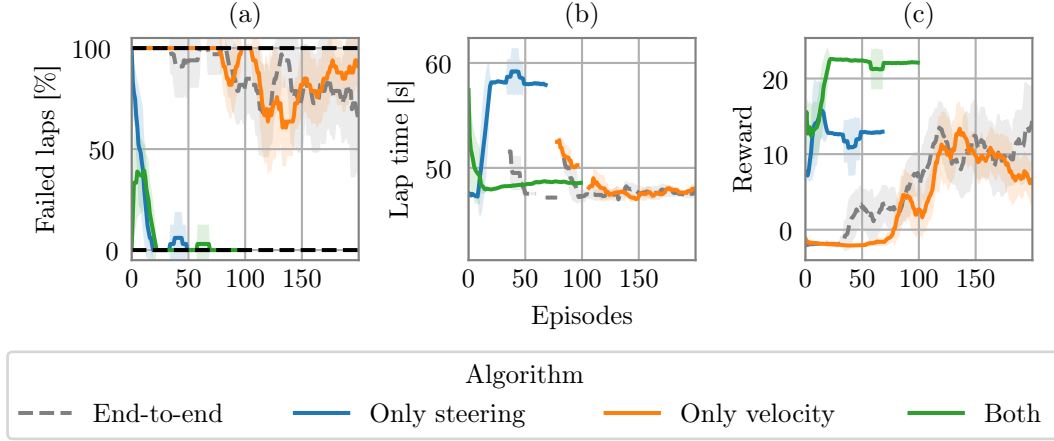
Figure 1.16: (a) The percentage failed laps, (b) lap time and (c) learning curve during training for agents utilising each algorithm structure.

As part of the ablation study, we qualitatively evaluated the trajectories executed by each algorithm on Barcelona-Catalunya. Figure 1.17 displays the paths taken by agents using different algorithms. The paths taken by agents without a steering controller are depicted on the left, whereas the paths taken by agents utilising a 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
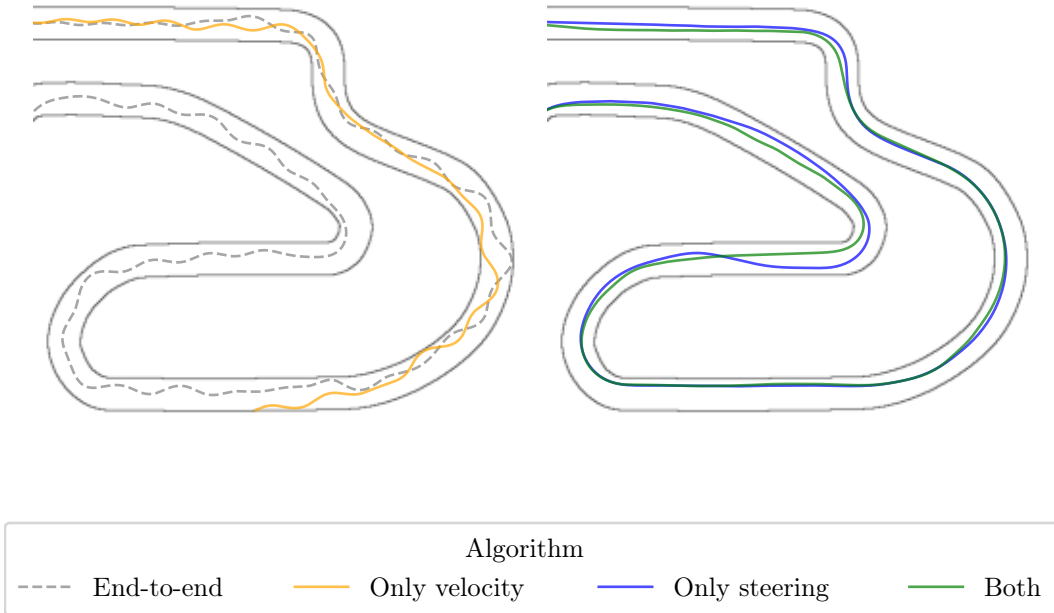

Figure 1.17: The paths taken by agents utilising each algorithm architecture. Paths taken by agents without a steering controller are depicted on the right, whereas the paths taken by agents utilising a steering controller are depicted on the left.

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.

We observed that by constraining the path using the Frenet frame, agents are able to train with a significantly reduced number of crashes. This allows them to train much faster than end-to-end agents. Partial end-to-end agents exhibited improved performance over end-to-end agents under evaluation conditions in terms of success rate. Furthermore, partial end-to-end agents exhibited smooth trajectories, resulting in lower slip angles than end-to-end agents throughout most sections of the tracks tested.

These evaluations were conducted under conditions where no model mismatches were present. However, the motivation for developing a partial end-to-end algorithm is that it may offer performance advantages in settings where model mismatches are present. In the next chapter, we will assess the performance of the partial end-to-end agents under conditions where model mismatches are introduced intentionally. This assessment aims to determine whether the partial end-to-end agents offer any advantages over the end-to-end baseline agents in such scenarios.

