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

Chapter 1

Modelling the autonomous racing problem

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

1.1 Requirements of F1tenth autonomous racing

We have chosen to model an F1tenth race setting because it is a well researched platform with a standardised vehicle model and hardware requirements. F1tenth racing utilises one-tenth scale remote control (RC) car chassis as a vehicle platform. These vehicles are equipped with all the elements that are necessary for autonomous navigation, including a front facing LiDAR sensor, an Nvidea Jetson module for on-board computing, a servo motor for controlling the front wheels, and a DC motor for controlling the rear wheels. The front wheels are only used for steering, while the rear-wheels are driven. An example of such an F1tenth vehicle is shown in Figure 1.1.

The standardised F1tenth simulator is incompatible with fully end-to-end systems due to its inclusion of a velocity controller. We therefore chose to develop an in-house simulator that excludes the velocity controller, which allows us to compare our system to a fully end-to-end system.

he race rules that we consider in our simulator are as follows: a single vehicle competes to complete a single lap of a racetrack in a time trial setting. The vehicle may be started anywhere along the length of the track. The start and finish line then coincides with the starting position of the vehicle. If the vehicle makes contact with the track boundary, the lap is considered failed and the lap time is not counted. Furthermore, we represent the track boundary as a solid wall, such that points on the LiDAR scan correspond to a track boundary.

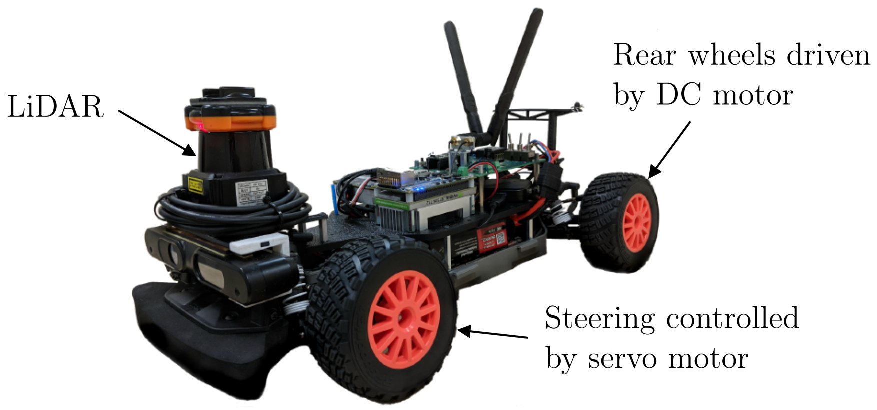

LiDAR
Rear wheels driven by DC motor
Steering controlled by servo motor

Figure 1.1: An example of an F1tenth vehicle [1], showing the position of the sensors and motors.

1.2 Vehicle dynamics

We start the discussion of our simulator by detailing the vehicle dynamics model that was utilised to accurately represent the motion of the vehicle. An F1tenth vehicle was modelled using the single-track bicycle model by Altoff and Würshing [2]. This model takes vehicle slipping into account, and is accurate even when the vehicle is driven close to its handling limits, thus making it suitable for the racing context. The single-track bicycle model assumes the car is a rigid body, with the two front wheels consolidated into a central wheel. Similarly, the rear wheels are also consolidated into a single center wheel.

We use discretised time notation for the single-track bicycle model,

$$t = k \cdot \Delta t, \tag{1.1}$$

where the time is denoted as t, the time step as k, and the difference in time between successive time steps k and $k+1$ as Δt. This discretisation is useful in the context of MDPs, since MDPs themselves are discrete-time.

The inputs to the single-track bicycle model at time step k are the current state, denoted as $\mathbf{x}$, and control actions, which are denoted $\mathbf{u}$. The output of the single-track bicycle model is then the time derivative of state, and is denoted as $\dot{\mathbf{x}}$. At every time step k, Euler's method was applied to integrate $\dot{\mathbf{x}}$, thereby determining the state at the next time step.

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

$$\mathbf{x} = [s_x,\ s_y,\ \delta,\ \upsilon,\ \psi,\ \dot{\psi},\ \beta]^\intercal. \tag{1.2}$$

A description of each variable within the state vector, along with its unit and reference direction is given in Table 1.1.

The control inputs to the state space model are represented by the vector $\mathbf{u}$. These inputs are a steering rate, denoted as $\dot{\delta}$ (measured in rad/s), and a longitudinal acceleration, denoted as a_{long} (measured in m/s):

$$\mathbf{u} = [\dot{\delta}, a_{\text{long}}]^\intercal. \tag{1.3}$$

The single-track bicycle model is depicted with its state variables and control inputs in Figure 1.2.

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

Table 1.1: A description of each state space variable, along with its symbol, unit and reference direction. A depiction of each of these state variables on a diagram of the vehicle is shown in Figure 1.2.

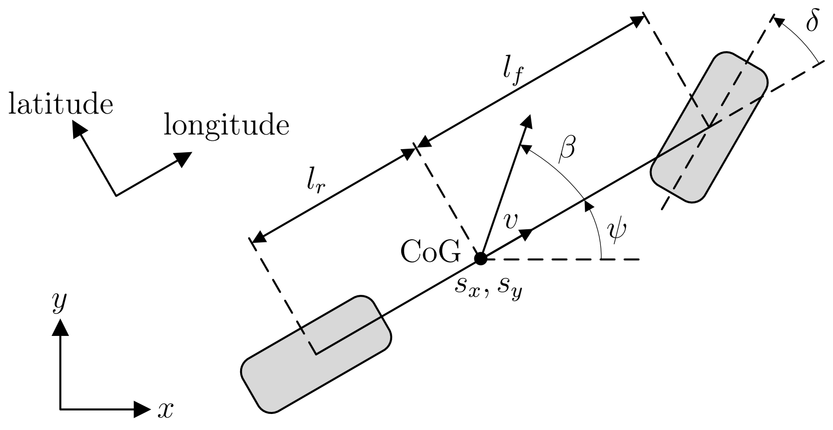


Figure 1.2: The single-track vehicle dynamics model with state variables. The centre of gravity (CoG) is depicted as being l_f m away from the front axle, and l_r m from the rear axle. Furthermore, the longitude and latitude axes are parallel and perpendicular to the forward direction.

Altoff and Würshing [2] assume that the vehicle is equipped with motors capable of achieving the desired motion specified by the control inputs, so long as the control inputs are within the physical limitations of the actuators. The following constraints are therefore applied to the steering angle, steering rate, and velocity:

$$\dot{\delta} \in [\underline{\dot{\delta}}, \overline{\dot{\delta}}], \quad \delta \in [\underline{\delta}, \overline{\delta}], \quad v \in [\underline{v}, \overline{v}]. \tag{1.4}$$

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

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

$$a_{\text{long}} \in [\underline{a}, \overline{a}(v)], \quad \overline{a}(v) = \begin{cases} a_{\max}\frac{v_S}{v} & \text{for } v > v_S \\ a_{\max} & \text{otherwise,} \end{cases} \tag{1.5}$$

where $a_{\max}$ is the absolute maximum acceleration that can be achieved without wheel slip, $\underline{a}$ is the maximum deceleration, and $\overline{a}$ is the maximum acceleration that the vehicle can achieve, taking wheel slip into account. A comprehensive description of each constraint, along with its corresponding units, is provided in Table 1.2.

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

Table 1.2: Constraint parameters of a standard F1tenth vehicle.

Practically, the constraints from Equations 1.4 and 1.5 are achieved by applying the case statements

$$\begin{gathered}
\dot{\delta} = \begin{cases} 0 & \text{for } (\delta \le \underline{\delta} \text{ AND } \dot{\delta} \le 0) \text{ OR } (\delta \ge \overline{\delta} \text{ AND } \dot{\delta} \ge 0) \text{ (condition } C1\text{)}, \\ \underline{\dot{\delta}} & \text{for NOT } C1 \text{ AND } \dot{\delta} \le \underline{\dot{\delta}}, \\ \overline{\dot{\delta}} & \text{for NOT } C1 \text{ AND } \dot{\delta} \ge \overline{\dot{\delta}}, \\ \dot{\delta} & \text{otherwise,} \end{cases} \\
a_{\text{long}} = \begin{cases} 0 & \text{for } (v \le \underline{v} \text{ AND } a_{\text{long}} \le 0) \text{ OR } (v \ge \overline{v} \text{ AND } a_{\text{long}} \ge 0) \\ & \text{(condition } C2\text{)}, \\ \underline{a} & \text{for NOT } C2 \text{ AND } a_{\text{long},d} \le \underline{a}, \\ \overline{a(v)} & \text{for NOT } C2 \text{ AND } a_{\text{long},d} \ge \overline{a}(v), \\ a_{\text{long}} & \text{otherwise,} \end{cases}
\end{gathered} \tag{1.6}$$

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

After applying these constraints to the input vector $\mathbf{u}$, non-linear state-space equations with inputs $\mathbf{u}$ and $\mathbf{x}$ are applied. Altoff and Würshing [2] define the state-space model as a piece-wise function defined dependent on the velocity. For large velocities, a dynamic bicycle model denoted by $f_1(\mathbf{x}, \mathbf{u})$ is used, as it accounts for tire slip and accurately represents the motion close to the physical limits of the vehicle. However, this dynamic bicycle model becomes singular for small velocities. Therefore, a kinematic bicycle model $f_2(\mathbf{x}, \mathbf{u})$ that does not take tire slip into account is used for velocities slower than 0.1 m/s. The piece-wise defined dynamics model is then

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{cases} f_1(\mathbf{x}, \mathbf{u}) & \text{if } |v| \ge 0.1 \text{ m/s}, \\ f_2(\mathbf{x}, \mathbf{u}) & \text{else.} \end{cases} \tag{1.7}$$

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

$$\begin{aligned}
\dot{s_x} &= v\cos(\psi + \beta), \\
\dot{s_y} &= v\sin(\psi + \beta), \\
\dot{\delta} &= \dot{\delta}, \\
\dot{v} &= a_{\text{long}}, \\
\dot{\psi} &= \dot{\psi}, \\
\ddot{\psi} &= \frac{\mu m}{I_z(l_r + l_f)}(l_f C_{S,f}(gl_r - a_{\text{long}}h_{cg})\delta + (l_r C_{S,r}(gl_f + a_{\text{long}}h_{cg}) \\
&\quad - l_f C_{S,f}(gl_r - a_{\text{long}}h_{cg}))\beta - (l_f^2 C_{S,f}(gl_r - a_{\text{long}}h_{cg}) + l_r^2 C_{S,r}(gl_f + a_{\text{long}}h_{cg}))\frac{\dot{\psi}}{v}), \\
\dot{\beta} &= \frac{\mu}{v(l_r + l_f)}(C_{s,f}(gl_r - a_{\text{long}}h_{cg})\delta - (C_{S,r}(gl_f + a_{\text{long}}h_{cg}) + C_{S,f} \\
&\quad (gl_r - a_{\text{long}}h_{cg}))\beta + (C_{S,r}(gl_f + a_{\text{long}}h_{cg})l_r - C_{S,f}(gl_r - a_{\text{long}}h_{cg})l_f)\frac{\dot{\psi}}{v}) - \dot{\psi},
\end{aligned} \tag{1.8}$$

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

$$\begin{aligned}
\dot{s_x} &= v\cos(\psi + \beta), \\
\dot{s_y} &= v\sin(\psi + \beta), \\
\dot{\delta} &= \dot{\delta}, \\
\dot{v} &= a_{\text{long}}, \\
\dot{\psi} &= \frac{v\cos(\beta)}{l_{wb}}\tan(\delta), \\
\ddot{\psi} &= \frac{1}{l_{wb}}(a_{\text{long}}\cos(x_7)\tan(\delta) - x_4\sin(\beta)\tan(\delta)\dot{\beta} + \frac{v\cos(\beta)}{\cos^2(\delta)}\dot{\delta}), \\
\dot{\beta} &= \frac{1}{1 + (\tan(\delta)\frac{l_r}{l_{wb}})^2} \cdot \frac{l_r}{l_{wb}\cos^2(\delta)}\dot{\delta},
\end{aligned} \tag{1.9}$$

where m is the vehicle mass, I_z is the moment of inertia about the z axis of the vehicle (i.e., a vertical axis that passes through the CoG), l_f is the distance from the CoG to the front axle, l_r is the distance from the CoG to the rear axle, and h_{cg} is the height of CoG. Furthermore, the parameters $C_{S,f}$ and $C_{S,r}$ are the cornering stiffness coefficients of the the front and rear tires, respectively. The tire cornering stiffness coefficient is the ratio between the lateral force acting on the tire, and its slip angle, which is the angle between the direction the wheel is pointing, and its direction of travel. Additionally, the road surface friction is denoted μ. The values for these vehicle parameters were identified for a standard F1tenth vehicle [1], and are summarised in Table 1.3.

The state space Equations 1.8 and 1.9 yield the derivative of the state with respect to time. Consequently, to determine the state at the next time step, we employ numerical integration

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \dot{\mathbf{x}}\Delta t, \tag{1.10}$$

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

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

Table 1.3: Vehicle model parameters for the single track model. The parameters were identified for a standard F1tenth vehicle.

1.3 Simulation environment

We will now discuss the design of a simulator that accurately represents the F1tenth autonomous racing problem. This involves simulating the vehicle using the single-track bicycle model, as well as creating a track environment that adheres to the rules specified in Section 1.1.

The simulator runs an initialisation procedure at the start of each episode, after which it is sampled at every time step. Algorithm 1 details the procedure that the simulator executes at the beginning of an episode. This procedure takes as input a bird's-eye image of the racetrack, as well as the coordinates, velocity and heading at which to initialise the vehicle.

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

Algorithm 1: The simulator initialisation procedure.

```
Input: An image of a track, starting position, velocity and heading: $[s_x, s_y, v, \psi]$
1 Generate occupancy grid
2 Find centerline
3 Start vehicle at random point along centerline
4 Set start/ finish line
```

Once the occupancy grid is generated, the simulator proceeds to determine the centerline of the track. The centerline refers to a line that stretches around the track, and maintains an equal distance from either side of the track boundaries. To represent the centerline, a cubic spline [3] is employed.

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

Once initialised, the simulator is sampled at discrete time steps to simulate the vehicle dynamics forward according to the control inputs. This is done using the single-track bicycle model from Section 1.2. The simulator receives an input a_k at time step k and produces outputs at time step $k+1$. The output of the simulator is an observation, denoted o_{k+1}, and reward, denoted as r_{k+1}, which is consistent with the MDP environment definition. The interaction between the autonomous racing algorithm and simulator is shown in Figure 1.3. Additionally, the procedure that the algorithm executes at every time step is detailed in Algorithm 2.

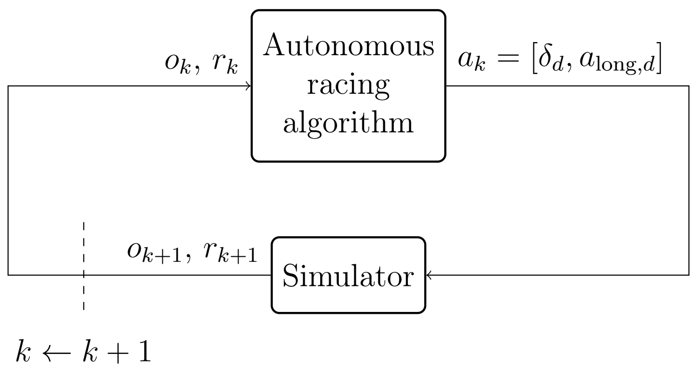

o_k, r_k
Autonomous racing algorithm
$a_k = [\delta_d, a_{\text{long},d}]$
o_{k+1}, r_{k+1}
Simulator
$k \leftarrow k+1$

Figure 1.3: The interaction between the racing algorithm and simulator at every time step is analogous to that of the agent and MDP.

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

$$o_k = [\underbrace{s_x, s_y, \delta, v, \psi,}_{\text{Pose}} \underbrace{l_0, l_1, l_2, \cdots L}_{\text{LiDAR scan}}]. \tag{1.11}$$

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

```
Algorithm 2: The simulator execution at every time step.
  Input: Control actions $\dot{\delta}_d$ and $a_{\text{long}_d}$ at time step $k$.
  Output: An observation $o_{k+1}$, reward $r_{k+1}$ at time step $k+1$.
1 Simulate servo motor: $\dot{\delta} = \frac{\delta_d - \delta}{|\delta_d - \delta|}\bar{\dot{\delta}}$,
2 Update state: $\mathbf{x}_{k+1} = \mathbf{x}_k + f(\mathbf{x}_k, \mathbf{u}_k)\Delta t$
3 Generate LiDAR scan: $[l_0, l_1, \cdots L]$
4 Sample the observation: $o_{k+1} = [s_x, s_y, \delta, v, \psi, l_0, l_1, l_2, \cdots L]$
5 Check for collisions
6 Get distance travelled: $\Delta D = D_{k+1} - D_k$
7 Generate reward: $r_{k+1} = r(s_k, a_k)$
```

The vehicle dynamics are simulated in line 1 of Algorithm 2. When simulating these dynamics, we abstract the longitudinal acceleration controller and assume that the vehicle can accelerate at the desired rate, so long as the desired rate is within the bounds of the upper and lower acceleration limits. The desired acceleration $a_{\text{long},d}$ then serves as the control input a_{long} to the state space model. However, we simulate the rate of change of steering angle caused by steering servo motor,

$$\dot{\delta} = \frac{\delta_d - \delta}{|\delta_d - \delta|}\bar{\dot{\delta}}, \tag{1.12}$$

which is used as the steering rate control input for the single-track bicycle model.

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

$$\Delta D = D_{k+1} - D_k, \tag{1.13}$$

where distance D_k represents the point on the centerline closest to the vehicle at time step k. The ΔD values at each time step are accumulated, and if the sum is equal to or greater than the track length, the vehicle has completed one lap. An illustration of the quantity ΔD is shown in Figure 1.4. Finally, the reward signal is generated based on the state and action,

$$r_{k+1} = r(\mathbf{x}_k, a_k). \tag{1.14}$$

The details of this reward signal are discussed in the next chapter.

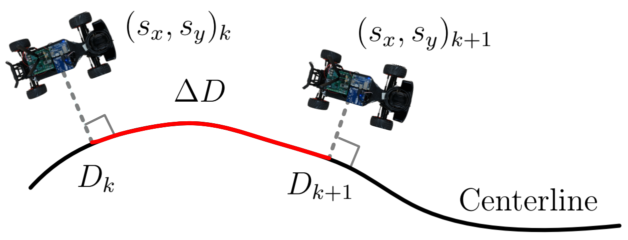


Figure 1.4: An F1tenth vehicle moving along the track centerline. The red section of the centerline indicates ΔD.

1.4 Summary

This chapter introduced the racing scenario, vehicle modelling, as well as simulator. By allowing F1tenth vehicles to race alone on the track, the setting that we consider provides an opportunity for autonomous racing algorithms to compete in a time-trial style environment. Furthermore, the F1tenth vehicle is modelled using single-track bicycle dynamics by Altoff and Würshing [2], which accurately predicts the motion of the vehicle close to the handling limits. The dynamics model is incorporated into a simulator that encompasses both the race car dynamics and the track environment. By discretizing time, the simulator conforms to the time discrete framework of the MDP. At each time step, the simulation receives a control input from the autonomous racing algorithm and generates an observation corresponding to the updated state of the environment. Approaches to solving the autonomous racing problem that utilise an end-to-end design have an agent whose outputs (i.e., the control actions) are directly passed to the simulator. In the next chapter, we utilise our racing simulation within an MDP environment to train such an end-to-end RL agent to race.

Chapter 2

End-to-end autonomous racing

Having introduced our simulation environment, we formulate an end-to-end solution method in which a reinforcement learning (RL) agent directly predicts controller commands based on observation information. We employ this end-to-end agent as a baseline to compare our partial end-to-end algorithm against, as similar end-to-end methods are commonly used to solve the racing problem [4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14].

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

2.1 End-to-end algorithm design

Our end-to-end autonomous racing algorithm is composed of an RL agent and a velocity constraint. The agent maps an observation sampled from the simulator to desired longitudinal acceleration ($a_{\text{long},d}$) and steering angle (δ_d) control commands. The velocity constraint then modifies the acceleration commands to ensure that the vehicle remains within safe velocity bounds. The steering angle from the agent and acceleration from the velocity constraint component are passed to the simulator described in Chapter 1. This end-to-end framework is depicted in Figure 2.1.

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

The end-to-end agent, which comprises a neural network, is shown in Figure 2.2. To ensure uniformity across all observation vector elements, each element in the input vector is normalized to the range $(0, 1)$. The neural network's design consists of three fully connected layers, with m_1, m_2, and 2 neurons in the input, hidden, and output layers,

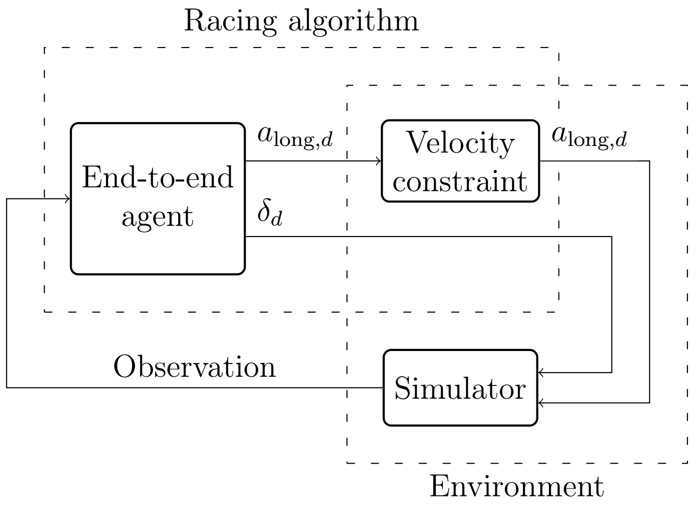


Figure 2.1: The end-to-end racing algorithm, which is comprised of an RL agent which outputs control actions, and a velocity constraint. The velocity constraint and simulator are both considered part of the environment.

respectively. The first two layers are ReLU-activated, while the output layer is activated by a hyperbolic tangent function to normalize the neural network output to the range $(-1, 1)$. While the number of neurons in the first two layers are determined empirically, the two neurons in the output layer correspond to the steering and acceleration actions. Scaling factors are applied to their outputs so that the selected steering and acceleration actions fall within the range $(\underline{\delta}, \overline{\delta})$ and $(\underline{a}, \overline{a})$ from Table 1.2, respectively.