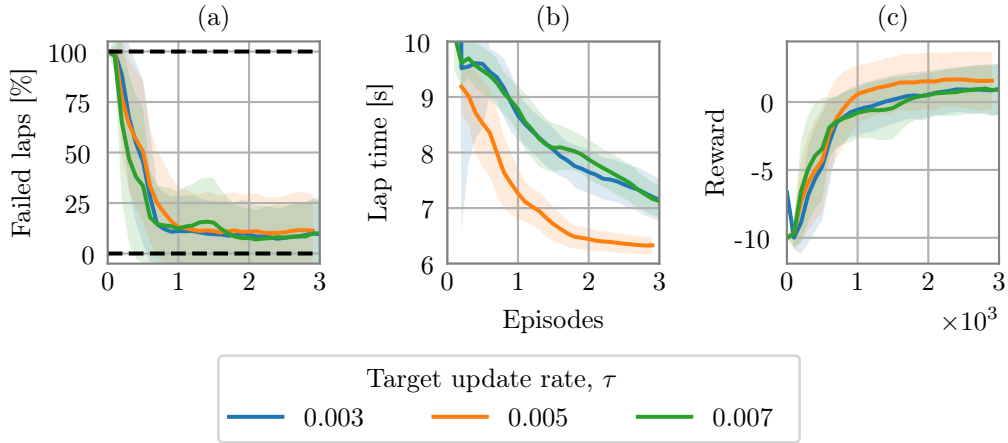# Appendices

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 $7 \cdot 10^{-3}$.
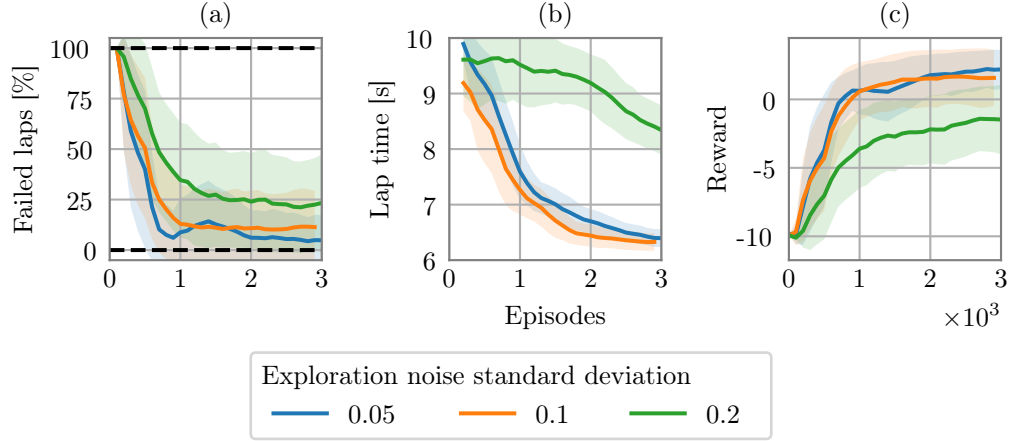
Figure A.2: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with exploration noise standard deviations ranging from 0.05 to 0.2.

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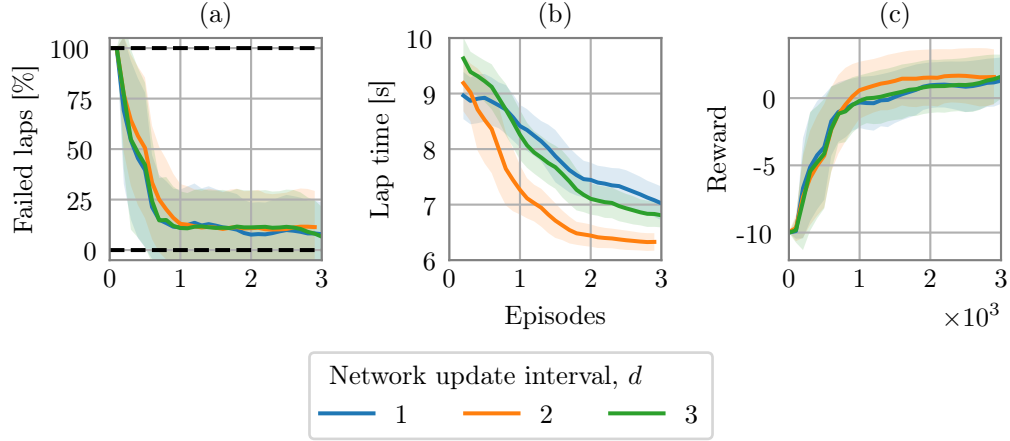
Figure A.3: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with network update intervals d ranging from 1 to 3.

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`