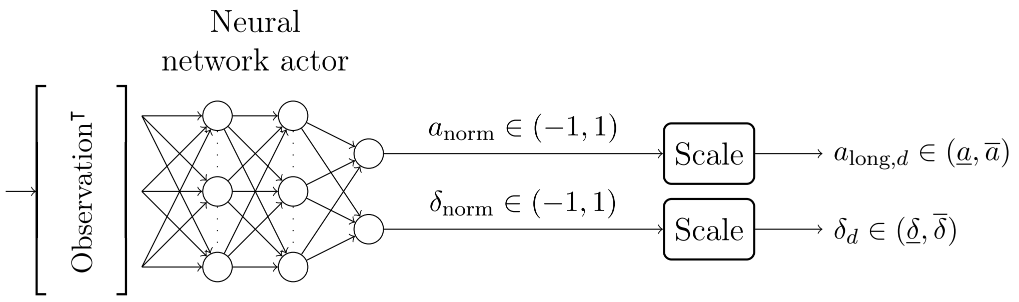

Neural network actor
Observation$^\intercal$
$a_{\text{norm}} \in (-1, 1)$
$\delta_{\text{norm}} \in (-1, 1)$
Scale
Scale
$a_{\text{long},d} \in (\underline{a}, \overline{a})$
$\delta_d \in (\underline{\delta}, \overline{\delta})$

Figure 2.2: The end-to-end agent. The outputs of the neural network are scaled to the ranges of a_{long} and δ in Table 1.2.

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

$$a_{\text{long},d} \leftarrow \begin{cases} 0 & \text{for } v \geq v_{\text{max}}, \\ 0 & \text{for } v \leq v_{\text{min}}, \\ a_{\text{long},d} & \text{otherwise,} \end{cases} \tag{2.1}$$

before being passed to the simulator. v_{max} and v_{min} are the imposed maximum and minimum allowable velocities.

2.2 Applying TD3 to end-to-end autonomous racing

We applied the TD3 RL algorithm from Section **??** to train the end-to-end agent. Several adaptations to the original TD3 algorithm were made to ensure its compatibility with the end-to-end racing algorithm. The adapted TD3 is shown in Algorithm 3.

Algorithm 3: Twin delay deep deterministic policy gradient

```
Input: Table 2.1 parameters
Output: Trained actor DNN $\pi_{\boldsymbol{\phi}}$

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

4  while MDP time steps $< M$ do
5      Reset simulator (start new episode)
6      for $t$=0, $T$ do
7          Sample action with exploration noise from end-to-end agent,
           $a_t = [a_{\text{long},d}, \delta_d] \leftarrow \text{scale}(\pi_{\boldsymbol{\phi}}(o_t) + \epsilon), \;\; \epsilon \sim \mathcal{N}(0, \sigma_{\text{action}})$
8          for $n$=0, $N$ do
9              Modify $a_{\text{long},d}$ according to Equation 2.1 to limit velocity
10             Simulator executes action $a_t$
11             Observe environment step reward $r_{t,n}$
12             Update MDP one step reward: $r_t = r_t + r_{t,n}$
13             Sample observation with noise,
               $o_t \leftarrow o_t + \epsilon, \;\; \epsilon \sim \mathcal{N}(0, \sigma_{\text{observation}})$
14         end
15         Store transition tuple $(o_t, a_t, r_t, o_{t+1})$ in $\mathcal{B}$

16         Sample mini-batch of $B$ transitions $(o_i, a_i, r_i, o_{i+1})$ from $\mathcal{B}$
17         Select target actions:
           $\tilde{a} \leftarrow \pi_{\boldsymbol{\phi}'}(o_{t+1}) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
18         Set TD target: $y_i \leftarrow r_i + \gamma \min_{q=1,2} Q_{\boldsymbol{\theta}'_q}(o_{i+1}, \tilde{a})$
19         Update critics by minimising the loss: $J_{\boldsymbol{\theta}_q} \leftarrow \frac{1}{N}\sum_i^N (y_i - Q_{\boldsymbol{\theta}_q}(o_i, a_i))^2$

           if $t \mod d$ then
20             Update $\boldsymbol{\phi}$ by the deterministic policy gradient:
               $\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) = \frac{1}{N}\sum_i^N \nabla_a Q_{\boldsymbol{\theta}_1}(o_i, a)|_{a=\pi_{\boldsymbol{\phi}}(o_i)} \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(o_i)$
21             Update target networks:
22             $\boldsymbol{\theta}'_q \leftarrow \tau\boldsymbol{\theta}_q + (1-\tau)\boldsymbol{\theta}'_q$
23             $\boldsymbol{\phi}' \leftarrow \tau\boldsymbol{\phi} + (1-\tau)\boldsymbol{\theta}'$
24         end
25     end
26 end
```

Note that end-to-end agents in the context of racing receive only partial information about the state of the environment, because the pose and LiDAR scan do not fully capture the environment state. Hence, the racing environment is only partially observable. As such, the input to a racing agent is therefore an observation, denoted as o, rather than the complete environment state s. This notation conforms to the notation used for the output of the simulator in Chapter 1. However, we use the notation for time steps in Chapter **??**, denoting a time step as t, rather than k.

In line 1 of Algorithm 3, the actor ($\pi_{\boldsymbol{\phi}}$) and critics ($Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$) are initialised. The end-to-end agent shown in Figure 2.2 is the actor $\pi_{\boldsymbol{\phi}}$. The design of the critics $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$ are now described. For simplicity, our critics have an identical structure which is analogous to that of the actor. We therefore describe the details of only one critic, which is depicted in Figure 2.3. The critic DNN receives a vector input comprised of observation

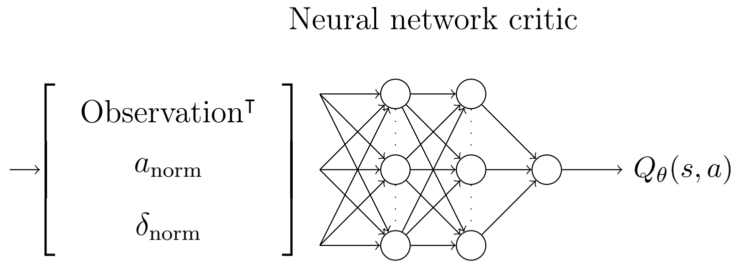


Figure 2.3: The critic DNN. Its input is a vector containing a normalised observation and action pair, and its output is an action-value.

and control actions normalised to the range $(-1,1)$. It comprises three fully connected layers: an input layer with m_1 neurons, a hidden layer with m_2 neurons, and an output layer with a single neuron. The output of this neuron is the action-value, denoted $Q_{\boldsymbol{\theta}}(o,a)$. The first two layers utilize the Rectified Linear Unit (ReLU) activation function, while the final layer uses a linear activation function. Additionally, the target networks are initialised identically to their counterparts.

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

In line 7 of Algorithm 3, an action is sampled from the end-to-end agent by forward passing the observation through the actor DNN. Gaussian noise with a zero mean and a standard deviation of σ_{action} is added to the normalised output (a_{norm} and δ_{norm}), which is then scaled to generate a longitudinal acceleration and a steering angle.

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

$$N = \frac{100}{f_{\text{agent}}}. \tag{2.2}$$

In Equation 2.2, N is a whole number. The environment is sampled by applying the velocity constraint from Equation 2.1 to limit the agent's selected longitudinal action, and then executing one simulator step. This is followed by sampling the reward signal from the simulator in line 11.

The reward signal is designed to closely approximate the objective of minimizing lap time for high reward discount rates. Specifically, we reward the agent for the distance it travels along the centerline between the current and previous time step, while penalizing it a small amount on every time step, as described by Fuchs et al. [5]. In addition, we impose a large penalty if the agent collides with the track boundary. As a result, we obtain a piece-wise reward signal expressed as

$$r(s_t, a_t) = \begin{cases} r_{\text{collision}} & \text{if collision occurred} \\ r_{\text{dist}}(D_t - D_{t-1}) + r_{\text{time}} & \text{otherwise.} \end{cases} \tag{2.3}$$

Here, $r_{\text{collision}}$, r_{dist}, and r_t represent the penalty for collisions, the reward for distance traveled, and the penalty for each time step, respectively. Notably, this reward signal is similar to those used in numerous prior works [4; 6; 7].

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

The remaining steps of Algorithm 3 are identical to the standard implementation of TD3, as described in Algorithm **??**. Specifically, we first sample a mini-batch of B transitions, from the replay buffer. Next, we employ the target actor network to select actions for each observed sample, which in turn are used to update the critics. To ensure the stability of the learning process, we update the actor and the target networks every d steps. Additionally, the target networks are updated via a soft update which is controlled by the target update rate parameter τ.

After the training procedure is completed, we utilise Algorithm 4 to evaluate the trained agents. Under evaluation conditions, learning was halted and the weights of the DNNs were not updated. Furthermore, no exploration noise was added to the agents selected actions. However, Gaussian noise was added to the observation vector to mimic practical sensor data in simulation. This added 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 LiDAR scan. Each agent completed 100 laps under these evaluation conditions.

2.3 Empirical design and hyperparameter values

The previous section introduced the design of the end-to-end algorithm and the TD3 algorithm with symbolic hyper-parameter values. The optimal values for these hyper-parameters cannot be derived, and require experimentation to be determined empirically. Furthermore, hyper-parameters are sensitive to the track. The following five sections of this chapter detail the experiments that were undertaken to determine a locally optimal set of hyperparameters for agents racing on a relatively simply track named Porto. The selected hyperparameters are listed in Table 2.1. Additionally, the average learning curve for 10 agents racing on this track using this set of hyper-parameter is shown in Figure 2.4.

Algorithm 4: Evaluating the end-to-end algorithm without exploration noise, and with observation noise.

```
Input: Trained actor DNN $\pi_\phi$
Output: Lap times, collisions over 100 laps
1  Initialise actor DNN $\pi_\phi$ with weights $\phi$ from training
2  for episode = 1, 100 do
3      for $t$=0, $T$ do
4          Select control action $a_t = [a_{\text{long,d}}, \delta_d] \sim \text{scale}(\pi_\phi(o_t + \epsilon))$
5          for $n$=0, $N$ do
6              Modify $a_{\text{long},d}$ according to Equation 2.1 to limit velocity
7              Simulator executes action $a_t$
8          end
9      end
10 end
```

Hyper-parameter	Symbol	Value
Algorithm	$1.5 \cdot 10^5$	
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}	0.25
Time step penalty	r_{time}	0.01
Collision penalty	$r_{\text{collision}}$	-10
Observation		
Number of LiDAR beams	L	20
Neural network		
Learning rate	α	10^{-3}
Neurons in input layer	m_1	400
Neurons in hidden layer	m_2	300
Velocity constraints		
Minimum velocity	v_{min}	3 m/s
Maximum velocity	v_{max}	5 m/s

Table 2.1: Selected values of hyper-parameters for the end-to-end racing algorithm on the Porto track.

To select each hyper-parameter value listed in this Table, we repeatedly trained agents using Algorithm 3 varied values of the hyper-parameter under consideration while keeping all other hyper-parameters fixed at the values listed in Table 2.1. When evluating agents, we are particularly interested in the rate at which they successfully complete laps, as well as their lap time during and after training. Furthermore, to ensure consistency in the results, we trained and evaluated multiple agents for each set of hyper-parameters. Specifically, we chose to train three agents for each hyperparameter set due to time constraints.

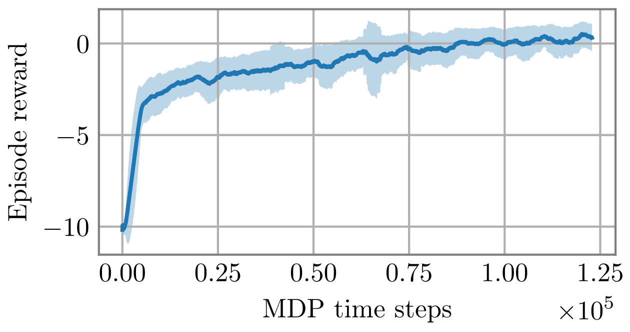


Figure 2.4: Average learning curve for 10 end-to-end agents trained on the simple Porto track.

2.4 TD3 hyper-parameters

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

We first determined an appropriate number of time steps to train the agent for. The objective for determining the length of the training time was to ensure that the agent would demonstrate satisfactory performance under evaluation conditions by racing quickly and consistently avoiding crashes. Ending training too soon may result in poor agent performance, while training for too many time steps long may result in unnecessarily prolonged training times. To achieve this, a set of three agents with the hyper-parameters listed in Table 2.1 were trained. These agents were intermittently evaluated using Algorithm 4 at 100 episode intervals during training. The percentage of failed laps and lap time under evaluation conditions are recorded as a function of training time, as depicted in Figure 2.5.

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

The optimal value for the rate at which actions are sampled from the agent, denoted as f_{agent}, was then determined. We investigated agent sample rates ranging from 3 Hz and 50 Hz, by training three agents with varying sample rates, and their remaining hyperparameters set equal to those listed in Table 2.1. Figure 2.6 shows the training time and percentage of failed laps of agents racing under evaluation conditions as a function

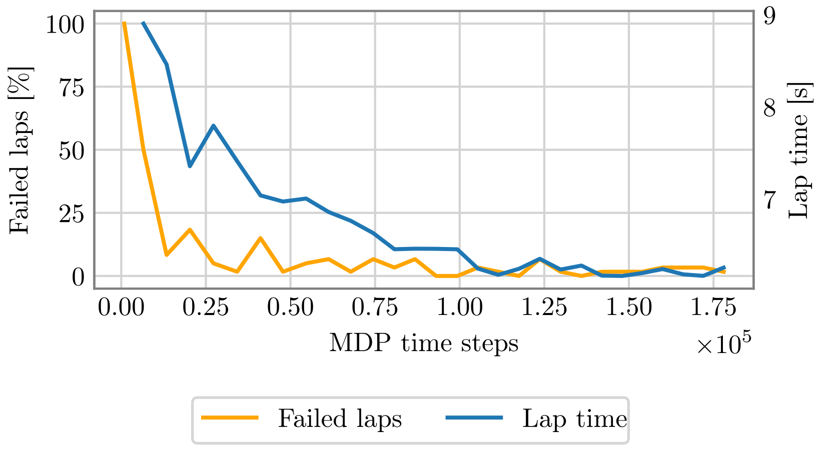


Figure 2.5: Percentage failed laps (left vertical axis) and lap time (right vertical axis) of three agents tested under evaluation conditions at 100 episode intervals during training.

of f_{agent}. Interestingly, there was a correlation between agent sampling rate and training time, as well as the rate at which end-to-end agents complete laps under evaluation conditions.

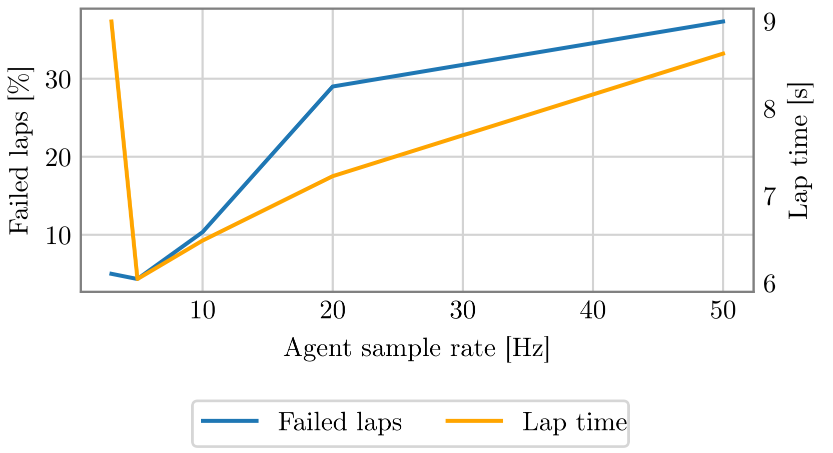


Figure 2.6: Training time (left vertical axis) and percentage failed laps (right vertical axis) of trained end-to-end agents racing under evalution conditions on the Porto track, with sampling rates ranging from 3 Hz to 50 Hz.

From this figure, we observed that agents trained with sampling rates higher than 5 Hz tend to crash. This outcome may be attributed to the fact that when a higher sampling rate is used, the agent needs to learn longer action sequences to complete a lap, leading to a more complex learning problem. We set the value of f_{agent} to be 5 Hz as it resulted in the minimum number of failed laps during testing, despite taking a relatively long 26.2 minutes to train each agent. It is notable that 5 Hz is a relatively slow sampling rate for compared to classical controllers. For instance, Li et al. [15] develop path tracking controllers with sampling rates up to 100 Hz.

We then experimentally determined the optimal value for the batch size B by training agents with batch sizes of 50, 100, 150, 200, 400, 600 and 1000 samples. Varying the batch size did influence lap time and failure rate during evaluation. The lap time and percentage of failed laps during evaluation are shown as a function of batch size in Figure 2.7. From this figure, we observe that lap time and failed laps under evaluation conditions are minimised when the batch size is set to 400. Based on these results, we selected a batch size of 400 samples for our agents.

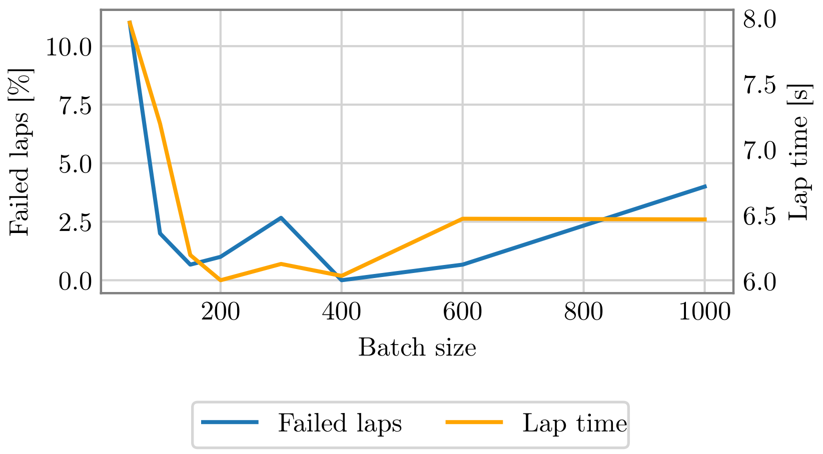


Figure 2.7: Training time and lap time under evaluation conditions of end-to-end agents with batch sizes from 50 to 1000. The percentage failed laps is mapped onto the left vertical axis, while the lap time is mapped onto the right vertical axis.

We then conducted an experimental analysis to select the reward discount rate, denoted γ. To determine the value for γ, we initially assessed the performance of agents with reward discount rates of 0.95, 0.98, 0.99 and 1 during training. The percentage failed laps and lap times during training, was well as the learning curves for these agents are shown in Figure 2.8.

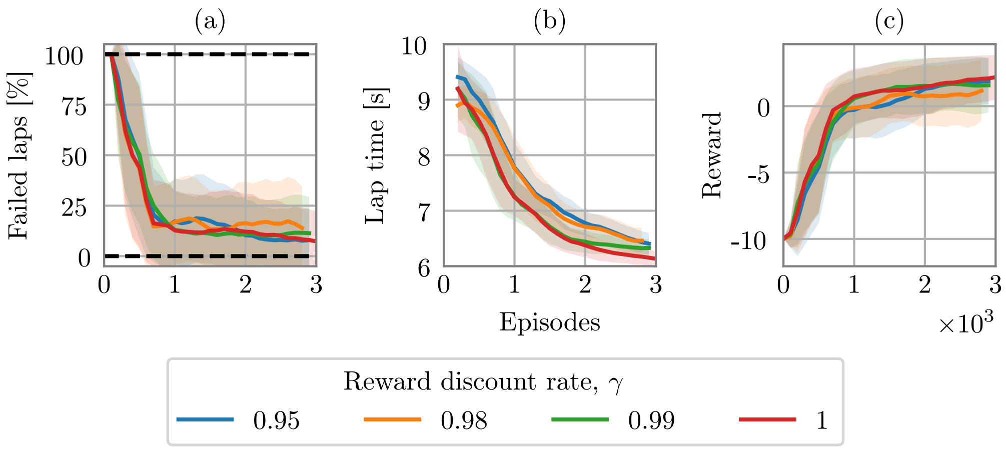


Figure 2.8: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of end-to-end agents with reward discount rates ranging from 0.9 to 1.

By only assessing the performance during training, these agents appear to be perform similarly. However, the TD3 algorithm has no mechanism for decreasing exploration noise with training time. Figure 2.8 is therefore an indicator of the performance of each agent with added exploration noise. As such, we also considered the performance of each agent under evaluation conditions where no exploration noise is present. The percentage failed laps and lap times for agents trained with each learning rate is shown in Table 2.2. The table show that a discount rate of 0.99 yields agents that successfully complete all of their laps. Based on this finding, we have selected a discount rate of 0.99 for our agents.

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

Table 2.2: Percentage failed laps and lap times under evaluation conditions for agents trained with reward discount rates ranging from 0.9 to 1.

We repeated this tuning procedure for the hyper-parameters τ, σ_{action}, and d. For these parameters, we followed the same procedure as for γ, i.e., we varied one hyper-parameter while holding the others constant and selected the value based on performance during training and evaluation. As such, the experiment results for these hyper-parameters are presented in Appendix A. Values of $5 \cdot 10^{-3}$ for τ, 0.1 for σ_{action}, and 2 for d yielded agents with the best performance.

After determining the hyper-parameters for TD3, we compared the performance of our implementation of the TD3 algorithm to a standard implementation of the popular Deep Deterministic Policy Gradient (DDPG) algorithm [6; 16; 10]. The percentage failed laps, lap time and learning curves of agents trained using both algorithms are depicted in Figure 2.9. The results reveal that TD3 outperforms DDPG by a substantial margin in terms of both crashes and lap time. Moreover, we have observed that the training stability of TD3 is superior to that of DDPG, as evidenced by the smoother learning curve of TD3 in contrast to the more erratic curve of DDPG.

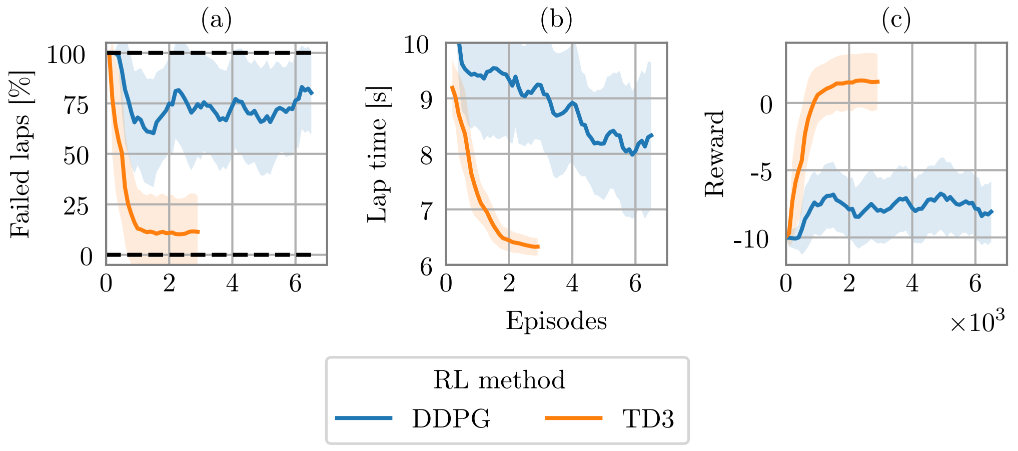


Figure 2.9: (a) The percentage failed laps and (b) lap time of completed laps during training, as well as (c) the learning curves of end-to-end agents that were trained using TD3 and DDPG.

2.5 Reward signal

Having experimentally determined optimal values for the TD3 hyper-parameters, we investigated the reward signal. Our objectives were to choose reward signal parameter values that yielded agents that (a) race safety while (b) minimising lap time. This was a challenging task, considering that these two objectives are in conflict with each other. Further complicating the task is that the lap time alone is too sparse a signal to allow the agent to learn effectively [7; 8].

The reward signal described in Equation 2.3 rewards the agent for the distance it travelled along the centerline between the current and previous time step, and penalises the agent a small amount on every time step [5]. Additionally, the agent receives a large penalty for colliding with the track boundary. We now present the tuning procedure for (a) the time step penalty r_{time}, (b) the distance reward r_{dist} and (c) the collision penalty $r_{\text{collision}}$.

To motivate the use of a time step penalty r_{time}, we examine the total reward accumulated over a successful episode,

$$R_{\text{total}} = \sum_{t=1}^{T} \left(r_{\text{dist}}(D_t - D_{t-1}) + r_{\text{time}} \right), \tag{2.4}$$

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

$$\begin{aligned} R_{\text{total}} &= r_{\text{dist}} \left((D_1 - 0) + \ldots + (D_T - D_{T-1}) \right) + \sum_{t=1}^{T} r_{\text{time}} \\ &= r_{\text{dist}} D_T + T r_{\text{time}}. \end{aligned} \tag{2.5}$$

To simplify the expression for total reward, r_{time} was set equal to -0.01, which is the negative of the simulator time step Δt. Additionally, T was substituted as

$$T = \frac{\text{lap time}}{\Delta t}. \tag{2.6}$$

By substituting Equation 2.6 into Equation 2.4, we get R_{total} as

$$R_{\text{total}} = r_{\text{dist}} D_T - \text{lap time}. \tag{2.7}$$

This equation shows that the reward signal from Equation 2.3 approximates minimising lap time for sufficiently large reward discount factors. Because D_T is a constant, it is clear that the agent must minimise lap time to maximise the total reward. Furthermore, if no time step penalty is applied, then every successful lap yields the same reward regardless of lap time.

We then trained and evaluated agents with and without the time step penalty, while setting the reward signal components r_{dist} and $r_{\text{collision}}$ to 0.25 and -10, respectively. Setting r_{time} to -0.01 improved the average evaluation lap time of agents from 9.26 seconds to 6.07 seconds, compared to agents that did not receive the penalty.

Next, we experimentally tuned the distance reward value relative to the r_{time} value of -0.01. We initially determined a plausible range of r_{dist} values to train our agents with.

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

$$r_{\text{dist}}(D_t - D_{t-1}) + r_{\text{time}} > 0. \tag{2.8}$$

Solving this inequality for r_{dist} gives us

$$r_{\text{dist}} > \frac{-r_{\text{time}}}{(D_t - D_{t-1})}, \tag{2.9}$$

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

$$(D_t - D_{t-1}) = v_{\text{max}}\Delta t. \tag{2.10}$$

After substituting this expression into Equation 2.9 and setting r_{time} equal to Δ_t, the minimum value for r_{distance} is found to be

$$r_{\text{dist}} > \frac{1}{v_{\text{max}}}. \tag{2.11}$$

Substituting the value for v_{max} as 5 m/s into Equation 2.11 yields an estimated minimum r_{dist} of 0.2.

Using this value as a guide for the region in which to search for r_{dist}, we trained agents with r_{dist} values of 0.1, 0.25, 0.3 and 1. The percentage failed laps and lap time of completed laps during training for these agents are shown in Figure 2.10. Unsurprisingly, the agent with r_{dist} set to 0.1 (i.e., less than the estimated minimum) learns that terminating the episode immediately is the optimal behaviour, as its failure rate remains at 100 percent.

Figure 2.10 also reveals that larger values of r_{dist} result in worse performance in terms of failed laps and lap time. When r_{dist} is set to a larger value, the time step penalty becomes less significant. As such, the agent is less incentivised to minimise lap time. Conversely, when r_{dist} is set close to the estimated minimum value, the time step penalty becomes significant, and the agent must optimise lap time to receive positive rewards. The value for r_{dist} was chosen as 0.25, as agents that were trained with this value had the lowest crash rate while also achieving competitive lap times.

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

To identify a suitable range within which we could conduct experimental searches for an optimal value, we operated on the premise that $r_{\text{collision}}$ should be substantial compared to the positive reward an agent can receive in an episode. As shown in Figure 2.4, agents attain an average reward value of 2 in episodes where crashes do not occur. Consequently, we trained agents with collision penalties ranging from -2 to -10. The percentage failed laps and lap times of agents trained with these values for $r_{\text{collision}}$ are presented in Table 2.3. We selected $r_{\text{collision}}$ as -10, as it is the only penalty that resulted in no failed laps.

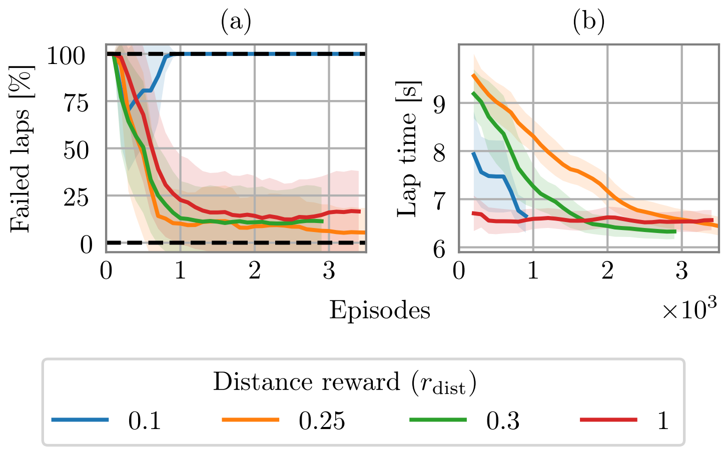


Figure 2.10: (a) The percentage failed laps and (b) lap times of completed laps during training of end-to-end agents with r_{dist} values ranging from 0.1 to 1.

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

Table 2.3: Percentage failed laps and lap times under evaluation conditions for agents trained with $r_{\text{collision}}$ values from ranging from 0 to −10.

Interestingly, the effect of increasing the collision penalty can be seen in the path taken by agent. Figure 2.11 shows the paths taken by agents with $r_{\text{collision}}$ set to −4 and −10. The agent with the lower collision penalty races close to the inside of the track, while the agent that is penalised more heavily takes a much more conservative path by staying clear of the track boundaries, instead preferring to drive near the centerline of the track.

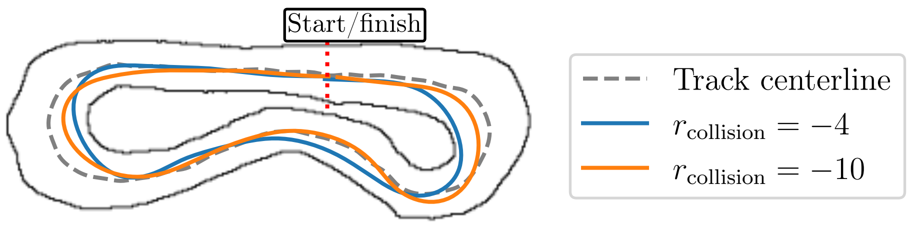


Figure 2.11: The paths taken by agents trained with $r_{\text{collision}}$ values of −4 and −10.

2.6 Observation space

Next, we investigated the optimal hyper-parameters for the observation space. Initially, we determined which observation elements yield agents with the best performance. We therefore trained agents that received (a) only the pose, (b) only a LiDAR scan and (c) a combination of vehicle pose and LiDAR scan. The performance of these agents during training, in terms of percentage failed laps, lap time and reward, is shown in Figure 2.12.

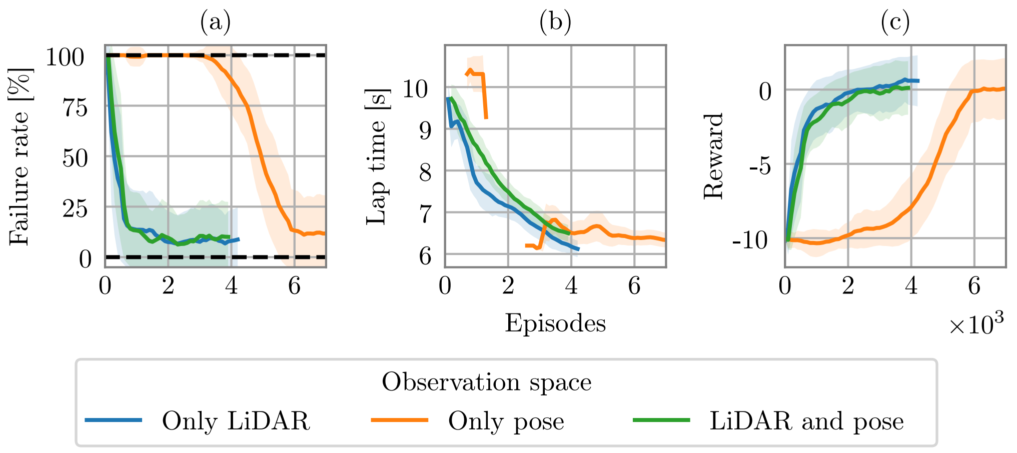
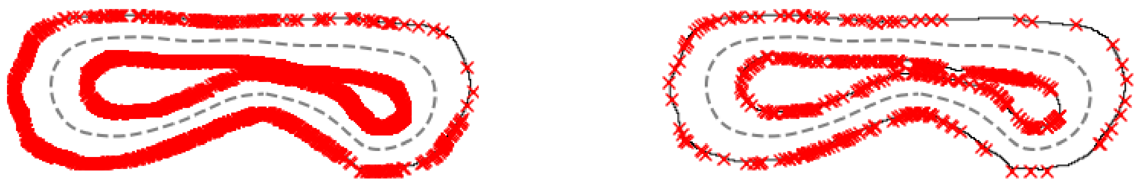


Figure 2.12: (a) the percentage failed laps and (b) lap time of completed laps during training, was well as the (c) learning curves showing episode reward for end-to-end agents with different observation spaces.

From this figure, agents utilising each of the observation spaces converge to a similar values for all three evaluation metrics. However, agents that receive a LiDAR scan train significantly faster than agents without a LiDAR scan in their observation. Specifically, the LiDAR scan allows the agent to learn to avoid track boundaries without needing to sample collision experiences at every point along the track boundary. This is clearly demonstrated in Figure 2.13, which shows all of the locations where an agent with only the pose, and an agent with both pose and LiDAR scan crashed during training. We found that agents without LiDAR scans crashed 5183 times, whereas those with LiDAR scans crashed only 464 times during the same training period.



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

Figure 2.13: The locations where an agent trained with (a) only the pose and (b) both the pose and LiDAR scan crashed during training.

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

We then conducted further experiments to determine the number of beams that should be included in the LiDAR scan. Figure 2.14 displays the percentage failed laps and lap

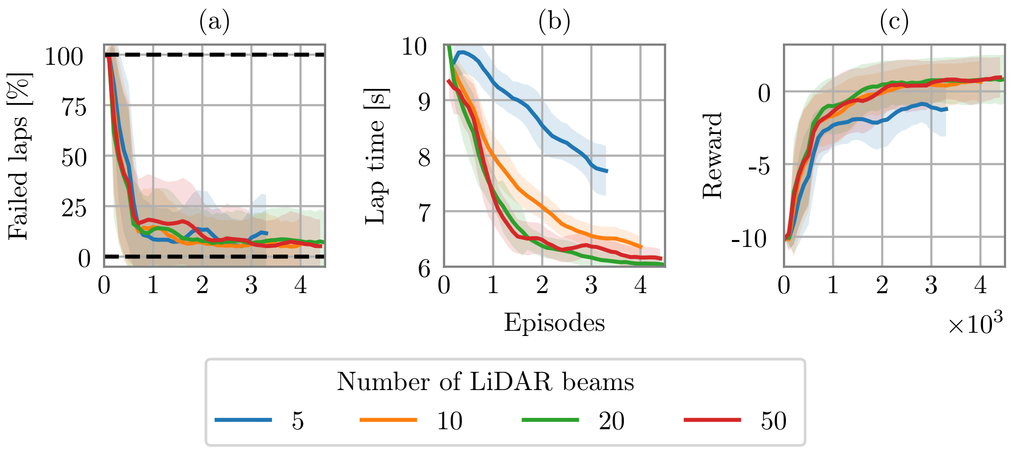


Figure 2.14: (a) the percentage failed laps and (b) lap time of completed laps during training, was well as the (c) learning curves showing episode reward for end-to-end agents with different numbers of LiDAR beams during training.

times during training, as well as the learning curves of agents with LiDAR scans consisting of 5, 10, 20 and 50 equal-spaced beams and a field of view of 180°. Our results indicate that increasing the number of LiDAR beams above 20 does not significantly impact the performance of agents in terms of any of the measured criteria. We therefore chose to incorporate 20 LiDAR into our observation space.

We then conducted experiments to investigate whether incorporating observation noise during training provides any benefits. Specifically, we trained two agents: one without any noise in the observation vector, and another with added Gaussian noise having standard deviations of 0.025 m for x and y coordinates, 0.05 rads for heading, 0.1 m/s for velocity, and 0.01 m for LiDAR scan.

The agents trained with noise achieved an average lap time of 6.77 seconds while completing 98.67% of the laps under evaluation conditions. In comparison, agents trained without noise completed all laps with an average time of 6.09 seconds. It is noteworthy that the agents trained with observation noise completed laps in a more erratic manner than agents trained without noise, as evidenced by the paths and velocity profiles of agents during evaluation, as shown in Figure 2.15. This was despite the presence of noise under evaluation conditions. We therefore chose to train agents without observation noise.

2.7 Neural network hyper-parameters

Next, an investigation was conducted to determine the optimal DNN layer configuration. We therefore changed the layer configuration of each the actor and critics, so that they

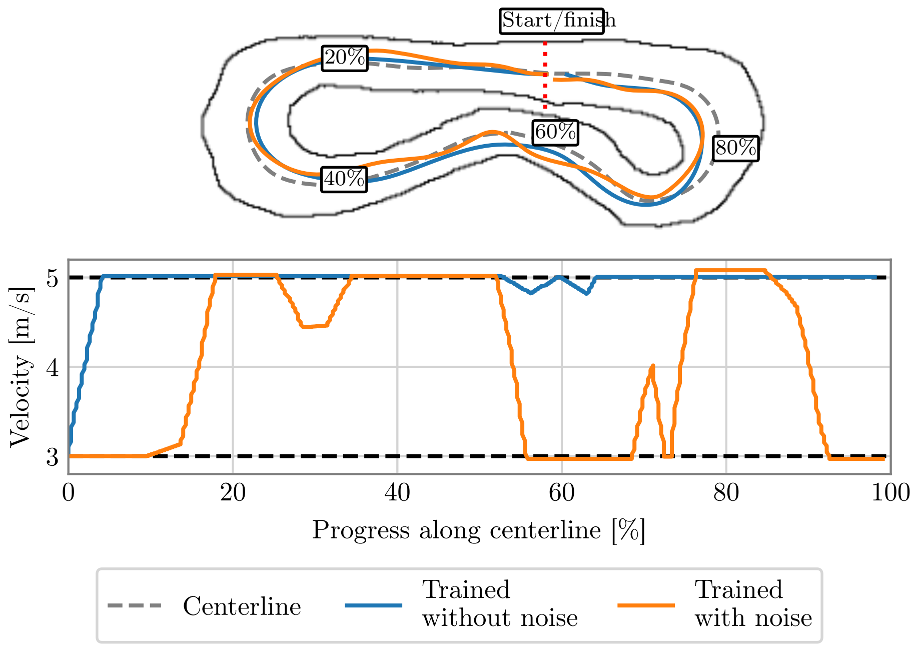


Figure 2.15: Path and velocity profiles of end-to-end agents that were trained with and without noise added to the observation vector.

remained identical in structure. The input and hidden hidden layers of these DNNs were initially specified to be 400 and 300 units, respectively. In this experiment, agents were trained with input and hidden hidden layers that were 100 units larger and smaller than the initial DNN configuration. The percentage failed laps, lap times learning curves for the training of these agents are depicted in Figure 2.16.

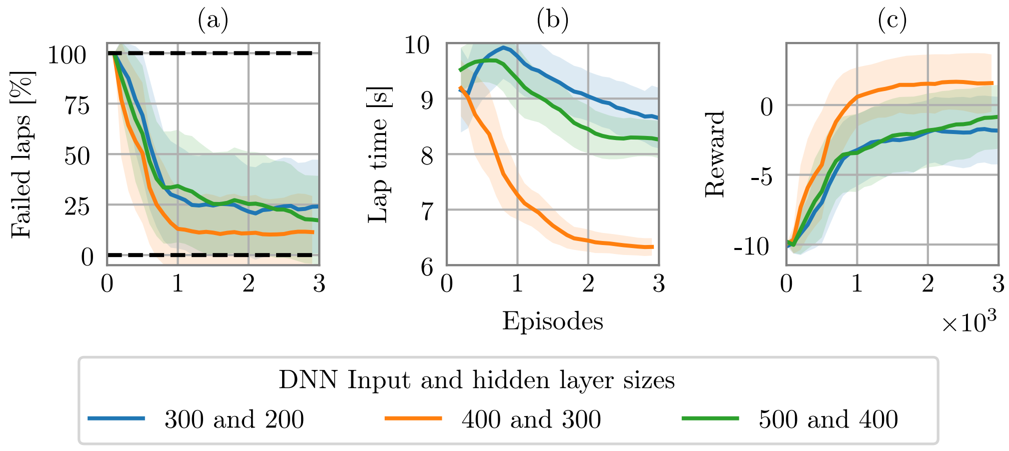


Figure 2.16: (a) the percentage failed laps and (b) lap time of completed laps during training, was well as the (c) learning curves showing episode reward for end-to-end agents with different DNN layer sizes.

The experimental results from Figure 2.16 indicates that increasing or decreasing the number of units in the input and hidden layers led to a deterioration in performance, particularly in terms of lap time. As a result, we have selected an input layer size of 400 units and a hidden layer size of 300 units for the actor and critic DNNs in our algorithm.

We then conducted additional experiments to determine the optimal learning rate α, using the same learning rate for the actor and critic DNNs. Specifically, we trained agents with learning rates of 10^{-4}, 10^{-3} and $2 \cdot 10^{-3}$. Their performance under evaluation conditions in terms of percentage successful laps and lap time is shown in Table 2.4. The percentage successful laps and lap time of agents were maximised and minimised, respectively, when the learning rate was set to 10^{-3}.

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

Table 2.4: Evaluation results of end-to-end agents with actor and critic DNN learning rates between 10^{-4} and $2 \cdot 10^{-3}$.

2.8 Velocity constraint

In our final hyper-parameter tuning investigation, we conducted experiments to determine the minimum and maximum allowable velocities, which is a common technique to ensure safe operation of the vehicle. For example, Ivanov et al. [6] restrict the torque applied to the vehicle's driving motors, thereby limiting its maximum speed to 2.4 m/s. Hsu et al. [11] adopt less conservative bounds, enforcing minimum and maximum speed limits of 1.125 and 9.3 m/s, respectively.

To determine the maximum safe velocity, we trained and evaluated the behaviour of agents with $v_{\max}$ values of 5, 6, 7 and 8 m/s. Figure 2.17 illustrates the velocity and slip angle profiles of the agents as they complete one lap under evaluation conditions. Interestingly, we observed that the agents tended to maintain the maximum velocity around the track. This behaviour likely occurs because even small values of $a_{\text{long},d}$ result in large changes in velocity in-between agent samples. Further exacerbating this effect is the slow rate at which actions can be sampled from the agent.

We observed from Figure 2.17 that agents with a maximum velocity greater than 5 m/s experienced slip angles larger than 0.2 radians, which is considered both dangerous and unrealistic drifting behavior. Furthermore, the assumption that tire stiffness varies linearly with lateral force only being valid for slip angles below 0.2 radians [17]. Allowing the agent to select large velocities enables it to exploit the simulation unrealistically to achieve fast lap times. Therefore, the vehicle's maximum speed was set to 5 m/s, which was the fastest velocity that did not result in the agent driving dangerously and exploiting the simulator by operating the car at large slip angles.

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

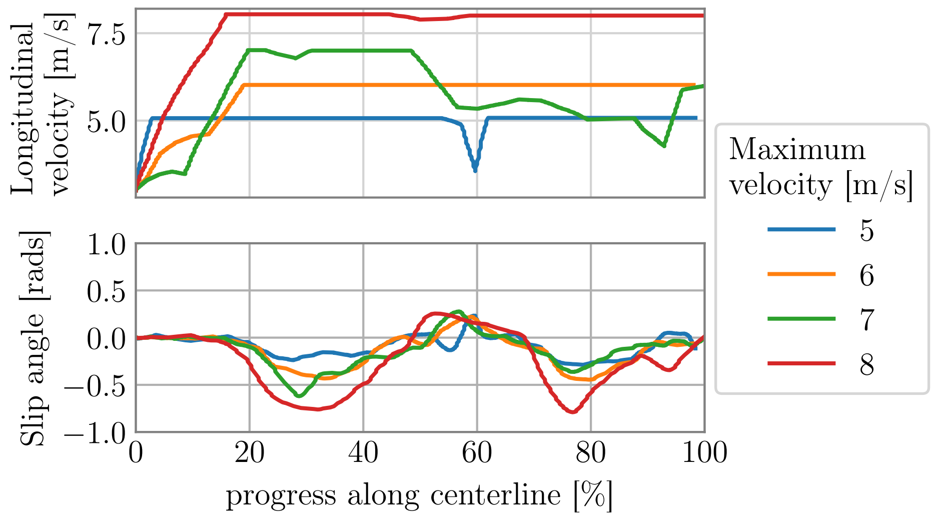


Figure 2.17: The velocity profile and slip angle of agents with different maximum velocities during one test lap.

2.9 End-to-end racing without model uncertainty

Having determined a set of hyper-parameters that yield optimal performance in terms of both safety and lap time for the end-to-end agent racing on the Porto track, we proceeded to evaluate the agent using Algorithm 4. Agents trained using the parameters listed in Table 2.1 demonstrated better performance than any other hyper-parameter set we tested, successfully completing every evaluation lap with an average lap time of 6.03 seconds. Figure 2.18 provides a visualization of an agent's lap, highlighting the path taken with a color map representing the agent's velocity. Notably, the agent maintained maximum velocity for the majority of the track. Nevertheless, the trajectory is smooth and the agent successfully navigated around the Porto circuit.

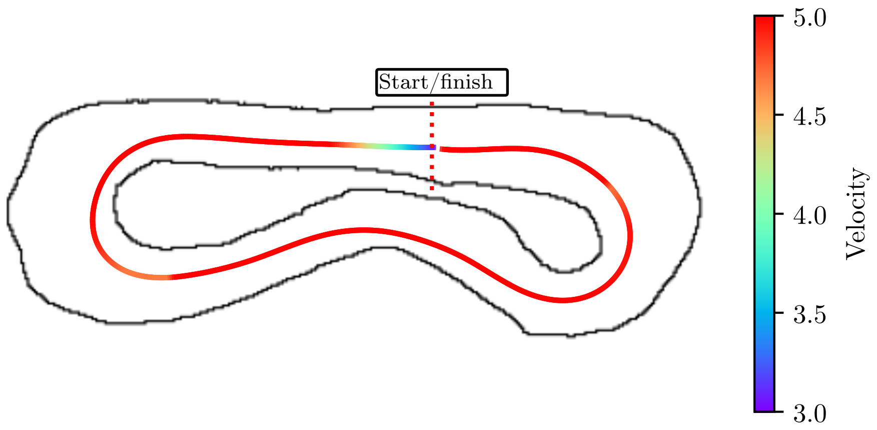


Figure 2.18: The path and velocity profile taken by an end-to-end agent completing Porto in the anti-clockwise direction.

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

We utilized the hyper-parameters selected for the Porto track as an initial estimation when selecting hyperparameters for the larger tracks. Similar to the procedure used for Porto, we conducted a hyper-parameter tuning process on the Circuit de Barcelona-Catalunya, whereby we systematically adjusted one hyper-parameter while keeping the others constant. To achieve satisfactory performance on the Circuit de Barcelona-Catalunya, the following adjustments were made to the hyper-parameters listed in Table 2.1: the number of MDP time steps (M) was increased to $3.5 \cdot 10^5$, agent sample rate (f_{agent}) was increased to 10 Hz, and the reward signal values for r_{dist} and $r_{\text{collision}}$ were changed to 0.3 and -2, respectively. These selected hyper-parameters were also applied to the Circuit de Monaco. The learning curves for agents trained on all three tracks using these hyper-parameters are shown in Figure 2.19. Importantly, we observe that these agents maximise reward on each of their respective tracks.

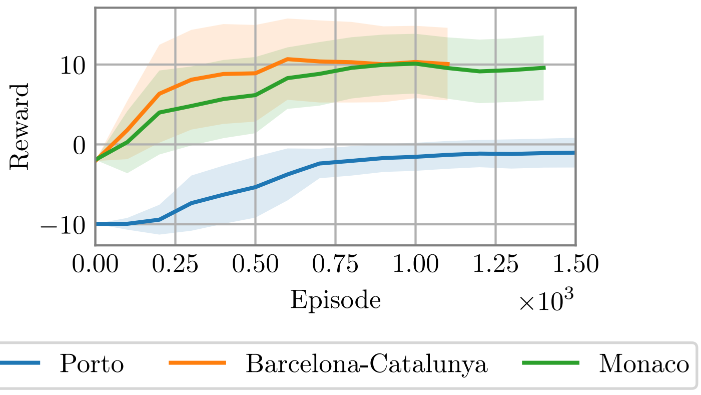


Figure 2.19: Learning curves for end-to-end agents trained and tested on Porto, Circuit de Barcelona-Catalunya and Circuit de Monaco.

Agents completed the Circuit de Barcelona-Catalunya 79.0% of the time and achieved an average lap time of 47.47 seconds under evaluation conditions. Figure 2.20 shows the path and velocity profile taken by an agent completing Circuit de Barcelona-Catalunya under evaluation conditions. Similar to the findings on the Porto track, agents racing on the Circuit de Barcelona-Catalunya selected maximum velocity for the majority of the track, even when navigating sharp corners. However, an interesting phenomenon emerged on the Circuit de Barcelona-Catalunya that was not present on the shorter Porto track: agents tended to exhibit a slaloming behavior, which was characterized by a winding path. This slaloming effect was quite severe, occurring at nearly every section of the track.

We proceeded to assess the performance of agents trained on the Circuit de Monaco. These agents accomplished successful lap completions for 61.67% of their attempts, achieving an average lap time of 35.48 seconds. Figure 2.20 depicts the path and velocity profile of an agent successfully completing the Circuit de Monaco under evaluation conditions. Interestingly, the slaloming was also present on the Circuit de Monaco, indicating that slaloming tends to be a common issue for end-to-end agents navigating long tracks.

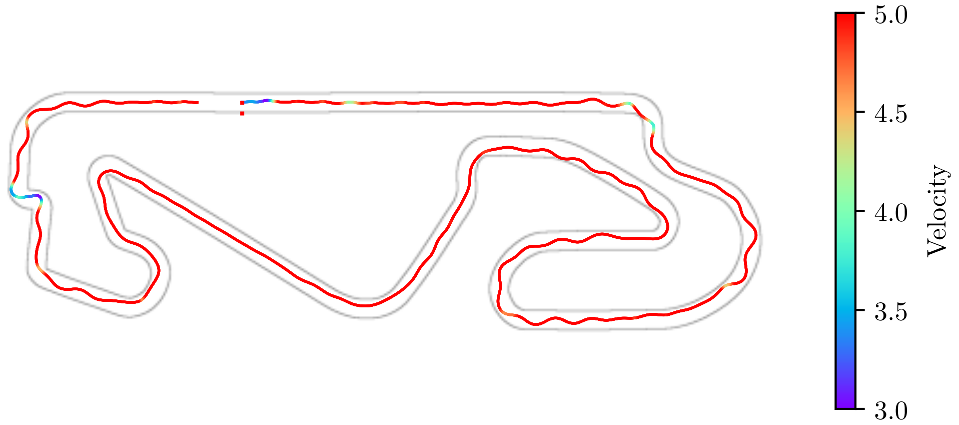


Figure 2.20: The path and velocity profile taken by an end-to-end agent completing Circuit de Barcelona-Catalunya

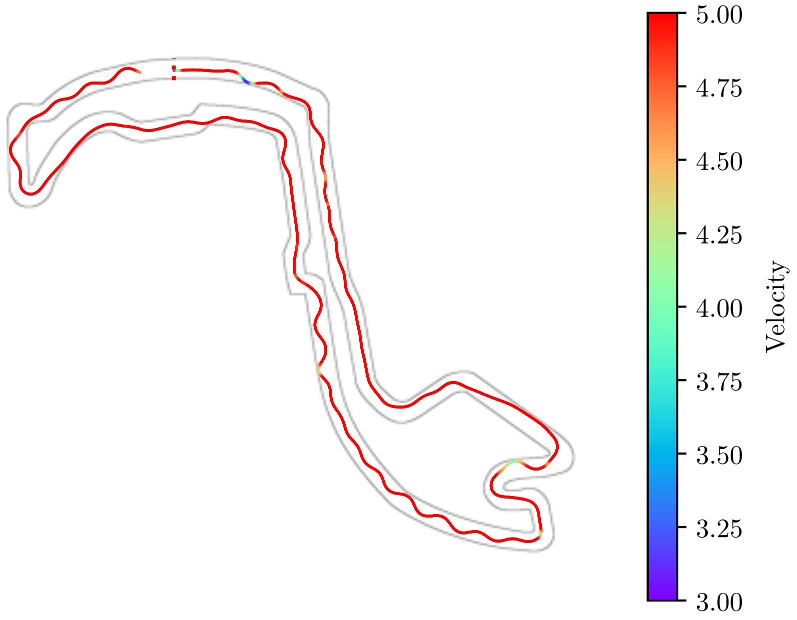
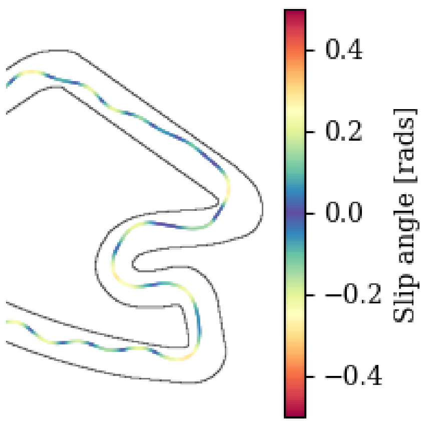


Figure 2.21: The path and velocity profile taken by an end-to-end agent completing Circuit de Monaco

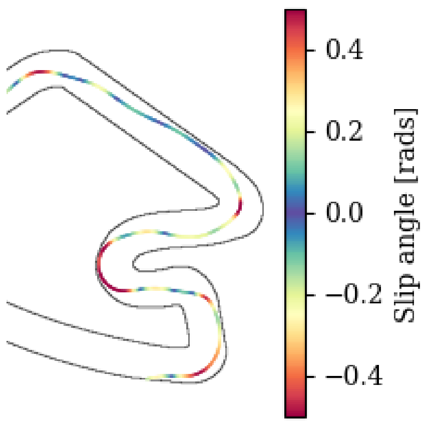
2.10 End-to-end racing with model uncertainty

Up to now, results have been presented for end-to-end agents that were trained and evaluated in identical environments. However, it is important to assess the performance of agents tasked with driving in situations where the vehicle model does not match the one utilised during training, commonly known as model mismatch. We introduced model mismatches by modifying the vehicle model parameters prior to executing the evaluation process outlined in Algorithm 4. This adjustment allows us to gain insights into how the agent performs in a more realistic setting where variations in the vehicle model are present.

Our initial focus was on investigating the impact of altering the road surface friction coefficient. This parameter is influenced by various dynamic factors, including temperature and precipitation, making it challenging to predict accurately. Consequently, it is likely that model mismatches in the road friction coefficient occur. Figure 2.22 presents a comparison between agents evaluated with a friction value of 0.6, equivalent to wet asphalt conditions, and agents racing with the nominal friction value of 1.04 on a section of the Monaco track. The slip angles of the agents are visualized by color-mapping them onto their respective paths. When evaluated with the nominal friction value, the agents display slaloming behavior, resulting in maximum slip angles of approximately 0.2 radians throughout most areas of the track. In contrast, agents evaluated with decreased friction exhibit drifting behavior, characterized by slip angles exceeding 0.4 radians.



(a)



(b)

Figure 2.22: Trajectory and slip angle of an end-to-end agent racing on Circuit de Monaco with (a) the nominal road-surface friction value of 1.03, and (b) a decreased road-surface friction value of 0.6.

Notably, Figure 2.22 illustrates an instance where an agent with decreased friction crashes shortly after executing a drift maneuver. This observation emphasizes the impact of friction on the agents' handling capabilities and reinforces the significance of creating algorithms that are robust to errors in the vehicle model parameters.

2.11 Training with domain randomisation

A commonly used technique to enhance robustness against modelling errors is domain randomisation, which involves randomising simulation parameters during training. The agent is then tasked with finding a single policy that performs optimally across different parameter settings [18]. Previous autonomous racing studies have explored various approaches in this regard. Chisari et al. [12] introduced Gaussian noise to the lateral force experienced by the tires at each time step, while Ghignone et al. [19] initialized each episode by adding Gaussian noise with a standard deviation of 0.0375 to the road surface friction coefficient, which remained constant throughout the episode.

In this experiment, we adopted the approach of Ghignone et al. [19] and trained agents with Gaussian noise added to the nominal friction coefficient of 1.0489 at the start of each episode. Agents were trained to race on Porto using this method with standard deviations of 0.01 and 0.05 respectively. These agents were then evaluated using the nominal friction coefficient.

While agents trained with standard deviation of 0.01 successfully completed 51% of their laps, agents trained with a standard deviation of 0.05 completed only 34% of their laps under evaluation conditions. These results indicate that domain randomisation has an adverse effect on the agents' performance.

Figure 2.23 illustrates the paths taken by these agents during evaluation, clearly indicating their inability to learn smooth driving behavior. Instead, the agents exhibited a tendency to slalom, resulting in sub-optimal performance.

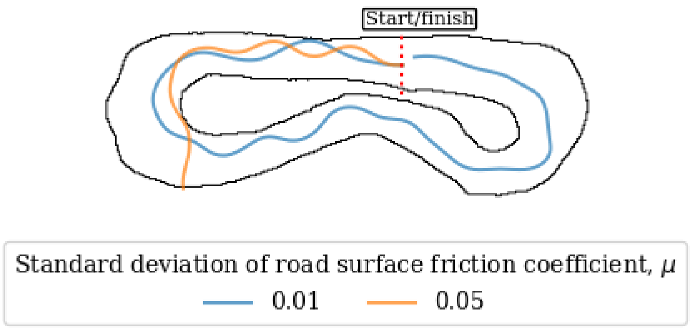


Figure 2.23: Paths taken by agents trained with randomised road-surface friction coefficients on the Porto track under evaluation conditions. The friction coefficient was set to the nominal value of 1.0489 during evaluation.

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

2.12 Summary

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

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

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

Appendices

Appendix A

Supporting results

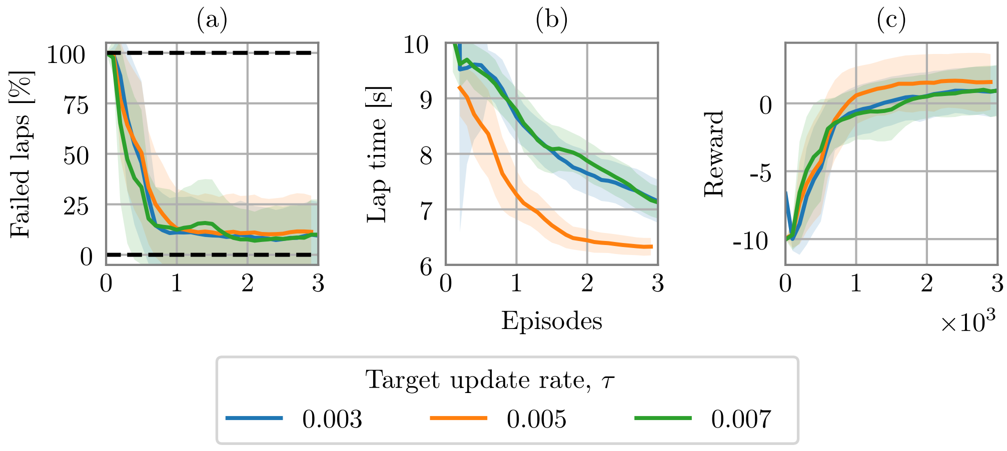


Figure A.1: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with target update rates ranging from 0.003 to 0.007.

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

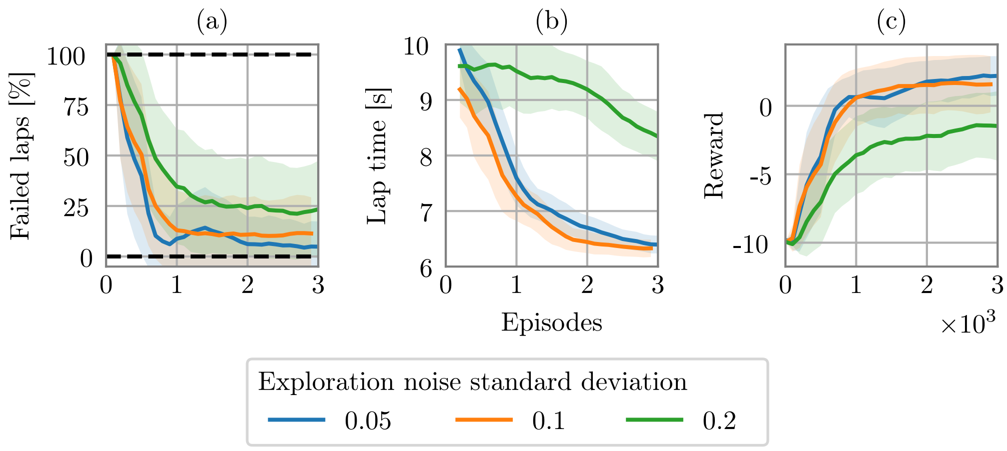


Figure A.2: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with exploration noise standard deviations ranging from 0.05 to 0.2.

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.

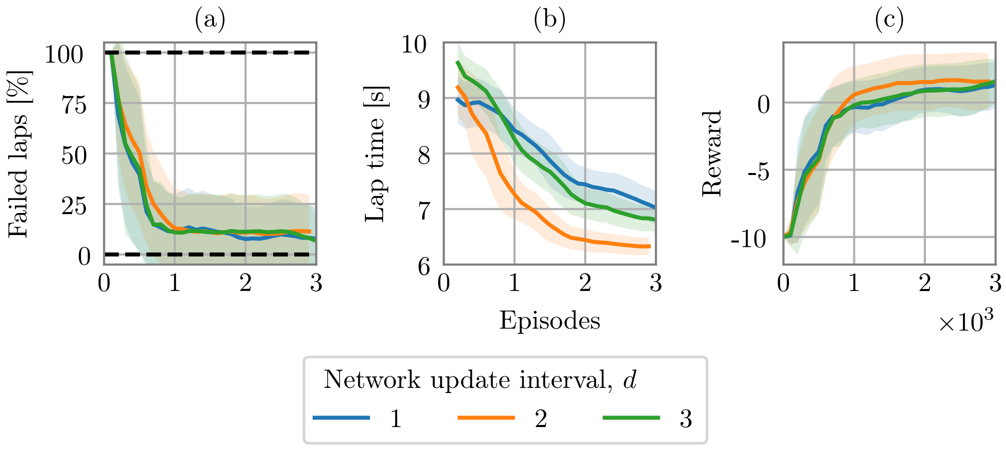


Figure A.3: Learning curves showing (a) the failure rate, i.e percentage of episodes that ended in a crash, (b) the lap time of completed laps, and (b) the episode reward for end-to-end agents with network update intervals d ranging from 1 to 3.

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

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

[3] 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`

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

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

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

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

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

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

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

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

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

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

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

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

[16] 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`

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

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

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